```java
// Collection classes were created to make it easy to keep track
// of groups of objects. An ArrayList differs from an array in
// that it automatically resizes itself automatically. ArrayLists
// are easy to add to and delete from.

import java.util.ArrayList; // The ArrayList library
import java.util.Iterator; // The Iterator Library
import java.util.Arrays; // The Arrays Library

public class LessonEleven {

    public static void main(String[] args)
    {

        // You can create an ArrayList variable
        ArrayList arrayListOne;

        // Then create an ArrayList object
        // You don't have to declare the ArrayList size like you
        // do with arrays (Default Size of 10)
        arrayListOne = new ArrayList();

        // You can create the ArrayList on one line

        ArrayList arrayListTwo = new ArrayList();

        // You can also define the type of elements the ArrayList
        // will hold
        ArrayList<String> names = new ArrayList<String>();

        // This is how you add elements to an ArrayList
        names.add("John Smith");
        names.add("Mohamed Alami");
        names.add("Oliver Miller");

        // You can also add an element in a specific position
        names.add(2, "Jack Ryan");

        // You retrieve values in an ArrayList with get
        // arrayListName.size() returns the size of the ArrayList
        for( int i = 0; i < names.size(); i++)
        {
            System.out.println(names.get(i));
        }

        // You can replace a value using the set method
        names.set(0, "John Adams");

        // You can remove an item with remove
        names.remove(3);

        // You can also remove the first and second item with
        // the removeRange method
        // names.removeRange(0, 1);

        // When you print out the ArrayList itself the toString
```

```
057         // method is called
058         System.out.println(names);
059 
060         // You can also use the enhanced for with an ArrayList
061         for(String i : names)
062         {
063             System.out.println(i);
064         }
065         System.out.println(); // Creates a newline
066 
067         // Before the enhanced for you had to use an iterator
068         // to print out values in an ArrayList
069 
070         // Creates an iterator object with methods that allow
071         // you to iterate through the values in the ArrayList
072         Iterator indivItems = names.iterator();
073 
074         // When hasNext is called it returns true or false
075         // depending on whether there are more items in the list
076 
077         while(indivItems.hasNext())
078         {
079             // next retrieves the next item in the ArrayList
080             System.out.println(indivItems.next());
081 
082         }
083 
084         // I create an ArrayList without stating the type of values
085         // it contains (Default is Object)
086         ArrayList nameCopy = new ArrayList();
087         ArrayList nameBackup = new ArrayList();
088 
089         // addAll adds everything in one ArrayList to another
090         nameCopy.addAll(names);
091         System.out.println(nameCopy);
092 
093         String paulYoung = "Paul Young";
094 
095         // You can add variable values to an ArrayList
096         names.add(paulYoung);
097 
098         // contains returns a boolean value based off of whether
099         // the ArrayList contains the specified object
100 
101         if(names.contains(paulYoung))
102         {
103             System.out.println("Paul is here");
104         }
105 
106         // containsAll checks if everything in one ArrayList is in
107         // another ArrayList
108         if(names.containsAll(nameCopy))
109         {
110             System.out.println("Everything in nameCopy is in
    names");
111         }
112 
```

```
113         // Clear deletes everything in the ArrayList
114         names.clear();
115 
116         // isEmpty returns a boolean value based on if the
    ArrayList
117         // is empty
118         if (names.isEmpty())
119         {
120 
121             System.out.println("The ArrayList is empty");
122 
123         }
124 
125         Object[] moreNames = new Object[4];
126 
127         // toArray converts the ArrayList into an array of objects
128         moreNames = nameCopy.toArray();
129 
130         // toString converts items in the array into a String
131         System.out.println(Arrays.toString(moreNames));
132 
133 
134     }
135 
136 }
```

```
001 import java.util.Arrays;
002
003 public class LessonNine {
004
005     public static void main(String[] args){
006
007         // An array is a variable that can hold a bunch of values
008         // Think of an array as a big box filled with other boxes
009         // Each box has a number on it called an index that you use
010         // to access its specific value
011
012         /* Array Rules
013          * An array can contain only values of the same type
014          * An arrays size can't be changed once it is set
015          * An array is an object
016          */
017
018         // You declare an array with the dataType[] arrayName
019         int[] randomArray;
020
021         // You create an array with
022         // dataType[] arrayName = new dataType[sizeOfArray];
023
024         int[] numberArray = new int[10];
025
026         // You can add values to the array in many ways
027         // Individually
028
029         // Most create array and define size first
030         randomArray = new int[20];
031         randomArray[1] = 2;
032
033         // You can also create the array and its values from the
    start
034         String[] stringArray = {"Just", "Random", "Words"};
035
036         // You can add values with a loop
037         // arrayName.length returns the number of elements in the
    array
038         for(int i = 0; i < numberArray.length; i++)
039         {
040
041             numberArray[i] = i;
042
043         }
044
045         // Draws 41 lines on the screen
046         int k = 1;
047         while(k <= 41){ System.out.print('-'); k++; }
048         System.out.println();
049
050         // Cycles through all of the boxes in the array and prints
    them
051         for(int j = 0; j < numberArray.length; j++)
052         {
053             System.out.print("| " + j + " ");
```

```
054         }
055         System.out.println("|");
056 
057         // Draws 41 lines on the screen
058         k = 1;
059         while(k <= 41){ System.out.print('-'); k++; }
060         System.out.println();
061 
062         // Multidimensional Array
063         // To but arrays in an array just add another []
064 
065         String[][] multiDArray = new String[10][10];
066 
067         // Adding values to a multidimensional array
068 
069         for(int i = 0; i < multiDArray.length; i++)
070         {
071 
072             // To get the length for the array in the array you
    must follow it
073             // with brackets with the index between them like [i]
074 
075             for(int j = 0; j < multiDArray[i].length; j++)
076             {
077 
078                 multiDArray[i][j] = i + " " + j;
079 
080             }
081 
082         }
083 
084         // Draws 61 lines on the screen
085         k = 1;
086         while(k <= 61){ System.out.print('-'); k++; }
087         System.out.println();
088 
089         // Prints out a multidimensional array with the values
    being the indexes
090 
091         for(int i = 0; i < multiDArray.length; i++)
092         {
093 
094             for(int j = 0; j < multiDArray[i].length; j++)
095             {
096 
097                 System.out.print("| " + multiDArray[i][j] + " ");
098 
099             }
100             System.out.println("|");
101 
102         }
103 
104         // Draws 61 lines on the screen
105         k = 1;
106         while(k <= 61){ System.out.print('-'); k++; }
107         System.out.println();
108 
```

```
109         // You can use the enhanced for loop to print out array
    values
110         // for(itemDataType tempVariable : arrayName)
111
112         for(int row : numberArray)
113         {
114             System.out.print(row);
115         }
116         System.out.println("\n");
117
118         // To use enhanced for for a multidimensional array you
    follow this formula
119         // for(dataType[] varForRow : arrayName)
120
121         for(String[] rows : multiDArray)
122         {
123             // for(elementDataType varForColumn : varForRow)
124             for(String column : rows)
125             {
126                 System.out.print("| " + column + " ");
127             }
128                 System.out.println("|");
129         }
130
131         // You can copy an array in a couple of ways
132         // Arrays.copyOf(arrayToCopy, numberToCopyFromBeginning);
133
134         int[] numberCopy = Arrays.copyOf(numberArray, 5);
135         for(int num : numberCopy)
136         {
137             System.out.print(num);
138         }
139         System.out.println("\n");
140
141         // You can copy an array from one index to another with
    copyOfRange
142         // int[] numberCopy = Arrays.copyOf(numberArray, 1, 5);
143
144         // You can print out the whole array with toString
145         System.out.println(Arrays.toString(numberCopy));
146
147
148         // Do define a default value for an array use fill
149         // Arrays.fill(arrayName, valueToFill);
150         // valueToFill must be the same for each element in the
    array
151
152         int[] moreNumbers = new int[100];
153         Arrays.fill(moreNumbers, 2);
154
155         // Filling a multidimensional array
156         char[][] boardGame = new char[10][10];
157         for(char[] row : boardGame)
158         {
159             Arrays.fill(row, '*');
160         }
161
```

```
162         // You can sort an array using sort()
163         int[] numsToSort = new int[10];
164 
165         // Generate array full of random numbers
166         for(int i = 0; i < 10; i++)
167         {
168             numsToSort[i] = (int) (Math.random() * 100);
169         }
170 
171         // Sort the array in ascending order
172         Arrays.sort(numsToSort);
173 
174         System.out.println(Arrays.toString(numsToSort));
175 
176         // binarySearch returns the index for the searched for
    value
177         // If it doesn't find it it returns a negative number
178 
179         int whereIs50 = Arrays.binarySearch(numsToSort, 50);
180 
181         System.out.println(whereIs50);
182     }
183 
184 }
```

```
001 import java.io.*;
002
003 // A binary stream is a series of data type values
004 // To read and write to them you use different methods
005 // based on the type of data that you are using
006
007 public class Lesson33{
008
009     public static void main(String[] args){
010
011         // Create an array of type Customer
012
013         Customer[] customers = getCustomers();
014
015         // A DataOutputStream allows you to print
016         // primitive data types to a file
017
018         DataOutputStream custOutput =
    createFile("/Users/derekbanas/Documents/workspace3/Java
    Code/src/customers.dat");
019
020         // Enhanced for loop for arrays
021         // Cycles through all of the people in the customers array
022
023         for(Customer person : customers){
024
025             createCustomers(person, custOutput);
026
027         }
028
029         // Closes the connection to the DataOutputStream
030
031         try {
032             custOutput.close();
033         } catch (IOException e) {
034
035             System.out.println("An I/O Error Occurred");
036
037             // Closes the program
038
039             System.exit(0);
040         }
041
042         getFileInfo();
043
044     }
045
046     // class that defines all the fields for my customers
047
048     private static class Customer{
049
050         public String custName;
051         public int custAge;
052         public double custDebt;
053         public boolean oweMoney;
054         public char custSex;
```

```
055
056         // constructor that's called when a customer is made
057
058         public Customer(String custName, int custAge, double
    custDebt, boolean oweMoney, char custSex){
059
060             this.custName = custName; // String
061             this.custAge = custAge; // Integer
062             this.custDebt = custDebt; // Double
063             this.oweMoney = oweMoney; // Boolean
064             this.custSex = custSex; // Character
065
066         }
067
068     }
069
070     // Creates an array of Customer Objects
071
072     private static Customer[] getCustomers(){
073
074         Customer[] customers = new Customer[5];
075
076         customers[0] = new Customer("John Smith", 21, 12.25, true,
    'M');
077         customers[1] = new Customer("Sally Smith", 30, 2.25, true,
    'F');
078         customers[2] = new Customer("Paul Ryan", 21, 0, false,
    'M');
079         customers[3] = new Customer("Mark Jacobs", 21, 3.25, true,
    'M');
080         customers[4] = new Customer("Steve Nash", 21, 5.25, true,
    'M');
081
082         return customers;
083
084     }
085
086     // Create the file and the DataOutputStream that will write to
    the file
087
088     private static DataOutputStream createFile(String fileName){
089
090         try{
091
092             // Creates a File object that allows you to work with
    files
093             // on the hard drive. There is no difference between
    File
094             // for character or binary stream writing, or reading
095
096             File listOfNames = new File(fileName);
097
098             // FileOutputStream is used to write streams of data to
    a file
099             // You define whether a new file is created versus
    appended
100             // to based on if you add a boolean to the
```

```
    FileOutputStream
101             // FileOutputStream(file, true) : Appends to the file
102             // FileOutputStream(file, false) : Creates a new file
103
104             // BufferedOutputStream gathers all the data and then
    writes
105             // it all at one time (Speeds up the Program)
106             // DataOutputStream is used to write primitive data to
    the file
107
108             DataOutputStream infoToWrite = new DataOutputStream(
109             new BufferedOutputStream(
110                     new FileOutputStream(listOfNames)));
111             return infoToWrite;
112         }
113
114         // You have to catch this when you call FileWriter
115
116         catch(IOException e){
117
118             System.out.println("An I/O Error Occurred");
119
120             // Closes the program
121
122             System.exit(0);
123
124         }
125         return null;
126
127     }
128
129     // Create a string with the customer info and write it to the
    file
130
131     private static void createCustomers(Customer customer,
    DataOutputStream custOutput){
132
133         try{
134         // Write primitive data to the file
135
136         // Writes a String in UTF format
137         custOutput.writeUTF(customer.custName);
138
139         // Writes an Integer
140         custOutput.writeInt(customer.custAge);
141
142         // Writes a Double
143         custOutput.writeDouble(customer.custDebt);
144
145         // Writes a Boolean
146         custOutput.writeBoolean(customer.oweMoney);
147
148         // Writes a Character
149         custOutput.writeChar(customer.custSex);
150
151         // You also have writeByte, writeFloat, writeLong
152         // and writeShort
```

```
153         }
154 
155         catch(IOException e){
156 
157             System.out.println("An I/O Error Occurred");
158             System.exit(0);
159 
160         }
161 
162     }
163 
164     // Read info from the file and write it to the screen
165 
166     private static void getFileInfo(){
167 
168         System.out.println("Info Written to File\n");
169 
170         // Open a new connection to the file
171 
172         File listOfNames = new
    File("/Users/derekbanas/Documents/workspace3/Java
    Code/src/customers.dat");
173 
174         boolean eof = false;
175 
176         try {
177 
178             // A DataInputStream object has the methods for reading
    the data
179             // The BufferedInputStream gathers the data in blocks
180             // FileInputStream gets data from the file
181 
182             DataInputStream getInfo = new DataInputStream(
183                     new BufferedInputStream(
184                     new FileInputStream(listOfNames)));
185 
186             // Using a while loop that pulls data until
    EOFException is thrown
187 
188             while (!eof){
189 
190                 // You have to read data in the exact order it was
    put in the file
191 
192                 String custName = getInfo.readUTF();
193                 int custAge = getInfo.readInt();
194                 double custDebt = getInfo.readDouble();
195                 boolean oweMoney = getInfo.readBoolean();
196                 char custSex = getInfo.readChar();
197 
198                 System.out.println(custName);
199                 System.out.println(custAge);
200                 System.out.println(custDebt);
201                 System.out.println(oweMoney);
202                 System.out.println(custSex + "\n");
203 
204             }
```

```
205
206         } // END OF TRY
207
208         catch (EOFException e) {
209
210             eof = true;
211         }
212
213         // Can be thrown by FileInputStream
214
215         catch (FileNotFoundException e) {
216
217             System.out.println("Couldn't Find the File");
218             System.exit(0);
219         }
220
221         catch(IOException e){
222
223             System.out.println("An I/O Error Occurred");
224             System.exit(0);
225
226         }
227
228     }
229
230
231 }
```

```
001 import javax.swing.*;
002
003 import java.awt.event.*;
004
005 // New event listener that monitors changing values for components
006
007 import javax.swing.event.ChangeEvent;
008 import javax.swing.event.ChangeListener;
009
010 // Allows me to format the numbers
011
012 import java.text.NumberFormat;
013
014 // Allows me to edit borders on panels
015
016 import javax.swing.border.*;
017
018
019 public class Lesson22 extends JFrame{
020
021     JButton button1;
022     JLabel label1, label2, label3;
023     JTextField textField1, textField2;
024     JCheckBox dollarSign, commaSeparator;
025     JRadioButton addNums, subtractNums, multNums, divideNums;
026     JSlider howManyTimes;
027
028     double number1, number2, totalCalc;
029
030     public static void main(String[] args){
031
032         new Lesson22();
033
034     }
035
036     public Lesson22(){
037
038         // Define the size of the frame
039
040         this.setSize(400, 400);
041
042         // Opens the frame in the middle of the screen
043
044         this.setLocationRelativeTo(null);
045
046         // Define how the frame exits (Click the Close Button)
047
048         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
049
050         // Define the title for the frame
051
052         this.setTitle("My Third Frame");
053
054         // The JPanel contains all of the components for your frame
055
056         JPanel thePanel = new JPanel();
```

```
057
058         // ----------------------------------------------------
    -
059
060         // Create a button with Click Here on it
061
062         button1 = new JButton("Calculate");
063
064         // Create an instance of ListenForEvents to handle events
065
066         ListenForButton lForButton = new ListenForButton();
067
068         // Tell Java that you want to be alerted when an event
069         // occurs on the button
070
071         button1.addActionListener(lForButton);
072
073         thePanel.add(button1);
074
075         // How to add a label -------------------------
076
077         label1 = new JLabel("Number 1");
078
079         thePanel.add(label1);
080
081         // How to add a text field ---------------------
082
083         textField1 = new JTextField("", 5);
084
085         thePanel.add(textField1);
086
087         // How to add a label -------------------------
088
089         label2 = new JLabel("Number 2");
090
091         thePanel.add(label2);
092
093         // How to add a text field ---------------------
094
095         textField2 = new JTextField("", 5);
096
097         thePanel.add(textField2);
098
099         // How to add checkboxes ----------------------
100
101         dollarSign = new JCheckBox("Dollars");
102         commaSeparator = new JCheckBox("Commas");
103
104         thePanel.add(dollarSign);
105
106         // By putting true in here it is selected by default
107
108         thePanel.add(commaSeparator, true);
109
110         // Creates radio buttons with default labels
111
112         addNums = new JRadioButton("Add");
```

```
113         subtractNums = new JRadioButton("Subtract");
114         multNums = new JRadioButton("Multiply");
115         divideNums = new JRadioButton("Divide");
116
117         // Creates a group that will contain radio buttons
118         // You do this so that when 1 is selected the others
119         // are deselected
120
121         ButtonGroup operation = new ButtonGroup();
122
123         // Add radio buttons to the group
124
125         operation.add(addNums);
126         operation.add(subtractNums);
127         operation.add(multNums);
128         operation.add(divideNums);
129
130         // Create a new panel to hold radio buttons
131
132         JPanel operPanel = new JPanel();
133
134         // Surround radio button panel with a border
135         // You can define different types of borders
136         // createEtchedBorder, createLineBorder, createTitledBorder
137         // createLoweredBevelBorder, createRaisedBevelBorder
138
139         Border operBorder =
    BorderFactory.createTitledBorder("Operation");
140
141         // Set the border for the panel
142
143         operPanel.setBorder(operBorder);
144
145         // Add the radio buttons to the panel
146
147         operPanel.add(addNums);
148         operPanel.add(subtractNums);
149         operPanel.add(multNums);
150         operPanel.add(divideNums);
151
152         // Selects the add radio button by default
153
154         addNums.setSelected(true);
155
156         // Add the panel to the main panel
157         // You don't add the group
158
159         thePanel.add(operPanel);
160
161         // How to create a slider ----------------
162
163         label3 = new JLabel("Perform How Many Times?");
164
165         thePanel.add(label3);
166
167         // Creates a slider with a min value of 0 thru 99
168         // and an initial value of 1
```

```
169
170         howManyTimes = new JSlider(0, 99, 1);
171
172         // Defines the minimum space between ticks
173
174         howManyTimes.setMinorTickSpacing(1);
175
176         // Defines the minimum space between major ticks
177
178         howManyTimes.setMajorTickSpacing(10);
179
180         // Says to draw the ticks on the slider
181
182         howManyTimes.setPaintTicks(true);
183
184         // Says to draw the tick labels on the slider
185
186         howManyTimes.setPaintLabels(true);
187
188         // Create an instance of ListenForEvents to handle events
189
190         ListenForSlider lForSlider = new ListenForSlider();
191
192         // Tell Java that you want to be alerted when an event
193         // occurs on the slider
194
195         howManyTimes.addChangeListener(lForSlider);
196
197
198         thePanel.add(howManyTimes);
199
200         this.add(thePanel);
201
202         this.setVisible(true);
203
204         // Gives focus to the textfield
205
206         textField1.requestFocus();
207
208     }
209
210     private class ListenForButton implements ActionListener{
211
212         // This method is called when an event occurs
213
214         public void actionPerformed(ActionEvent e){
215
216             // Check if the source of the event was the button
217
218             if(e.getSource() == button1){
219
220                 // getText returns a String so you have to parse it
221                 // into a double in this situation
222
223                 try{
224                     number1 =
    Double.parseDouble(textField1.getText());
```

```
225                     number2 =
    Double.parseDouble(textField2.getText());
226                 }
227
228                 catch(NumberFormatException excep){
229
230                     // JOptionPane displays a popup on the screen
231                     // (parentComponent, message, title, error
    icon)
232                     // Error Icons: WARNING_MESSAGE,
    QUESTION_MESSAGE, PLAIN_MESSAGE
233
234                     JOptionPane.showMessageDialog(Lesson22.this,
    "Please Enter the Right Info", "Error", JOptionPane.ERROR_MESSAGE);
235                     System.exit(0); // Closes the Java app
236                 }
237
238
239                 if(addNums.isSelected()) { totalCalc =
    addNumbers(number1, number2, howManyTimes.getValue());
240
241                 } else if(subtractNums.isSelected()) { totalCalc =
    subtractNumbers(number1, number2, howManyTimes.getValue());
242
243                 } else if(multNums.isSelected()) { totalCalc =
    multiplyNumbers(number1, number2, howManyTimes.getValue());
244
245                 } else { totalCalc = divideNumbers(number1,
    number2, howManyTimes.getValue()); }
246
247                 // If the dollar is selected in the checkbox print
    the number as currency
248
249                 if(dollarSign.isSelected()) {
250
251                     // Defines that you want to format a number
    with $ and commas
252
253                     NumberFormat numFormat =
    NumberFormat.getCurrencyInstance();
254
255                     JOptionPane.showMessageDialog(Lesson22.this,
    numFormat.format(totalCalc), "Solution",
    JOptionPane.INFORMATION_MESSAGE);
256
257                 }
258
259                 // If the comma is selected in the checkbox print
    the number with commas
260
261                 else if(commaSeparator.isSelected()) {
262
263                     // Defines that you want to format a number
    with commas
264
265                     NumberFormat numFormat =
    NumberFormat.getNumberInstance();
```

```
266
267                     JOptionPane.showMessageDialog(Lesson22.this,
    numFormat.format(totalCalc), "Solution",
    JOptionPane.INFORMATION_MESSAGE);
268
269                 } else {
270
271                 JOptionPane.showMessageDialog(Lesson22.this,
    totalCalc, "Solution", JOptionPane.INFORMATION_MESSAGE);
272
273                 }
274
275             }
276
277         }
278
279     }
280
281     // Implements ActionListener so it can react to events on
    components
282
283         private class ListenForSlider implements ChangeListener{
284
285             @Override
286             public void stateChanged(ChangeEvent e) {
287
288                 // Check if the source of the event was the button
289
290                 if(e.getSource() == howManyTimes){
291
292                     label3.setText("Perform How Many Times? " +
    howManyTimes.getValue() );
293
294
295                 }
296
297             }
298
299
300
301         }
302
303         public static double addNumbers(double number1, double
    number2, int howMany){
304
305             double total = 0;
306
307             int i = 1;
308
309             while(i <= howMany){
310                 total = total + (number1 + number2);
311                 i++;
312             }
313
314             return total;
315
316         }
```

```
317
318         public static double subtractNumbers(double number1, double
    number2, int howMany){
319
320             double total = 0;
321
322             int i = 1;
323
324             while(i <= howMany){
325                 total = total + (number1 - number2);
326                 i++;
327             }
328
329             return total;
330
331         }
332
333         public static double multiplyNumbers(double number1, double
    number2, int howMany){
334
335             double total = 0;
336
337             int i = 1;
338
339             while(i <= howMany){
340                 total = total + (number1 * number2);
341                 i++;
342             }
343
344             return total;
345
346         }
347
348         public static double divideNumbers(double number1, double
    number2, int howMany){
349
350             double total = 0;
351
352             int i = 1;
353
354             while(i <= howMany){
355                 total = total + (number1 / number2);
356                 i++;
357             }
358
359             return total;
360
361         }
362
363 }
```

```
001 // Basic class definition
002 // public means this class can be used by other classes
003 // Class names should begin with a capital letter
004 // A file can't contain two public classes. It can contain classes
    that are not public
005 // If you place class files in the same folder the java compiler
    will be able to find them
006
007 public class Monster{
008
009     // Class Variables or Fields
010     // You declare constants with final
011
012     public final String TOMBSTONE = "Here Lies a Dead monster";
013
014     // private fields are not visible outside of the class
015     private int health = 500;
016     private int attack = 20;
017     private int movement = 2;
018     private int xPosition = 0;
019     private int yPosition = 0;
020     private boolean alive = true;
021
022     // public variables are visible outside of the class
023     // You should have as few as possible public fields
024     public String name = "Big Monster";
025
026     // Class Methods
027     // Accessor Methods are used to get and set the values of
    private fields
028
029     public int getAttack()
030     {
031         return attack;
032     }
033
034     public int getMovement()
035     {
036         return movement;
037     }
038
039     public int getHealth()
040     {
041         return health;
042     }
043
044     // You can create multiple versions using the same method name
045     // Now setHealth can except an attack that contains decimals
046     // When overloading a method you can't just change the return
    type
047     // Focus on creating methods that except different parameters
048
049     public void setHealth(int decreaseHealth)
050     {
051         health = health - decreaseHealth;
052         if (health < 0)
```

```
053         {
054             alive = false;
055         }
056     }
057 
058     public void setHealth(double decreaseHealth)
059     {
060         int intDecreaseHealth = (int) decreaseHealth;
061         health = health - intDecreaseHealth;
062         if (health < 0)
063         {
064             alive = false;
065         }
066     }
067 
068     /* The Constructor
069      * Code that is executed when an object is created from this class definition
070      * The method name is the same as the class
071      * The constructor is only executed once per object
072      * The constructor can't return a value
073      */
074 
075     public Monster(int health, int attack, int movement)
076     {
077         this.health = health;
078         this.attack = attack;
079         this.movement = movement;
080         /* If the attributes had the same names as the class health, attack, movement
081          * You could refer to the objects fields by proceeding them with this
082          * this.health = health;
083          * this.attack = attack;
084          * objectFieldName = attributeFieldName;
085          */
086 
087     }
088 
089     // You can overload constructors like any other method
090     // The following constructor is the one provided by default if you don't create any other constructors
091     // Default Constructor
092 
093     public Monster()
094     {
095 
096     }
097 
098     /* You can use the this keyword to call other constructors
099      * public LessonSeven(int newHealth)
100      * {
101      *      health = newHealth;
102      * }
103      *
104      * public LessonSeven(int newHealth, int newAttack)
105      * {
```

```
106       *      this(newHealth); // Any calls to another constructor
    must occur on the first line
107       *      attack = newAttack;
108       * }
109       */
110
111 }
```

```
001 public class LessonThree {
002
003     public static void main(String[] args)
004     {
005         // Creates a random number between 0 and 50
006         int randomNumber = (int) (Math.random() * 50);
007
008         /* Relational Operators:
009          * Java has 6 relational operators
010          * > : Greater Than
011          * < : Less Than
012          * == : Equal To
013          * != : Not Equal To
014          * >= : Greater Than Or Equal To
015          * <= : Less Than Or Equal To
016          */
017
018         // If randomNumber is less than 25, execute the code
    between {} and then stop checking
019         if (randomNumber < 25)
020         {
021             System.out.println("The random number is less than
    25");
022         }
023
024         // If randomNumber wasn't less than 25, then check if it's
    greater than it. If so, execute the code between {} and then stop
    checking
025         else if (randomNumber > 25)
026         {
027             System.out.println("The random number is greater than
    25");
028         }
029
030         // Checks if randomNumber equals 25
031         else if (randomNumber == 25)
032         {
033             System.out.println("The random number is equal to 25");
034         }
035
036         // Checks if randomNumber is not equal to 25
037         else if (randomNumber != 15)
038         {
039             System.out.println("The random number is not equal to
    15");
040         }
041
042         // Checks if randomNumber is less than or equal to 25
043         else if (randomNumber <= 25)
044         {
045             System.out.println("The random number is less than or
    equal to 25");
046         }
047
048         // Checks if randomNumber is greater than or equal to 25
049         else if (randomNumber >= 25)
```

```
050         {
051             System.out.println("The random number is greater than
    or equal to 25");
052         }
053 
054         // If none of the above were correct print out the random
    number
055         else
056         {
057             System.out.println("The random number is " +
    randomNumber);
058         }
059 
060         // Prints out the random number
061         System.out.println("The random number is " + randomNumber);
062 
063         /* Logical Operators:
064          * Java has 6 logical operators
065          * ! : Converts the boolean value to its right to its
    opposite form ie. true to false
066          * & : Returns true if boolean value on the right and left
    are both true (Always evaluates both boolean values)
067          * && : Returns true if boolean value on the right and left
    are both true (Stops evaluating after first false)
068          * | : Returns true if either boolean value on the right or
    left are true (Always evaluates both boolean values)
069          * || : Returns true if either boolean value on the right
    or left are true (Stops evaluating after first true)
070          * ^ : Returns true if there is 1 true and 1 false boolean
    value on the right or left
071          */
072 
073         if (!(false))
074         {
075             System.out.println("I turned false into true");
076         }
077 
078 
079         if ((false) && (true))
080         {
081             System.out.println("\nBoth are true");
082         }
083 
084         // There is also a & logical operator it checks the second
    boolean result even if the first comes back false
085 
086         if ((true) || (true))
087         {
088             System.out.println("\nAt least 1 are true");
089         }
090 
091         // There is also a | logical operator it checks the second
    boolean result even if the first comes back true
092 
093         if ((true) ^ (false))
094         {
095             System.out.println("\n1 is true and the other false");
```

```
096         }
097
098         int valueOne = 1;
099         int valueTwo = 2;
100
101         // The Conditional or Ternary Operator assigns one or
    another value based on a condition
102         // If true valueOne is assigned to biggestValue. If not
    valueTwo is assigned
103         int biggestValue = (valueOne > valueTwo) ? valueOne :
    valueTwo;
104
105         System.out.println(biggestValue + " is the biggest\n");
106
107         char theGrade = 'B';
108
109         /* When you have a limited number of possible values a
    switch statement makes sense
110          * The switch statement checks the value of theGrade to the
    values that follow case
111          * If it matches it executes the code between {} and then
    break ends the switch statement
112          * default code is executed if there are no matches
113          * You are not required to use the break or default
    statements
114          * The expression must be an int, short, byte, or char
115          */
116         switch (theGrade)
117         {
118         case 'A':
119             System.out.println("Great Job");
120             break; // Ends the switch statement
121         case 'b': // You can use multiple case statements in a row
122         case 'B':
123             System.out.println("Good Job, get an A next time");
124             break;
125         case 'C':
126             System.out.println("OK, but you can do better");
127             break;
128         case 'D':
129             System.out.println("You must work harder");
130             break;
131         default:
132             System.out.println("You failed");
133             break;
134         }
135
136     }
137
138 }
```

```
001         /* An exception object is created when an error occurs
002          * It tells you what error occurred
003          * Here are many of the java exceptions
004          *
005          * java.lang.RuntimeException : exceptions that can be
    thrown during the normal
006          * operation of the Java Virtual Machine. These exceptions
    are your responsibility
007          * as a programmer
008          *
009          * ArithmeticException, ArrayStoreException,
    BufferOverflowException,
010          * BufferUnderflowException, CannotRedoException,
    CannotUndoException,
011          * ClassCastException, CMMException,
    ConcurrentModificationException,
012          * DOMException, EmptyStackException,
    IllegalArgumentException,
013          * IllegalMonitorStateException, IllegalPathStateException,
014          * IllegalStateException, ImagingOpException,
    IndexOutOfBoundsException,
015          * MissingResourceException, NegativeArraySizeException,
    NoSuchElementException,
016          * NullPointerException, ProfileDataException,
    ProviderException,
017          * RasterFormatException, SecurityException,
    SystemException,
018          * UndeclaredThrowableException, UnmodifiableSetException,
019          * UnsupportedOperationException
020          *
021          * java.lang.Exception : exceptions that are checked for by
    the java compiler
022          *
023          * AclNotFoundException, ActivationException,
    AlreadyBoundException,
024          * ApplicationException, AWTException,
    BackingStoreException,
025          * BadAttributeValueExpException,
    BadBinaryOpValueExpException,
026          * BadLocationException, BadStringOperationException,
027          * BrokenBarrierException, CertificateException,
    ClassNotFoundException,
028          * CloneNotSupportedException, DataFormatException,
029          * DatatypeConfigurationException, DestroyFailedException,
030          * ExecutionException, ExpandVetoException,
    FontFormatException,
031          * GeneralSecurityException, GSSException,
    IllegalAccessException,
032          * IllegalClassFormatException, InstantiationException,
033          * InterruptedException, IntrospectionException,
034          * InvalidApplicationException, InvalidMidiDataException,
035          * InvalidPreferencesFormatException,
    InvalidTargetObjectTypeException,
036          * InvocationTargetException, IOException, JAXBException,
    JMException,
037          * KeySelectorException, LastOwnerException,
```

```
    LineUnavailableException,
038          * MarshalException, MidiUnavailableException,
    MimeTypeParseException,
039          * MimeTypeParseException, NamingException,
    NoninvertibleTransformException,
040          * NoSuchFieldException, NoSuchMethodException,
    NotBoundException,
041          * NotOwnerException, ParseException,
    ParserConfigurationException,
042          * PrinterException, PrintException,
    PrivilegedActionException,
043          * PropertyVetoException, RefreshFailedException,
    RemarshalException,
044          * RuntimeException, SAXException, ScriptException,
    ServerNotActiveException,
045          * SOAPException, SQLException, TimeoutException,
    TooManyListenersException,
046          * TransformerException, TransformException,
    UnmodifiableClassException,
047          * UnsupportedAudioFileException,
    UnsupportedCallbackException,
048          * UnsupportedFlavorException,
    UnsupportedLookAndFeelException,
049          * URIReferenceException, URISyntaxException,
    UserException, XAException,
050          * XMLParseException, XMLSignatureException,
    XMLStreamException, XPathException
051          */
052
053         /* Common Exceptions
054          * ArithmeticException: An arithmetic operation occurs with
    no answer
055          * (Division by Zero)
056          *
057          * ClassNotFoundException: A class is called for that
    doesn't exist
058          *
059          * IllegalArgumentException: A method has been passed an
    illegal argument
060          *
061          * IndexOutOfBoundsException: Thrown when an index for an
    array, string is
062          * called for, but doesn't exist
063          *
064          * InputMismatchException: (Part of NoSuchElementException)
    User enters the
065          * wrong data type into a Scanner method
066          *
067          * IOException: An I/O operation failed
068          */
069 import java.util.Scanner; // Library that allows us to capture user
    input
070 import java.util.*; // Allows me to check for
    InputMismatchException
071 import java.io.*; // Allows for system input and output through
    data streams, serialization and the file system
072
```

```
073 public class LessonSix {
074
075     // Creates a Scanner object that monitors keyboard input
076     static Scanner userInput = new Scanner(System.in);
077
078     public static void main(String[] args){
079
080         divideByZero(2);
081
082         System.out.print("How old are you? ");
083         int age = checkValidAge();
084
085         if (age != 0){
086         System.out.println("You are " + age + " years old");
087         }
088
089         getAFile("./somestuff.txt");
090
091     }
092
093     public static void divideByZero(int a)
094     {
095
096         try
097         {
098             // The following throws an error because you can't
    divide by zero
099             // Exception in thread "main"
    java.lang.ArithmeticException
100             System.out.println(a/0);
101         }
102
103         // If the exception ArithmeticException is thrown the
    following execute
104         catch (ArithmeticException e)
105         {
106             // Your custom error message
107             System.out.println("You can't divide by zero");
108
109             // Java's error message for this exception
110             System.out.println(e.getMessage());
111
112             // Prints the exception name and error message
113             System.out.println(e.toString());
114
115             // Prints the standard error stack trace
116             e.printStackTrace();
117         }
118
119     }
120
121     public static int checkValidAge()
122     {
123
124         try
125         {
126             return userInput.nextInt(); // nextInt() receives the
```

```
    user input
127         }
128
129         catch (InputMismatchException e)
130         {
131             userInput.next(); // Skips the last user input and
    waits for the next
132             System.out.print("That isn't a whole number");
133             return 0;
134         }
135
136     }
137
138     /* If you prefer to throw an exception to the calling method
    you use throw
139      * public static void getAFile(String fileName) throws
    IOException, FileNotFoundException
140      * {
141      *      FileInputStream file = new FileInputStream(fileName);
142      * }
143      *
144      * If main called this method, main would have to handle the
    exception
145      *
146      * public static void main(String[] args) {
147      *      try {
148      *          getAFile("./somestuff.txt");
149      *      }
150      *      catch(IOException e) {
151      *          System.out.println("An unknown IO Error Occured");
152      *      }
153      */
154
155     public static void getAFile(String fileName)
156     {
157         try
158         {
159         /* If I tried to do this without providing for an exception
160         * I'd get the error Unhandled Exception Type
    FileNotFoundException
161         * A checked exception is an exception the compiler forces
    you to protect against
162         */
163         FileInputStream file = new FileInputStream(fileName);
164         }
165
166         catch (FileNotFoundException e)
167         {
168             System.out.println("Sorry I couldn't find that file");
169         }
170
171         // You can catch numerous exceptions (List most specific
    first
172         catch (IOException e) // Catches any IO Exception
173         {
174             System.out.println("An unknown IO Error Occured");
175         }
```

```
176
177         /* To ignore an exception do this
178          * catch (ClassNotFoundException e)
179          * {}
180          */
181
182         /* Java 7 allows you to catch multiple exceptions at once
183          * catch (FileNotFoundException | IOException e)
184          * {}
185          */
186
187         // This will catch any exception (This should always be
    last)
188         catch (Exception e)
189         {
190             System.out.println("I catch every exception");
191         }
192
193         // If used finally is always executed whether there was an
    exception or not
194         // It is used for clean up work like closing files and
    database connections
195         finally
196         {
197             System.out.println("");
198         }
199
200     }
201
202 }
```

```
001 import java.io.*;
002
003 import javax.swing.*;
004
005
006 public class Lesson31 extends JFrame{
007
008     static String filePath,parentDirectory;
009
010     static File randomDir, randomFile, randomFile2;
011
012     public static void main(String[] args){
013
014         // Creates a File object in memory
015
016         randomDir = new
    File("/Users/derekbanas/Documents/workspace3/Java Code/Random");
017
018         // Make a directory
019
020         randomDir.mkdir();
021
022         // Make a file named random.txt
023
024         randomFile = new File("random.txt");
025
026         // Make a file and define where to save it in the file
    system
027
028         randomFile2 = new
    File("/Users/derekbanas/Documents/workspace3/Java
    Code/Random/random2.txt");
029
030         // createNewFile and getCanonicalPath have to be called in
031         // a try block to catch IOException
032
033         try {
034
035             // createNewFile creates the file in the file system
036
037             randomFile.createNewFile();
038
039             randomFile2.createNewFile();
040
041             // Returns the path for the file
042
043             filePath = randomFile.getCanonicalPath();
044
045         } catch (IOException e) {
046
047             // You have to catch the IOException
048             e.printStackTrace();
049
050         }
051
052         // Check to see if the file exists in the current directory
```

```
053
054         if (randomFile.exists()){
055
056             System.out.println("File Exists");
057             System.out.println("File is readable: " +
    randomFile.canRead());
058             System.out.println("File is writable: " +
    randomFile.canWrite());
059             System.out.println("File location: " + filePath);
060             System.out.println("File name: " +
    randomFile.getName());
061
062             // Since you created the file without defining a path
    this returns null
063
064             System.out.println("Parent directory: " +
    randomFile.getParent());
065
066             // This returns the parent because it was defined
067
068             parentDirectory = randomFile2.getParent();
069
070             System.out.println("File Two Parent Directory: " +
    parentDirectory);
071
072             System.out.println("Is this a directory: " +
    randomDir.isDirectory());
073
074             // list provides a string array containing all the
    files
075
076             String[] filesInDir = randomDir.list();
077
078             System.out.println("Files in Random Directory\n");
079
080             // Use the enhanced for loop to cycle through the files
081
082             for(String fileName : filesInDir){
083                 System.out.println(fileName + "\n");
084             }
085
086
087             System.out.println("Is this a file: " +
    randomFile.isFile());
088
089             System.out.println("Is this hidden: " +
    randomFile.isHidden());
090
091             // Milliseconds since Jan 1, 1970 when modified
092
093             System.out.println("Last modified: " +
    randomFile.lastModified());
094
095             // Return size of file
096
097             System.out.println("Size of file: " +
    randomFile.length());
```

```
098
099             // Changes the name of the file
100
101             randomFile2.renameTo(new
    File("/Users/derekbanas/Documents/workspace3/Java
    Code/Random/random3.txt"));
102
103             // Output the full path for the file unless the path
    wasn't
104             // defined when the File was created
105
106             System.out.println("New Name: " +
    randomFile2.toString());
107
108             new Lesson31();
109
110         } else {
111
112             System.out.println("File Doesn't Exist");
113
114         }
115
116         // You call delete to delete a file
117
118         if(randomFile.delete()){
119             System.out.println("File Deleted");
120         }
121
122         // I could get an array of File objects from the directory
123
124         File[] filesInDir = randomDir.listFiles();
125
126         for(File fileName : filesInDir){
127             fileName.delete();
128         }
129
130         // You can only delete a directory if it is empty
131
132         if(randomDir.delete()){
133             System.out.println("Directory Deleted");
134         }
135
136
137     }
138
139     public Lesson31(){
140
141         // Creates a file chooser at the location specified
142
143         JFileChooser fileChooser = new JFileChooser(randomDir);
144
145         // Opens the file chooser
146
147         fileChooser.showOpenDialog(this);
148
149
150
```

```
151         }
152
153 }
```

Java Fundamentals Tutorial: Operators

Prev Next

4. Operators

Operators in Java

4.1. Arithmetic Operators

Operator	Use	Description
`+`	`x + y`	Adds `x` and `y`
`-`	`x - y`	Subtracts `y` from `x`
	`-x`	Arithmetically negates `x`
`*`	`x * y`	Multiplies `x` by `y`
`/`	`x / y`	Divides `x` by `y`
`%`	`x % y`	Computes the remainder of dividing `x` by `y`

In Java, you need to be aware of the *type* of the result of a binary (two-argument) arithmetic operator.

- If either operand is of type **`double`**, the other is converted to **`double`**.
- Otherwise, if either operand is of type **`float`**, the other is converted to **`float`**.
- Otherwise, if either operand is of type **`long`**, the other is converted to **`long`**.
- Otherwise, *both* operands are converted to type **`int`**.

For unary (single-argument) arithmetic operators:

- If the operand is of type **`byte`**, **`short`**, or **`char`**, the result is a value of type **`int`**.
- Otherwise, a unary numeric operand remains as is and is not converted.

This can result in unexpected behavior for a newcomer to Java programming. For example, the code below results in a compilation error:

```
short a = 1;
short b = 2;
short c = a + b;     // Compilation error
```

The reason is that the result of the addition is an **`int`** value, which cannot be stored in a variable typed **`short`** unless you explicitly cast it, as in:

```
short c = (short) (a + b);
```

Also, there are a couple of quirks to keep in mind regarding division by 0:

- A non-zero floating-point value divided by 0 results in a signed **`Infinity`**. **`0.0/0.0`** results in **`NaN`**.
- A non-zero integral value divided by integral 0 results in an **`ArithmeticException`** thrown at runtime.

Note

In Java, unlike C/C++, it is legal to compute the remainder of floating-point numbers. The result is the remainder after dividing the dividend by the divisor an integral number of times. For example, **`7.9 % 1.2`** results in **`0.7`** (approximately, given the precision of a **`float`** or **`double`** type).

4.2. Shortcut Arithmetic Operators

Operator	Use	Description
`++`	`x++`	`y = x++;` is the same as `y = x; x = x + 1;`
	`++x`	`y = ++x;` is the same as `x = x + 1; y = x;`

`--`	`x--`	`y = x--;` is the same as `y = x; x = x - 1;`
	`--x`	`y = --x;` is the same as `x = x - 1; y = x;`

The location of the shortcut operator is only important if the operator is used in an expression where the operand value is assigned to another variable or returned from a method.

That is to say that:

```
x++;
```

is equivalent to:

```
++x;
```

But in an assignment:

```
y = x++;
```

is not the same as:

```
y = ++x;
```

Shortcut operators are often used in **`for`** loops to increment the loop index variable (as we will see soon).

4.3. String Concatenation

- The plus (**+**) operator performs string concatenation.
    - The result is a new **`String`** object, for example:

```
String first = "George";
String last = "Washington";
String name = first + " " + last;   // "George Washington"
```

- You can concatenate a **`String`** with Java primitive types.
    - Java automatically generates a **`String`** representation of the primitive value for concatenation, for example:

```
int count = 8;
String msg = "There are " + count + " cows.";   // "There are 8 cows."
```

- The arithmetic **+** operator has the same order of precedence as the **`String`** concatenation **+**.

    - In a complex expression, they are evaluated from left to right, unless you use parentheses to override, for example:

```
String test = 1 + 1 + " equals " + 1 + 1;   // "2 equals 1
1"
```

- For other object types used as string concatenation operands, Java automatically invokes the object's **`toString()`** method, for example:

```
Date now = new Date();
String msg = "The time is: " + now;
// "The time is: Mon Feb 07 18:04:48 PST 2011"
```

4.4. Relational Operators

Operator	Use	Description
>	**x > y**	**x** is greater than **y**
>=	**x >= y**	**x** is greater than or equal to **y**
<	**x < y**	**x** is less than **y**
<=	**x <= y**	**x** is less than or equal to **y**
==	**x == y**	**x** is equal to **y**
!=	**x != y**	**x** is not equal to **y**

Only numerical primitive types (**`byte`**, **`short`**, **`char`**, **`int`**, **`float`**, **`long`**, and **`double`**) can be compared using the ordering operators (**>**, **>=**, **<**, and **<=**). Objects and **`boolean`**s can only be tested for [in]equality (**==**, **!=**).

Due to Java's type safety (**`boolean`**s are not integers and vice versa), it is not possible to make the common mistake:

```
int x = 5;
int y = 3 + 2;

// Does not compile
if (x = y) {
```

```
    // Do something
}

// You must use equality operator
if (x == y) {
    // Do something
}
```

4.5. Logical Boolean Operators

Operator	Use	Evaluates to `true` if
`&&`	`x && y`	Both `x` and `y` are `true`
`||`	`x || y`	Either `x` or `y` are `true`
`!`	`!x`	`x` is not `true`

- All operands to these operators must be `boolean` values.
- The logical `&&` and `\\` operators are subject to *short circuit evaluation*.

These logical operators accept `boolean` values only. Attempting to use numerical or other operand types results in a compilation error.

As with many modern programming languages, Java uses *short circuit evaluation* when evaluating logical boolean operators. This means that conditional operators evaluate the second operand only when needed:

- In `x && y`, `y` is evaluated only if `x` is `true`. If `x` is `false`, the expression immediately evaluates to `false`.
- In `x || y`, `y` is evaluated only if `x` is `false`. If `x` is `true`, the expression immediately evaluates to `true`.

For example:

```
int i = 0;
int j = 5;
if ( (i != 0) && ((j / i) > 1) ) {
    // Do something
}
```

Because we use a logical AND (`&&`), the first expression evaluates to `false` and so the second expression is not tested and the overall condition is `false`.

4.6. Bitwise Operators

Operator	Use	Evaluates to `true` if
`~`	`~x`	Bitwise complement of `x`
	`x &`	

`&`	`y`	AND all bits of `x` and `y`
`|`	`x | y`	OR all bits of `x` and `y`
`^`	`x ^ y`	XOR all bits of `x` and `y`
`>>`	`x >> y`	Shift `x` right by `y` bits, with sign extension
`>>>`	`x >>> y`	Shift `x` right by `y` bits, with `0` fill
`<<`	`x << y`	Shift `x` left by `y` bits

In general, these bitwise operators may be applied to integral values only. For the `~`, `&`, `\|`, and `^` operators:

- If either operand is of type `long`, the other is converted to `long`.
- Otherwise, *both* operands are converted to type `int`.

For the shift operators:

- If the value being shifted is of type `long`, the result is of type `long`.
- Otherwise, the result is of type `int`.

Operation	Decimal Result	Binary Result
`x`	`71`	`00000000000000000000000001000111`
`y`	`5`	`00000000000000000000000000000101`
`~x`	`-72`	`11111111111111111111111110111000`
`x & y`	`5`	`00000000000000000000000000000101`
`x | y`	`71`	`00000000000000000000000001000111`
`x ^ y`	`66`	`00000000000000000000000001000010`
`x >> y`	`2`	`00000000000000000000000000000010`

`x >>> y`	2	00000000000000000000000000000010
`x << y`	2272	00000000000000000000100011100000

The bitwise operators may also be used with **`boolean`** values to produce a **`boolean`** result. However, you cannot combine **`boolean`** and integral operands in a bitwise expression.

4.7. Assignment Operators

Operator	Use	Shortcut for
`=`	`x = y`	`x = y`
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`
`&=`	`x &= y`	`x = x & y` (also works for **`boolean`** values)
`|=`	`x |= y`	`x = x | y` (also works for **`boolean`** values)
`^=`	`x ^= y`	`x = x ^ y` (also works for **`boolean`** values)
`>>=`	`x >>= y`	`x = x >> y`
`>>>=`	`x >>>= y`	`x = x >>> y`
`<<=`	`x <<= y`	`x = x << y`

Actually, a compound assignment expression of the form ***E1 op= E2*** is equivalent to ***E1 = (T) ((E1) op (E1))***, where ***T*** is the type of ***E1***.

For example, the following code is compiles without error:

```
short x = 3;
x += 4.6;
```

and results in **x** having the value **7** because it is equivalent to:

```
short x = 3;
x = (short) (x + 4.6);
```

4.8. Other Operators

Operator	Use	Description
()	**(x + y) * z**	Require operator precedence
?:	**z = b ? x : y**	Equivalent to: **if (b) { z = x; } else { z = y; }**
[]	**array[0]**	Access array element
.	**str.length()**	Access object method or field
(*type*)	**int x = (int) 1.2;**	Cast from one type to another
new	**d = new Date();**	Create a new object
instanceof	**o instanceof String**	Check for object type, returning **boolean**

The short-cut *conditional operator* (**?:**) (also known as the “if-then-else operator” or the “ternary operator”) is used very often to make Java code more compact, so it is worthwhile to get accustomed to it.

For example:

```
int x = 0;
int y = 5;

// Long way

int z1;
if (x == 0) {
    z1 = 0;
} else {
    z1 = x / y;
}
```

```
// Short-cut way
int z2 = (x == 0) ? 0 : x / y;
```

We'll explore the other operators listed on this slide in more depth throughout this class.

4.9. Operator Precedence

1. `[]`, `()`, `.`
2. `++`, `--`, `~`, `!`, `(type)`, `new`, `-`
3. `*`, `/`, `%`
4. `+`, `-`
5. `<<`, `>>`, `>>>`
6. `<`, `<=`, `>`, `>=`, `instanceof`
7. `==`, `!=`
8. `&`
9. `^`
10. `|`
11. `&&`
12. `||`
13. `? :`
14. `=`, `*=`, `/=`, `+=`, `-=`, `%=`, `<<=`, `>>=`, `>>>=`, `&=`, `^=`, `|=`

Use the `()` operator to change precedence of operators:

```
x << (((((x == 0 ? 0 : (x / y)) + 5) * z) > 4) ? 3 : 7)
```

Other than in the most trivial situations, It is always a good practice to use `()` to document your intentions, regardless of precedence rules.

For example, change:

```
int x = 5 - y * 2 > 0 ? y : 2 * y;
```

to:

```
int x = ((5 - (y * 2)) > 0) ? y : (2 * y);
```

even though both lines will assign the same value to `x`

BACK TO TOP

```
/* Java doesn't allow you to inherit from more than one
 * class. So, what do you do when you want do you do
 * when you want to add additional functionality?
 * You create an interface. Interfaces are empty
 * classes. They provide all of the methods you must
 * use, but none of the code.
 */

// This is how you define an interface. They normally
// have a name that is an adjective. Adjectives modify
// nouns. Most objects have noun names
public interface Drivable {

        // You can put fields in an interface, but understand
        // that their values are final and can't be changed
        double PI = 3.14159265;

        // All methods in an interface must be implemented
        // They are public and abstract by default
        // An abstract method must be defined by any class
        // that uses the interface
        int getWheels();

        // You can't define a method as final and abstract
        // final means the method can't be changed and
        // abstract means it must be changed
        void setWheels(int numWheels);

        double getSpeed();

        void setSpeed(double speed);
}


/* If you want to create a class in which every method
 * doesn't have to be implemented use abstract classes. */


// Create an abstract class with the abstract keyword
```

```
public abstract class Crashable{

	boolean carDrivable = true;

	public void youCrashed(){
		this.carDrivable = false;
	}

	public abstract void setCarStrength(int carStrength);

	public abstract int getCarStrength();

}


/* You define that you want a class to use an interface
 * with the implements keyword. This class must create
 * a method for each method defined in Drivable
 * You can implement more than 1 interface like this
 * public class Vehicle implements Drivable, Crashable
 */
// You make a class part of an abstract class by using
//the extends keyword
public class Vehicle extends Crashable implements Drivable {

	int numOfWheels = 2;
	double theSpeed = 0;

	int carStrength = 0;

	// All methods must be as visible as those in the
	// interface. If they are public in the interface
	// they must be public in the subclass
	public int getWheels(){
		return this.numOfWheels;
	}

	public void setWheels(int numWheels){
		this.numOfWheels = numWheels;
```

```
	}

	public double getSpeed(){
		return this.theSpeed;
	}

	public void setSpeed(double speed){
		this.theSpeed = speed;
	}

	public Vehicle(int wheels, double speed){
		this.numOfWheels = wheels;
		this.theSpeed = speed;
	}

	public void setCarStrength(int carStrength){
		this.carStrength = carStrength;
	}

	public int getCarStrength(){
		return this.carStrength;
	}

}
public class LessonFifteen {

	public static void main(String[] args){

		Vehicle car = new Vehicle(4, 100.0);

		// Using methods from the interface Drivable
		System.out.println("Cars Max Speed: "+car.getSpeed());
		System.out.println("Cars Number of Wheels:
"+car.getWheels());

		// Using methods from abstract method Crashable
		car.setCarStrength(10);
		System.out.println("Car Strength: "+car.getCarStrength()); }  }
```

```
import javax.swing.*;
import javax.swing.border.Border;

import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.LayoutManager;
import java.awt.event.*;

// In swing you use a JFrame, to make the program
// work online you use JApplet

public class Lesson40 extends JApplet{

	// The main panel that holds everything

	JPanel thePanel;

	// The panels that hold the radio buttons

	JPanel ques1Panel, ques2Panel, ques3Panel, ques4Panel;

	// When clicked the personality report is displayed

	JButton getResultBut;

	// The radio buttons for each personality quirk

	JRadioButton extravertRadio, introvertRadio,
		sensorRadio, intuitiveRadio, feelerRadio,
		thinkerRadio, judgingRadio, perceivingRadio;

	// Holds the personality report

	JEditorPane yourReport;

	// You use init() instead of main() with Applets

	public void init(){

		// Sets the size of the frame
```

```
        this.setSize(675, 870);

        // Creates the main panel and makes everything justify left

thePanel = new JPanel((LayoutManager) new FlowLayout(FlowLayout.LEFT));

        // The panels for the radio buttons

        ques1Panel = new JPanel();
        ques2Panel = new JPanel();
        ques3Panel = new JPanel();
        ques4Panel = new JPanel();

        // Creates borders that surround the radio buttons

Border border1 = BorderFactory.createTitledBorder("Do you prefer to work");
Border border2 = BorderFactory.createTitledBorder("Which is most important");
        Border border3 = BorderFactory.createTitledBorder("Do you act on");
Border border4 = BorderFactory.createTitledBorder("Which do you prefer");

        // Attaches the border to the right panels

        ques1Panel.setBorder(border1);
        ques2Panel.setBorder(border2);
        ques3Panel.setBorder(border3);
        ques4Panel.setBorder(border4);

        // Makes are only one radio button can be selected

        ButtonGroup group1 = new ButtonGroup();
        ButtonGroup group2 = new ButtonGroup();
        ButtonGroup group3 = new ButtonGroup();
        ButtonGroup group4 = new ButtonGroup();

        // Creates and sets text for radio buttons

        extravertRadio = new JRadioButton("In groups");
        introvertRadio = new JRadioButton("On your own");
        sensorRadio = new JRadioButton("The specifics");
```

```
intuitiveRadio = new JRadioButton("The big picture");
feelerRadio = new JRadioButton("What feels right");
thinkerRadio = new JRadioButton("List of facts");
judgingRadio = new JRadioButton("To plan");
perceivingRadio = new JRadioButton("To adapt");

// Sets some radio buttons to true by default

extravertRadio.setSelected(true);
sensorRadio.setSelected(true);
feelerRadio.setSelected(true);
judgingRadio.setSelected(true);

// Adds radio buttons to their panels

ques1Panel.add(extravertRadio);
ques1Panel.add(introvertRadio);
ques2Panel.add(sensorRadio);
ques2Panel.add(intuitiveRadio);
ques3Panel.add(feelerRadio);
ques3Panel.add(thinkerRadio);
ques4Panel.add(judgingRadio);
ques4Panel.add(perceivingRadio);

// Assigns radio buttons to be in groups together

group1.add(extravertRadio);
group1.add(introvertRadio);
group2.add(sensorRadio);
group2.add(intuitiveRadio);
group3.add(feelerRadio);
group3.add(thinkerRadio);
group4.add(judgingRadio);
group4.add(perceivingRadio);

// Adds the radio button panels to the main panel

thePanel.add(ques1Panel);
thePanel.add(ques2Panel);
thePanel.add(ques3Panel);
```

```
            thePanel.add(ques4Panel);

            // Creates a button

            getResultBut = new JButton("Get Result");

            // Creates an object that will monitor button clicks

            GetResultsListener butListener = new GetResultsListener();

            // Assigns an object that will monitor this buttons clicks
            // When clicked a method will execute

            getResultBut.addActionListener(butListener);

            // Add the button & panel to the frame and show the frame

            thePanel.add(getResultBut);

            this.add(thePanel);

            this.setVisible(true);



        }

        class GetResultsListener implements ActionListener{

            // Called when the button is clicked

            public void actionPerformed(ActionEvent e) {

                    // Define strings that will make the html that the
                    // JEditorPane will display

String pageToOpen = "",
directoryLoc = "file:///Users/derekbanas/Documents/workspace3/Java
Code/src/";
```

```
String textToDisplay ="<html><div><img src=\"" + directoryLoc;

                        // Check if the result button was clicked

                        if (e.getSource() == getResultBut){

                                // Add to the string based on selected radio button

                                if (extravertRadio.isSelected()) pageToOpen += "E";
                                if (introvertRadio.isSelected()) pageToOpen += "I";

                                if (sensorRadio.isSelected()) pageToOpen += "S";
                                if (intuitiveRadio.isSelected()) pageToOpen += "N";

                                if (feelerRadio.isSelected()) pageToOpen += "F";
                                if (thinkerRadio.isSelected()) pageToOpen += "T";

                                if (judgingRadio.isSelected()) pageToOpen += "J";
                                if (perceivingRadio.isSelected()) pageToOpen += "P";

                                // Remove panels to make way for a report

                                thePanel.remove(ques1Panel);
                                thePanel.remove(ques2Panel);
                                thePanel.remove(ques3Panel);
                                thePanel.remove(ques4Panel);

// Finish the html file that JEditorPane will display

                        textToDisplay += pageToOpen + ".png" + "\" /></html>";

// Define what JEditorPane will display

                        yourReport = new JEditorPane("text/html", textToDisplay);

                                // Shut off editing and size the JEditorPane

                                yourReport.setEditable(false);

                                yourReport.setSize(650, 825);
```

```
                    // Add the JEditorPane to a scroller and define how
                    // to handle scrollbars

                    JScrollPane scroller = new JScrollPane(yourReport,

        JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                            JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);

                    // Size the scroll pane

                    scroller.setPreferredSize(new Dimension(650, 825));

                    // Add Scroll pane and JEditorPane to the frame

                    thePanel.add(scroller);

                    getResultBut.setVisible(false);

                    // Redraw the frame after the changes are made

                    thePanel.revalidate();
                    thePanel.repaint();


                }

            }

        }

}



<html>
<head>
<title>Personality Test</title>
</head>
```

```
<body>
<APPLET code="JavaLesson40.class" width="675" height="870">
Sorry you can't run Java
</APPLET>
</body>
</html>
```

```java
// The API for accessing and processing data stored in a database

import java.sql.*;

public class Lesson34 {
    public static void main(String[] args) {

        // A connection object is used to provide access to a database

        Connection conn = null;

        try {
            // The driver allows you to query the database with Java
            // forName dynamically loads the class for you

            Class.forName("com.mysql.jdbc.Driver");

            // DriverManager is used to handle a set of JDBC drivers
            // getConnection establishes a connection to the database
            // You must also pass the userid and password for the database

            conn =
DriverManager.getConnection("jdbc:mysql://localhost/customer","mysqladm","turtledove");

            // Statement objects executes a SQL query
            // createStatement returns a Statement object

            Statement sqlState = conn.createStatement();

            // This is the query I'm sending to the database

            String selectStuff = "Select first_name from customer";

            // A ResultSet contains a table of data representing the
            // results of the query. It can not be changed and can
            // only be read in one direction

            ResultSet rows = sqlState.executeQuery(selectStuff);

            // next is used to iterate through the results of a query

            while(rows.next()){
                System.out.println(rows.getString("first_name"));
            }
        }

        catch (SQLException ex) {

            // String describing the error


            System.out.println("SQLException: " + ex.getMessage());

            // Vendor specific error code

            System.out.println("VendorError: " + ex.getErrorCode());
        }

        catch (ClassNotFoundException e) {
            // Executes if the driver can't be found
            e.printStackTrace();
        }

    }
}
```

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;

// Thrown when a URL doesn't contain http://
// and other rules like that

import java.net.MalformedURLException;
import java.net.URL;

// A text component that allows for rich text
// and basic html display

import javax.swing.JEditorPane;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;

// Provides information on events triggered
// through interaction with links

import javax.swing.event.HyperlinkEvent;

// Monitors user activity with links

import javax.swing.event.HyperlinkListener;

public class Lesson39 extends JFrame implements HyperlinkListener,
ActionListener{

	public static void main(String[] args){

		new
Lesson39("file:///Volumes/My%20Book/Presentations/HTML%20Tutorial/htmlexa
mple.html");

	}
```

```
String defaultURL;

JPanel toolPanel = new JPanel();
JTextField theURL = new JTextField(25);

// Displays basic html pages
// Doesn't understand JavaScript

JEditorPane htmlPage;

public Lesson39(String defaultURL){

        JFrame frame = new JFrame("Java Browser");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   this.defaultURL = defaultURL;

   // If the user interacts with the JTextField the
   // actionPerformed method is called

        theURL.addActionListener(this);

        // Set default text in the JTextField

        theURL.setText(defaultURL);

        // Add the text field to a panel

        toolPanel.add(theURL);

        // Add the panel to the northern quadrant of a frame

        frame.add(toolPanel, BorderLayout.NORTH);

        try {
                htmlPage = new JEditorPane(defaultURL);

                // If the user interacts with the JEditorPane
                // actions are triggered. Ex. Click on a link
```

```
                // change the JEditorPane page location

                htmlPage.addHyperlinkListener(this);

                // You can leave this true for rich text documents
                // but it will mess up web page display

                htmlPage.setEditable(false);

                // Add the JEditorPane to a Scroll pane

                JScrollPane scroller = new JScrollPane(htmlPage);

                // Add Scroll pane and JEditorPane to the frame

                frame.add(scroller, BorderLayout.CENTER);

        }

        // If something goes wrong with locating the html page
        // this will handle that error

        catch (IOException e) {
                e.printStackTrace();
        }

        // Set size of the frame and display it

        frame.setSize(1200, 800);
   frame.setVisible(true);

}

public void hyperlinkUpdate(HyperlinkEvent e) {

        // Checks if a link was clicked
        // EventType.ENTERED : Checks for hovering
        // EventType.EXITED : Checks for leaving link

        if(e.getEventType() == HyperlinkEvent.EventType.ACTIVATED){
```

```
                try {

                        // Sets the URL to be displayed
                        // getURL gets the URL for the link

                        htmlPage.setPage(e.getURL());

                        // toExternalForm creates a String representation of
the URL

                        theURL.setText(e.getURL().toExternalForm());

                }

                catch(IOException e1){
                e1.printStackTrace();
        }

        }

    }

    public void actionPerformed(ActionEvent e) {

        String pageURL = "";

        // Gets the Object that had an event triggered

        if(e.getSource() == theURL){

                // Get the text in the JTextField

                pageURL = theURL.getText();

        } else {

                pageURL = defaultURL;

                // Opens an alert box when an error is made
```

```
                JOptionPane.showMessageDialog(Lesson39.this,
                        "Please Enter a Web Address", "Error",
                        JOptionPane.ERROR_MESSAGE);

            }

            try{

                // Sets the URL to be displayed

                htmlPage.setPage(new URL(pageURL));
                theURL.setText(pageURL);

            }

            catch(MalformedURLException e2){
                JOptionPane.showMessageDialog(Lesson39.this,
                        "Please use http://", "Error",
                        JOptionPane.ERROR_MESSAGE);
            }

            catch(IOException e1){
                e1.printStackTrace();
            }

        }

}
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta name="generator" content="HTML Tidy for Linux (vers 6 November 2007),
see www.w3.org" />
<meta http-equiv="Content-Type" content="text/html;charset=ISO-8859-1" />
```

```
<style type="text/css">
<![CDATA[

h1 {color:red}

]]>
</style>
<meta name="description" content="Welcome" />
<meta name="keywords" content="Flowers," />
<meta name="author" content="Bob" />
<title>Welcome to Bob's Hardware Store, we sell Flowers, Seeds, Rakes</title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<style type="text/css">
/*<![CDATA[*/
 span.c2 {text-decoration: line-through}
 div.c1 {text-align: center}
/*]]>*/
</style>
</head>
<body>
<h1>Text to Draw Attention to</h1>
<abbr title="A">Something Abbreviated</abbr> <acronym
title="as">ASAP</acronym> <strong>I Want this to be Bold</strong> <big>This is
going to be Big</big>
<blockquote>Indent me and show I'm important</blockquote>
<div class="c1">Center Me Please</div>
<em>I like Being Italicized</em> <em>I like being Italicized too</em>
<p>This is a big paragraph. Really, it is!</p>
<q>You can quote me on that</q> <span class="c2">I've been struck</span>
<small>I feel so small</small> <strong>I'm the strongest</strong> <sub>I'm
Subscript</sub> <sup>I'm Superscript</sup> <br /><img
src="file:///Volumes/My%20Book/Presentations/HTML%20Tutorial/Frame4.jpg"
alt="Photo" height="150" width="200" />
<dl>
<dt>Bicycle</dt>
<dd>- Has two Wheels</dd>
<dt>Car</dt>
<dd>- Has Four Wheels</dd>
</dl>
<ol>
```

```
<li>Car</li>
<li>Truck</li>
<li>Bicycle</li>
</ol>
<ul>
<li>Car</li>
<li>Truck</li>
<li>Bicycle</li>
</ul>
<a href="http://www.newthinktank.com" target="_blank">The New Think Tank Website</a> <a href="http://www.newthinktank.com#Great">Some Great Stuff</a> <a href="derekbanas@example.com?Subject=Hello%20again">Send Mail</a>
<table summary="Here">
<caption>Here are some things I did</caption>
<tr>
<th>Sat</th>
<th>Sun</th>
<th>Mon</th>
</tr>
<tr>
<td>Played Ball</td>
<td>Watched Football</td>
<td>Worked</td>
</tr>
</table>
<form action="mail2.php" method="post"><strong>Send Message to this Email</strong><br />
<input type="text" name="email" size="40" /><br />
<p><strong>Subject</strong><br />
<input type="text" name="subject" size="40" /><br /></p>
<p><strong>Text Area</strong><br />
<textarea cols="40" rows="10" name="message">
Default Text in Text Area Box
</textarea>
<br /></p>
<p><strong>Text Input</strong><br />
<input type="text" name="textinput" /><br /></p>
<p><strong>Password Input</strong><br />
<input type="password" name="password" /><br /></p>
```

```
<p><strong>Radio Input</strong><br />
<input type="radio" name="radioinput" value="Love Radio Buttons" />Love Radio Buttons <input type="radio" name="radioinput" value="Hate Radio Buttons" />Hate Radio Buttons<br /></p>
<p><strong>Checkbox Input</strong><br />
<input type="checkbox" name="checkboxinput" value="Love Checkboxs" />Love Checkboxs <input type="checkbox" name="checkboxinput" value="Hate Checkboxs" />Hate Checkboxs<br /></p>
<p><strong>Select Input</strong><br />
<select name="selectinput">
<option value="loveselect">I love select buttons</option>
<option value="hateselect">I hate select buttons</option>
</select><br /></p>
<p><strong>Option Input</strong><br />
<select name="optioninput">
<option value="loveselect">I love option buttons</option>
<option value="hateselect">I hate option buttons</option>
</select><br /></p>
<p><input type="submit" value="Send" /> <input type="hidden" name="submitted" value="TRUE" /></p>
</form>
</body>
</html>
```

```
001 import java.awt.BorderLayout;
002 import java.awt.Font;
003 import java.sql.*;
004
005 import javax.swing.JFrame;
006 import javax.swing.JLabel;
007 import javax.swing.JScrollPane;
008 import javax.swing.JTable;
009
010 import javax.swing.table.DefaultTableCellRenderer;
011 import javax.swing.table.DefaultTableModel;
012 import javax.swing.table.TableColumn;
013
014
015 public class Lesson36{
016
017     // Used to hold the column data for each player
018
019     static Object[][] databaseInfo;
020
021     // The column titles for the JTable
022
023     static Object[] columns = {"Year", "PlayerID", "Name", "TTRC", "Team", "Salary",
    "CPR", "POS"};
024
025     // A ResultSet contains a table of data filled
026     // with the results of the query.
027
028     static ResultSet rows;
029
030     // ResultSetMetaData contains information on
031     // the data returned by the query
032
033     static ResultSetMetaData metaData;
034
035     // DefaultTableModel defines the methods JTable will use
036     // I'm overriding the getColumnClass
037
038     static DefaultTableModel dTableModel = new DefaultTableModel(databaseInfo,
    columns){
039         public Class getColumnClass(int column) {
040             Class returnValue;
041
042             // Verifying that the column exists (index > 0 && index < number of
    columns
043
044             if ((column >= 0) && (column < getColumnCount())) {
045               returnValue = getValueAt(0, column).getClass();
046             } else {
047
048               // Returns the class for the item in the column
049
050               returnValue = Object.class;
051             }
052             return returnValue;
053           }
054         };

055
056     public static void main(String[] args){
057
058         JFrame frame = new JFrame();
059         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
060
061         Connection conn = null;
062
063         try {
064             // The driver allows you to query the database with Java
065             // forName dynamically loads the class for you
066
067             Class.forName("com.mysql.jdbc.Driver");
```

```
068
069             // DriverManager is used to handle a set of JDBC drivers
070             // getConnection establishes a connection to the database
071             // You must also pass the userid and password for the database
072
073             conn =
    DriverManager.getConnection("jdbc:mysql://localhost/lahman591","mysqladm","turtledove");
074
075             // Statement objects executes a SQL query
076             // createStatement returns a Statement object
077
078             Statement sqlState = conn.createStatement();
079
080             // This is the query I'm sending to the database
081
082             String selectStuff = "select b.yearID, b.playerID, " +
083                     "CONCAT(m.nameFirst, ' ', m.nameLast) AS Name, " +
084                     "((b.H+b.BB)+(2.4*(b.AB+b.BB)))*(t.TB+(3*(b.AB+b.BB)))/(9*
    (b.AB+b.BB))-(.9*(b.AB+b.BB)) AS TTRC, " +
085                     "b.teamID AS Team, s.salary AS Salary, " +
086                     "CAST( s.salary/(((b.H+b.BB)+(2.4*(b.AB+b.BB)))*(t.TB+(3*
    (b.AB+b.BB)))/(9*(b.AB+b.BB))-(.9*(b.AB+b.BB))) as decimal(10,2)) AS CPR, " +
087                     "f.POS AS POS FROM Batting b, Master m, Salaries s, TOTBYR t,
    Fielding f " +
088                     "WHERE b.playerID = m.playerID AND t.playerID = m.playerID " +
089                     "AND t.yearID = 2010 AND b.yearID = t.yearID AND s.playerID =
    b.playerID " +
090                     "AND s.yearID = b.yearID AND b.AB > 50 AND b.playerID = f.playerID
    " +
091                     "AND b.playerID = t.playerID GROUP BY b.playerID ORDER BY TTRC
    DESC LIMIT 200;";
092
093             /* Have to cut out salary because it isn't in database for 2011
094             String selectStuff = "select b.yearID, b.playerID, " +
095                     "CONCAT(m.nameFirst, ' ', m.nameLast) AS NAME, " +
096                     "((b.H+b.BB)+(2.4*(b.AB+b.BB)))*(t.TB+(3*(b.AB+b.BB)))/(9*
    (b.AB+b.BB))-(.9*(b.AB+b.BB)) AS TTRC, " +
097                     "b.teamID AS Team, 0 AS Salary, " +
098                     "0 AS CPR, f.POS AS POS " +
099                     "FROM Batting b, Master m, Fielding f, TOTBYR as t " +
100                     "WHERE b.playerID = m.playerID AND t.playerID = m.playerID " +
101                     "AND t.yearID = 2011 AND b.yearID = t.yearID " +
102                     "AND b.AB > 100 AND f.playerID = b.playerID " +
103                     "GROUP BY b.playerID ORDER BY TTRC DESC LIMIT 200;";
104             */
105
106             // A ResultSet contains a table of data representing the
107             // results of the query. It can not be changed and can
108             // only be read in one direction
109
110             rows = sqlState.executeQuery(selectStuff);
111
112             /*
113             int numOfCol;
114
115             // Retrieves the number, types and properties of the Query Results
116
117             metaData = rows.getMetaData();
118
119             // Returns the number of columns
120
121             numOfCol = metaData.getColumnCount();
122
123             // One way to get the column titles
124
125             columns = new String[numOfCol];
126
127             for(int i=1; i<=numOfCol; i++)
128             {
129
130                 // Returns the column name
```

```
131
132                 columns[i] = metaData.getColumnName(i);
133                 System.out.println(i);
134             }
135             */
136
137             // Temporarily holds the row results
138
139             Object[] tempRow;
140
141          // next is used to iterate through the results of a query
142
143             while(rows.next()){
144
145                 // Gets the column values based on class type expected
146
147                 tempRow = new Object[]{rows.getInt(1), rows.getString(2),
    rows.getString(3),
148                         rows.getDouble(4), rows.getString(5), rows.getInt(6),
    rows.getDouble(7),
149                         rows.getString(8)};
150
151                 // Adds the row of data to the end of the model
152
153                 dTableModel.addRow(tempRow);
154
155             }
156         }
157
158         catch (SQLException ex) {
159
160             // String describing the error
161
162             System.out.println("SQLException: " + ex.getMessage());
163
164             // Vendor specific error code
165
166             System.out.println("VendorError: " + ex.getErrorCode());
167         }
168
169         catch (ClassNotFoundException e) {
170             // Executes if the driver can't be found
171             e.printStackTrace();
172         }
173
174         // Create a JTable using the custom DefaultTableModel
175
176         JTable table = new JTable(dTableModel);
177
178         // Increase the font size for the cells in the table
179
180         table.setFont(new Font("Serif", Font.PLAIN, 20));
181
182         // Increase the size of the cells to allow for bigger fonts
183
184         table.setRowHeight(table.getRowHeight()+10);
185
186         // Allows the user to sort the data
187
188         table.setAutoCreateRowSorter(true);
189
190         /* If you want to right justify column
191          *
192         TableColumn tc = table.getColumn("TTRC");
193         RightTableCellRenderer rightRenderer = new RightTableCellRenderer();
194         tc.setCellRenderer(rightRenderer);
195         */
196
197         // Disable auto resizing
198         table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
199
```

```
200         // Set the width for the columns
201
202         TableColumn col1 = table.getColumnModel().getColumn(0);
203         col1.setPreferredWidth(100);
204
205         TableColumn col2 = table.getColumnModel().getColumn(1);
206         col2.setPreferredWidth(190);
207
208         TableColumn col3 = table.getColumnModel().getColumn(2);
209         col3.setPreferredWidth(260);
210
211         TableColumn col5 = table.getColumnModel().getColumn(5);
212         col5.setPreferredWidth(200);
213
214         TableColumn col6 = table.getColumnModel().getColumn(6);
215         col6.setPreferredWidth(200);
216
217         // Change justification of column to Center
218
219         TableColumn tc = table.getColumn("Team");
220         CenterTableCellRenderer centerRenderer = new CenterTableCellRenderer();
221         tc.setCellRenderer(centerRenderer);
222
223         tc = table.getColumn("POS");
224         centerRenderer = new CenterTableCellRenderer();
225         tc.setCellRenderer(centerRenderer);
226
227
228         JScrollPane scrollPane = new JScrollPane(table);
229         frame.add(scrollPane, BorderLayout.CENTER);
230         frame.setSize(800, 500);
231         frame.setVisible(true);
232
233     }
234
235 }
236
237 // How to change justification to the right
238
239 class RightTableCellRenderer extends DefaultTableCellRenderer {
240       public RightTableCellRenderer() {
241         setHorizontalAlignment(JLabel.RIGHT);
242       }
243
244 }
245
246 // Change justification to the center
247
248 class CenterTableCellRenderer extends DefaultTableCellRenderer {
249       public CenterTableCellRenderer() {
250         setHorizontalAlignment(JLabel.CENTER);
251       }
252
253 }
```

```
001 // The LinkedList class is a collection based on a Linked List
002 // instead of an array. They are particularly good when you
003 // expect to perform many additions and deletions with a
004 // collection. When using a linked list you don't have to
005 // move items around when you add or delete an item. They
006 // aren't particularly suited to providing access based off
007 // of index searches like an array though. Each object in a
008 // linked list contains a pointer to the objects that proceed
009 // and follow it.
010 // When you change an ArrayList a new array is created by it.
011
012 import java.util.Arrays;
013 import java.util.LinkedList; // LinkedList Library methods
014
015 public class LessonTwelve {
016
017     public static void main(String[] args){
018
019         // Creates a LinkedList object
020         LinkedList linkedListOne = new LinkedList();
021
022         // Creates a LinkedList that contains Strings
023         LinkedList<String> names = new LinkedList<String>();
024
025         // You use add to add items to the Linked List
026         names.add("Ahmed Bennani");
027         names.add("Ali Syed");
028
029         // addLast places the object last in the list
030         names.addLast("Nathan Martin");
031
032         // addLast places the object first in the list
033         names.addFirst("Joshua Smith");
034
035         // You can define what position to place the object in
036         names.add(0, "Noah Peeters");
037
038         // You replace a value in an index with set()
039         names.set(2, "Paul Newman");
040
041         // You remove items either by providing the index, or
042         // the value
043         names.remove(4);
044         names.remove("Joshua Smith");
045
046         // removeFirst() removes the first element
047         // removeLast() removes the last element
048         // removeFirstOccurrence(Object) removes the
049         // first Object that matches what you passed
050
051         // You can use the enhanced for to print them out
052         for(String name : names)
053         {
054             System.out.println(name);
055         }
056
```

```
057         /* OUTPUT
058          * Noah Peeters
059          * Paul Newman
060          * Ali Syed
061          */
062
063         // You can retrieve values with get()
064         System.out.println("\nFirst Index: " + names.get(0));
065
066         /* OUTPUT
067          * First Index: Noah Peeters
068          */
069
070         // Retrieve the first value with getFirst()
071         System.out.println("\nFirst Index: " + names.getFirst());
072
073         /* OUTPUT
074          * First Index: Noah Peeters
075          */
076
077         // Retrieve the first value with getFirst()
078         System.out.println("\nLast Index: " + names.getLast());
079
080         /* OUTPUT
081          * Last Index: Ali Syed
082          */
083
084         LinkedList<String> nameCopy = new LinkedList<String>
    (names);
085
086         // When you print out the LinkedList itself the toString
087         // method is called
088         System.out.println("\nnameCopy: " + nameCopy);
089
090         /* OUTPUT
091          * nameCopy: [Noah Peeters, Paul Newman, Ali Syed]
092          */
093
094         // You can check if an object is in the list with
    contains()
095         if(names.contains("Noah Peeters"))
096         {
097             System.out.println("\nNoahs Here");
098         }
099
100         /* OUTPUT
101          * Noahs Here
102          */
103
104         // You can check if everything in one list is in another
105         if(names.containsAll(nameCopy))
106         {
107             System.out.println("\nCollections are the same");
108         }
109
110         /* OUTPUT
111          * Collections are the same
```

```
112          */
113
114         // Return the index for an object with indexOf
115         System.out.println("\nIndex of Ali is: " +
    names.indexOf("Ali Syed"));
116
117         /* OUTPUT
118          * Index of Ali is: 2
119          */
120
121         // Check if a list is empty with isEmpty()
122         System.out.println("List Empty: " + names.isEmpty());
123
124         /* OUTPUT
125          * List Empty: false
126          */
127
128         // Get the number of items in the list with size
129         System.out.println("How many values: " + names.size());
130
131         /* OUTPUT
132          * How many values: 3
133          */
134
135         // peek() retrieves the first element, but doesn't throw an
    error
136         // if there is no element to retrieve
137         System.out.println("Look without error: " + names.peek());
138
139         /* OUTPUT
140          * Look without error: Noah Peeters
141          */
142
143         // poll() returns the first value and deletes it from the
    list
144         System.out.println("Remove first element: " +
    nameCopy.poll());
145
146         /* OUTPUT
147          * Remove first element: Noah Peeters
148          */
149
150         // poll() returns the last value and deletes it from the
    list
151         System.out.println("Remove last element: " +
    nameCopy.pollLast());
152
153         /* OUTPUT
154          * Remove last element: Ali Syed
155          */
156
157         // push puts a new element on the front of the list
158         nameCopy.push("Noah Peeters");
159
160         // pop removes an element on the front of the list
161         nameCopy.pop();
162
```

```
163         System.out.println("\nnameCopy: " + nameCopy);
164
165         /* OUTPUT
166          * nameCopy: [Paul Newman]
167          */
168
169         // Create a new array to hold values from the Linked List
170         Object[] nameArray = new Object[4];
171
172         // toArray converts the LinkedList into an array of objects
173         nameArray = names.toArray();
174
175         // toString converts items in the array into a String
176         System.out.println(Arrays.toString(nameArray));
177
178         /* OUTPUT
179          * [Noah Peeters, Paul Newman, Ali Syed]
180          */
181
182         // clear() deletes everything in the Linked List
183         names.clear();
184
185
186     }
187
188 }
```

```
001 import java.util.Arrays;
002
003 // Basic class definition
004 // public means this class can be used by other classes
005 // Class names should begin with a capital letter
006 // A file can't contain two public classes. It can contain classes
    that are not public
007 // If you place class files in the same folder the java compiler
    will be able to find them
008
009 public class MonsterTwo{
010
011     static char[][] battleBoard = new char[10][10];
012
013     public static void buildBattleBoard(){
014
015         for(char[] row : battleBoard)
016         {
017
018             Arrays.fill(row, '*');
019
020         }
021
022     }
023
024     public static void redrawBoard()
025     {
026
027         int k = 1;
028         while(k <= 30){ System.out.print('-'); k++; }
029         System.out.println();
030
031         for (int i = 0; i < battleBoard.length; i++)
032         {
033
034             for(int j = 0; j < battleBoard[i].length; j++)
035             {
036
037                 System.out.print("|" + battleBoard[i][j] + "|");
038
039             }
040             System.out.println();
041
042         }
043         k = 1;
044         while(k <= 30){ System.out.print('-'); k++; }
045         System.out.println();
046
047
048     }
049
050     // Class Variables or Fields
051     // You declare constants with final
052
053     public final String TOMBSTONE = "Here Lies a Dead monster";
054
```

```
055     // private fields are not visible outside of the class
056     private int health = 500;
057     private int attack = 20;
058     private int movement = 2;
059
060     private boolean alive = true;
061
062     // public variables are visible outside of the class
063     // You should have as few as possible public fields
064     public String name = "Big Monster";
065     public char nameChar1 = 'B';
066     public int xPosition = 0;
067     public int yPosition = 0;
068     public static int numOfMonsters = 0;
069
070     // Class Methods
071     // Accessor Methods are used to get and set the values of
    private fields
072
073     public int getAttack()
074     {
075         return attack;
076     }
077
078     public int getMovement()
079     {
080         return movement;
081     }
082
083     public int getHealth()
084     {
085         return health;
086     }
087
088     public boolean getAlive()
089     {
090         return alive;
091     }
092
093     // You can create multiple versions using the same method name
094     // Now setHealth can except an attack that contains decimals
095     // When overloading a method you can't just change the return
    type
096     // Focus on creating methods that except different parameters
097
098     public void setHealth(int decreaseHealth)
099     {
100         health = health - decreaseHealth;
101         if (health < 0)
102         {
103             alive = false;
104         }
105     }
106
107     public void setHealth(double decreaseHealth)
108     {
109         int intDecreaseHealth = (int) decreaseHealth;
```

```
110         health = health - intDecreaseHealth;
111         if (health < 0)
112         {
113             alive = false;
114         }
115     }
116 
117     /* The Constructor
118      * Code that is executed when an object is created from this
    class definition
119      * The method name is the same as the class
120      * The constructor is only executed once per object
121      * The constructor can't return a value
122      */
123 
124     public MonsterTwo(int health, int attack, int movement, String
    name)
125     {
126         this.health = health;
127         this.attack = attack;
128         this.movement = movement;
129         this.name = name;
130         /* If the attributes had the same names as the class
    health, attack, movement
131          * You could refer to the objects fields by proceeding them
    with this
132          * this.health = health;
133          * this.attack = attack;
134          * objectFieldName = attributeFieldName;
135          */
136 
137         int maxXBoardSpace = battleBoard.length - 1;
138         int maxYBoardSpace = battleBoard[0].length - 1;
139 
140         int randNumX, randNumY;
141 
142         do {
143 
144             randNumX = (int) (Math.random() * maxXBoardSpace);
145             randNumY = (int) (Math.random() * maxYBoardSpace);
146 
147         } while(battleBoard[randNumX][randNumY] != '*');
148 
149         this.xPosition = randNumX;
150         this.yPosition = randNumY;
151 
152         this.nameChar1 = name.charAt(0);
153 
154         battleBoard[this.yPosition][this.xPosition] =
    this.nameChar1;
155 
156         numOfMonsters++;
157 
158     }
159 
160     // You can overload constructors like any other method
161     // The following constructor is the one provided by default if
```

```
    you don't create any other constructors
162     // Default Constructor
163
164     public MonsterTwo()
165     {
166         numOfMonsters++;
167     }
168
169 public static void main(String[] args)
170 {
171
172
173 }
174
175 }
```

```
001 import java.util.Scanner; // Library that allows us to capture user
    input
002
003 public class LessonFour {
004
005     // Creates a Scanner object that monitors keyboard input
006     static Scanner userInput = new Scanner(System.in);
007
008     public static void main(String[] args)
009     {
010         /*
011          * The while loop : A while loop executes the code between
    {} until the
012          * condition is no longer true
013          */
014         // while loops create a loop iterator before the loop
    begins
015         int i = 1;
016
017         while(i < 20)
018         {
019             // Use continue to skip over printing 3
020             if(i == 3)
021             {
022                 i+=2; // When using continue, don't forget to
    change the iterator
023                 continue; // continue skips a loop iteration, but
    not the loop
024             }
025
026             System.out.println(i);
027             i++;
028
029             // Just print odd numbers
030             if((i%2) == 0)
031             {
032                 i++;
033             }
034
035             // i is greater than 10 jump out of the loop
036             if(i > 10)
037             {
038                 break; // Forces the program to leave the loop
    prematurely
039             }
040         }
041
042         /*
043          * Code that calculates the value for PI
044          */
045         double myPi = 4.0; // My starting value for pi
046
047         // Starting value for my loop iterator in the loop
048         double j = 3.0;
049
050         // The while loop
```

```
051         while(j<11)
052         {
053             // I calculate pi with this formula
054             // PI = 4 - 4/3 + 4/5 - 4/7...
055             myPi = myPi - (4/j) + (4/(j+2));
056             j += 4;
057             System.out.println(myPi);
058         }
059 
060         System.out.println("Value of PI: " + Math.PI);
061 
062         /*
063          * Execute while loop until the user quits it
064          */
065         String contYorN = "Y";
066         int h = 1;
067 
068         // equalsIgnoreCase checks if the input string is equal to
    y or Y
069         while (contYorN.equalsIgnoreCase("y"))
070         {
071             System.out.println(h);
072             System.out.print("Continue y or n?");
073             contYorN = userInput.nextLine(); // Accepts a string
    input from the user
074             h++;
075 
076         }
077 
078         /*
079          * The do while loop : Used when you want to guarantee the
    code
080          * between {} will execute at least once. The code is
    executed and
081          * then java checks if it should execute again
082          */
083         // loop iterator for the do while loop
084         // It must be created before the expression is evaluated
    below
085         int k = 1;
086 
087         do
088         {
089             System.out.println(k);
090             k++;
091         } while (k <= 10); // Counts from 1 to 10
092 
093         /*
094          * The for loop : Another looping tool in Java
095          * for( declare iterator; conditional statement; change
    iterator value)
096          */
097 
098         for (int l=10; l >= 1; l--)
099         {
100             System.out.println(l);
101         }
```

```
102
103         // System.out.println(l); The variable l is not callable
    outside of for
104
105         int m, n; // You don't have to declare variables in the for
    loop
106         // You can use multiple variables in the for loop
107         for (m=1, n=2; m <= 9; m+=2, n+=2)
108         {
109             System.out.println(m + "\n" + n);
110         }
111
112         // You can have for loops inside of for loops, but I'll
    save
113         // that topic until I talk about arrays
114
115     }
116
117 }
```

```
001 import java.util.Scanner; // Scanner is a class that contains a
    bunch of methods
002
003 public class LessonFive {
004     // main is a method that contains all of the code to be
    executed when a program runs
005
006     // This is a class variable that is available to every method
007     // If you declare a variable in a method it is only accessible
    in that method (local variable)
008
009     static double myPI = 3.14159265;
010
011     // Any changes made to randomNumber in any functions will
    effect its global value
012
013     static int randomNumber;
014
015     // Creates a Scanner object that monitors keyboard input
016
017     static Scanner userInput = new Scanner(System.in);
018
019     public static void main(String[] args)
020     {
021
022         /* Basic Method
023          * accessModifier static returnDataType methodName
    (parameters)
024          * { Statements }
025          * Access Modifier: Determines who can execute a method
026          * static: Used when you want to be able to execute a
    method that isn't part of a class definition
027          * Return Data Type: The data type of value returned after
    a method executes (void if no values are returned)
028          * Method Name: Must start with a letter, but can include
    letters, numbers, $, or _
029          * Parameters / Arguments: Values passed to a method
030          */
031
032         System.out.println(addThem(1,2)); // addThem(1,2) will be
    replaced with the value that method returns
033
034         // Demonstrating passing by value
035         int d = 5;
036
037         // Changes to the variable d in tryToChange don't effect
    its value globally
038         // We are passing the value of d to tryToChange and not the
    variable
039
040         tryToChange(d);
041         System.out.println("Static Variable d = "+ d);
042
043         // Guessing a random number
044         System.out.println("\n");
045
```

```
046         System.out.println(getRandomNum()); // Both prints and
    generates the value for randomNumber
047
048         int guessResult = 1;
049         int randomGuess = 0;
050
051         while(guessResult != -1)
052         {
053         System.out.print("Guess a number between 0 and 50: ");
054
055         // Accepts an integer input from the user
056         // You can't declare this variable inside the while loop if
    you want to access it outside the while loop
057
058         randomGuess = userInput.nextInt();
059
060         guessResult = checkGuess(randomGuess);
061         }
062
063         System.out.println("Yes the random number is " +
    randomGuess);
064
065         System.out.println(randomNumber); // Random value was
    changed globally by getRandomNum
066
067     }
068
069     // Adds the two numbers sent and returns the solution
070     // public is the access modifier and means anyone can execute
    this method
071     // Java Methods can return any primitive data type, or
    reference to an object (More on that later)
072
073     public static int addThem(int a, int b)
074     {
075         double smallPI = 3.140; // This variable is local to the
    addThem function
076
077         // compare returns 0 if equal | -1 if smallPI is less than
    myPI | 1 if smallPI is greater than myPI
078
079         System.out.println(Double.compare(smallPI, myPI));
080
081         int c = a + b;
082
083         // return returns a value that replaces the call to this
    method
084         // It must be an int since you defined this method returns
    ints above
085
086         return c;
087     }
088
089     // When you define an attribute / parameter you must define its
    type
090     // That's why you can't type tryToChange(d)
091     // Because this function doesn't return a value return type is
```

```
    void
092
093     public static void tryToChange(int d)
094     {
095         d = d + 1;
096         System.out.println("tryToChange d = " + d);
097     }
098
099     public static int getRandomNum()
100     {
101         // Creates a random number between 0 and 50
102         // Since randomNumber is a class variable you don't have to
    declare, or define its type
103         // If int randomNumber was declared in this method it
    wouldn't effect the global variable named randomNumber
104
105         randomNumber = (int) (Math.random() * 51);
106         return randomNumber;
107     }
108
109     public static int checkGuess(int guess)
110     {
111         if(guess == randomNumber)
112         {
113             return -1;
114         } else {
115             return guess; // Must return a value of type int
116         }
117     }
118
119 }
```

```
public class LessonSixteen{

	public static void main(String[] args){

		// Every object inherits all the methods in the Object class

		Object superCar = new Vehicle();

		// superCar inherits all of the Object methods, but an object
		// of class Object can't access the Vehicle methods

		// System.out.println(superCar.getSpeed()); * Throws an error

		// You can cast from type Object to Vehicle to access those methods

		System.out.println(((Vehicle)superCar).getSpeed());

		// The methods of the Object class

		Vehicle superTruck = new Vehicle();

		// equals tells you if two objects are equal
		System.out.println(superCar.equals(superTruck));

		// hashCode returns a unique identifier for an object
		System.out.println(superCar.hashCode());

		// finalize is called by the java garbage collector when an object
		// is no longer of use. If you call it there is no guarantee it will
		// do anything though

		// getClass returns the class of the object

		System.out.println(superCar.getClass());

		// THE CLASS OBJECT
		// You can use the Class object method getName to get just the class
name
```

```
System.out.println(superCar.getClass().getName());

// You can check if 2 objects are of the same class with getClass()

if(superCar.getClass() == superTruck.getClass()){
        System.out.println("They are in the same class");
}

// getSuperclass returns the super class of the class

System.out.println(superCar.getClass().getSuperclass());

// the toString method is often overwritten for an object

System.out.println(superCar.toString());

// toString is often used to convert primitives to strings

int randNum = 100;
System.out.println(Integer.toString(randNum));

// THE CLONE METHOD
// clone copies the current values of the object and assigns
// them to another. If changes are made after the clone both
// objects aren't effected though

superTruck.setWheels(6);

Vehicle superTruck2 = (Vehicle)superTruck.clone();

System.out.println(superTruck.getWheels());

System.out.println(superTruck2.getWheels());

// They are separate objects and don't have equal hashcodes
System.out.println(superTruck.hashCode());
System.out.println(superTruck2.hashCode());
```

```
            // There are subobjects defined in an object clone won't
            // also clone them. You'd have to do that manually, but this
            // topic will be covered in the future because of complexity

        }

}
```

```
public class Vehicle extends Crashable implements Drivable, Cloneable{

        int numOfWheels = 2;
        double theSpeed = 0;

        int carStrength = 0;

        public int getWheels(){
                return this.numOfWheels;
        }

        public void setWheels(int numWheels){
                this.numOfWheels = numWheels;
        }

        public double getSpeed(){
                return this.theSpeed;
        }

        public void setSpeed(double speed){
                this.theSpeed = speed;
        }

        public Vehicle(){

        }
```

```
	public Vehicle(int wheels, double speed){
		this.numOfWheels = wheels;
		this.theSpeed = speed;
	}

	public void setCarStrength(int carStrength){
		this.carStrength = carStrength;
	}

	public int getCarStrength(){
		return this.carStrength;
	}

	public String toString(){
		return "Num of Wheels: " + this.numOfWheels;
	}

	public Object clone(){
		Vehicle car;

		try{
			car = (Vehicle) super.clone();
		}

		catch(CloneNotSupportedException e){
			return null;
		}
		return car;
	}

}
```

```java
// Animal will act as a Super class for other Animals
public class Animal {

    private String name = "Animal";
    public String favFood = "Food";

    // You use protected when you want to allow subclasses
    // To be able to access methods or fields
    // If you would have used private their would be no
    // way for subclasses to call this method
    // This is a final method which means it can't be overwritten

    protected final void changeName(String newName){

        // this is a reference to the object you're creating

        this.name = newName;

    }

    protected final String getName(){

        return this.name;

    }

    public void eatStuff(){

        System.out.println("Yum " + favFood);

    }

    public void walkAround(){

        System.out.println(this.name + " walks around");

    }

    public Animal(){

    }

    public Animal(String name, String favFood){

        this.changeName(name);
        this.favFood = favFood;

    }

}
```

```
001 import java.io.*;
002
003 // A character stream is just a series of characters
004 // Important information is normally separated by a comma,
005 // space, or tab.
006
007 public class Lesson32{
008
009     public static void main(String[] args){
010
011         // Create an array of type Customer
012
013         Customer[] customers = getCustomers();
014
015         // PrintWriter is used to write characters to a file in
    this situation
016
017         PrintWriter custOutput =
    createFile("/Users/derekbanas/Documents/workspace3/Java
    Code/src/customers.txt");
018
019         // Enhanced for loop for arrays
020         // Cycles through all of the people in the customers array
021
022         for(Customer person : customers){
023
024             createCustomers(person, custOutput);
025
026         }
027
028         // Closes the connection to the PrintWriter
029
030         custOutput.close();
031
032         getFileInfo();
033
034     }
035
036     // class that defines all the fields for my customers
037
038     private static class Customer{
039
040         public String firstName, lastName;
041         public int custAge;
042
043         // constructor that's called when a customer is made
044
045         public Customer(String firstName, String lastName, int
    custAge){
046
047             this.firstName = firstName;
048             this.lastName = lastName;
049             this.custAge = custAge;
050
051         }
052
```

```
053     }
054
055     // Creates an array of Customer Objects
056
057     private static Customer[] getCustomers(){
058
059         Customer[] customers = new Customer[5];
060
061         customers[0] = new Customer("John", "Smith", 21);
062         customers[1] = new Customer("Sally", "Smith", 30);
063         customers[2] = new Customer("Paul", "Ryan", 21);
064         customers[3] = new Customer("Mark", "Jacobs", 21);
065         customers[4] = new Customer("Steve", "Nash", 21);
066
067         return customers;
068
069     }
070
071     // Create the file and the PrintWriter that will write to the
    file
072
073     private static PrintWriter createFile(String fileName){
074
075         try{
076
077             // Creates a File object that allows you to work with
    files on the hardrive
078
079             File listOfNames = new File(fileName);
080
081             // FileWriter is used to write streams of characters to
    a file
082             // BufferedWriter gathers a bunch of characters and
    then writes
083             // them all at one time (Speeds up the Program)
084             // PrintWriter is used to write characters to the
    console, file
085
086             PrintWriter infoToWrite = new PrintWriter(
087             new BufferedWriter(
088                     new FileWriter(listOfNames)));
089             return infoToWrite;
090         }
091
092         // You have to catch this when you call FileWriter
093
094         catch(IOException e){
095
096             System.out.println("An I/O Error Occurred");
097
098             // Closes the program
099
100             System.exit(0);
101
102         }
103         return null;
104
```

```
105     }
106
107     // Create a string with the customer info and write it to the
    file
108
109     private static void createCustomers(Customer customer,
    PrintWriter custOutput){
110
111         // Create the String that contains the customer info
112
113         String custInfo = customer.firstName + " " +
    customer.lastName + " ";
114         custInfo += Integer.toString(customer.custAge);
115
116         // Writes the string to the file
117
118         custOutput.println(custInfo);
119
120     }
121
122     // Read info from the file and write it to the screen
123
124     private static void getFileInfo(){
125
126         System.out.println("Info Written to File\n");
127
128         // Open a new connection to the file
129
130         File listOfNames = new
    File("/Users/derekbanas/Documents/workspace3/Java
    Code/src/customers.txt");
131
132         try {
133
134             // FileReader reads character files
135             // BufferedReader reads as many characters as possible
136
137             BufferedReader getInfo = new BufferedReader(
138                     new FileReader(listOfNames));
139
140             // Reads a whole line from the file and saves it in a
    String
141
142             String custInfo = getInfo.readLine();
143
144             // readLine returns null when the end of the file is
    reached
145
146             while(custInfo != null){
147
148                 // System.out.println(custInfo);
149
150                 // Break lines into pieces
151
152                 String[] indivCustData = custInfo.split(" ");
153
154                 // Convert the String into an integer with parseInt
```

```
155
156                 int custAge = Integer.parseInt(indivCustData[2]);
157
158                 System.out.print("Customer " + indivCustData[0] + "
    is " + custAge +"\n");
159
160                 custInfo = getInfo.readLine();
161
162             }
163
164
165
166         }
167
168         // Can be thrown by FileReader
169
170         catch (FileNotFoundException e) {
171
172             System.out.println("Couldn't Find the File");
173             System.exit(0);
174         }
175
176         catch(IOException e){
177
178             System.out.println("An I/O Error Occurred");
179             System.exit(0);
180
181         }
182
183     }
184
185
186 }
```

```
001 // Class that will represent a system file name
002
003 import java.io.File;
004
005 // Used to write data to a file
006
007 import java.io.FileOutputStream;
008
009 // Triggered when an I/O error occurs
010
011 import java.io.IOException;
012
013 // Represents your XML document and contains useful methods
014
015 import org.jdom2.Document;
016
017 // Represents XML elements and contains useful methods
018
019 import org.jdom2.Element;
020
021 // Top level JDOM exception
022
023 import org.jdom2.JDOMException;
024
025 // Represents text used with JDOM
026
027 import org.jdom2.Text;
028
029 // Creates a JDOM document parsed using SAX Simple API for XML
030
031 import org.jdom2.input.SAXBuilder;
032
033 // Formats how the XML document will look
034
035 import org.jdom2.output.Format;
036
037 // Outputs the JDOM document to a file
038
039 import org.jdom2.output.XMLOutputter;
040
041 public class Lesson46 {
042     public static void main(String[] args) {
043
044         // writeXML();
045
046         readXML();
047     }
048
049     private static void readXML(){
050
051         SAXBuilder builder = new SAXBuilder();
052         try {
053
054             // Parses the file supplied into a JDOM document
055
056             Document readDoc = builder.build(new File("./src/jdomMade.xml"));
057
058             // Returns the root element for the document
059
060             System.out.println("Root: " + readDoc.getRootElement());
061
062             // Gets the text found between the name tags
063
064             System.out.println("Show: " +
    readDoc.getRootElement().getChild("show").getChildText("name"));
065
066             // Gets the attribute value for show_id assigned to name
067
068             System.out.println("Show ID: " +
    readDoc.getRootElement().getChild("show").getChild("name").getAttributeValue("show_id")
```

```
    + "\n");
069
070             Element root = readDoc.getRootElement();
071
072             // Cycles through all of the children in show and prints them
073
074             for (Element curEle : root.getChildren("show")) {
075               System.out.println("Show Name: " + curEle.getChildText("name"));
076               System.out.println("Show ID: " +
    curEle.getChild("name").getAttributeValue("show_id"));
077               System.out.print("On " + curEle.getChildText("network") + " in the ");
078               System.out.println(curEle.getChild("network").getAttributeValue("country")
    + "\n");
079             }
080
081         }
082
083         catch (JDOMException e) {
084             e.printStackTrace();
085         }
086
087         catch (IOException e) {
088             // TODO Auto-generated catch block
089             e.printStackTrace();
090         }
091
092     }
093
094     private static void writeXML(){
095     try{
096
097         // Creates a JDOM document
098
099         Document doc = new Document();
100
101         // Creates a element named tvshows and makes it the root
102
103         Element theRoot = new Element("tvshows");
104         doc.setRootElement(theRoot);
105
106         // Creates elements show and name
107
108         Element show = new Element("show");
109         Element name = new Element("name");
110
111         // Assigns an attribute to name and gives it a value
112
113         name.setAttribute("show_id", "show_001");
114
115         // Adds text between the name tags
116
117         name.addContent(new Text("Life On Mars"));
118
119         Element network = new Element("network");
120
121         network.addContent(new Text("ABC"));
122
123         network.setAttribute("country", "US");
124
125         // Adds name and network to the show tag
126
127         show.addContent(name);
128         show.addContent(network);
129
130         // Adds the show tag and all of its children to the root
131
132         theRoot.addContent(show);
133
134         // Add a new show element like above
135
136         Element show2 = new Element("show");
```

```
137
138         Element name2 = new Element("name");
139
140         name2.setAttribute("show_id", "show_002");
141
142         name2.addContent(new Text("Freaks and Geeks"));
143
144         Element network2 = new Element("network");
145
146         network2.addContent(new Text("ABC"));
147
148         network2.setAttribute("country", "US");
149
150         show2.addContent(name2);
151         show2.addContent(network2);
152
153         theRoot.addContent(show2);
154
155         // Uses indenting with pretty format
156
157         XMLOutputter xmlOutput = new XMLOutputter(Format.getPrettyFormat());
158
159         // Create a new file and write XML to it
160
161         xmlOutput.output(doc, new FileOutputStream(new File("./src/jdomMade.xml")));
162
163         System.out.println("Wrote to file");
164
165         }
166
167         catch (Exception e) {
168
169             e.printStackTrace();
170         }
171     }
172
173 }
174
175 /* Installing JDOM
176  *
177  * Download jdom.jar https://github.com/hunterhacker/jdom/downloads
178  * Download jdom-2.0.2.zip
179  *
180  * Right click default eclipse directory - Build Path
181  *
182  * Click Java Build Path
183  *
184  * Click Libraries tab - external libraries
185  *
186  * Add the JDOM libraries
187  *
188  *
189  * */
```

```
001 import java.util.regex.*;
002
003 public class LessonNineteen{
004
005     public static void main(String[] args){
006
007         String longString = " Derek Banas CA 12345 PA (412)555-1212
    johnsmith@hotmail.com 412-555-1234 412 555-1234 ";
008         String strangeString = " 1Z aaa **** *** {{{ {{ { ";
009
010         /*
011         [ ]  Insert characters that are valid
012         [^ ]  Insert characters that are not valid
013         \\s  Any white space
014         \\S  Any non white space
015         {n,m}  Whatever proceeds must occur between n and m times
016         */
017
018         // Word must contain letters that are 2 to 20 characters in
    length
019         // [A-Za-z]{2,20} 0r \w{2,20}
020
021         regexChecker("\\s[A-Za-z]{2,20}\\s", longString);
022
023         /*
024         \\d  Any digits 0-9
025         \\D  Anything not a number
026         {n}  Whatever proceeds must occur n times
027         */
028
029         // Only 5 digits
030         // \\s[0-9]{5}\\s or \\d{5}
031
032         regexChecker("\\s\\d{5}\\s", longString);
033
034         /*
035         |  Is used for OR clause situations
036         */
037
038         // Must start with a A or C, followed by 1 letter in
    brackets
039         // Must be a maximum of 2 characters in length
040         // A[KLRZ]|C[AOT]
041
042         regexChecker("A[KLRZ]|C[AOT]", longString);
043
044         /*
045         {n,}  Whatever proceeds must occur at least n times
046         +  Whatever proceeds must occur one or more times
047         . ^ * + ? { } [ ] \ | ( )  Characters that must be escaped
    or backslashed
048         */
049
050         // Grab any string that contains 1 or more !
051
052         regexChecker("(\\{{1,})", strangeString);
```

```
053         regexChecker("(\\{+)", strangeString);
054 
055         // Get anything that occurs 3 times except newline
056         // .  Anything but newline
057 
058         regexChecker(".{3}", strangeString);
059 
060         /*
061         \\w  Any word type character A-Z, a-z, 0-9, _
062         \\W  Any non word character
063         *  Occurs zero or more times
064         */
065 
066         regexChecker("\\w*", strangeString);
067 
068         regexChecker("[A-Za-z0-9._\\%-]+@[A-Za-z0-9.-]+\\.[A-Za-z]
    {2,4}", longString);
069 
070 
071         // ?  0 or 1 of what proceeds it
072 
073         regexChecker("([0-9]( |-)?)?(\\(?[0-9]{3}\\)?|[0-9]{3})( |-
    )?([0-9]{3}( |-)?[0-9]{4}|[a-zA-Z0-9]{7})", longString);
074 
075         regexReplace(longString);
076 
077     }
078 
079     public static void regexChecker(String theRegex, String
    str2Check){
080 
081         // You define your regular expression (REGEX) using Pattern
082 
083         Pattern checkRegex = Pattern.compile(theRegex);
084 
085         // Creates a Matcher object that searches the String for
086         // anything that matches the REGEX
087 
088         Matcher regexMatcher = checkRegex.matcher( str2Check );
089 
090         // Cycle through the positive matches and print them to
    screen
091         // Make sure string isn't empty and trim off any whitespace
092 
093         while ( regexMatcher.find() ){
094             if (regexMatcher.group().length() != 0){
095                 System.out.println( regexMatcher.group().trim() );
096 
097                 // You can get the starting and ending indexs
098 
099                 System.out.println( "Start Index: " +
    regexMatcher.start());
100                 System.out.println( "Start Index: " +
    regexMatcher.end());
101             }
102         }
103 
```

```
104         System.out.println();
105     }
106 
107     public static void regexReplace(String str2Replace){
108 
109         // REGEX that matches 1 or more white space
110 
111         Pattern replace = Pattern.compile("\\s+");
112 
113         // This doesn't really apply, but this is how you ignore
    case
114         // Pattern replace = Pattern.compile("\\s+",
    Pattern.CASE_INSENSITIVE);
115 
116         // trim the string t prepare it for a replace
117 
118         Matcher regexMatcher = replace.matcher(str2Replace.trim());
119 
120         // replaceAll replaces all white space with commas
121 
122         System.out.println(regexMatcher.replaceAll(", "));
123 
124     }
125 
126 }
```

```java
// Here I introduce the String class
// A String is an object unlike the other primitive data types

import java.util.Arrays;

public class LessonThirteen {

    public static void main(String[] args){

        // You create a String like this
        String randomString = "I'm just a random string";

        // If you want to use quotes in a string escape it with \
        // Always surround Strings with quotes " " and not Apostrophes ' '
        String gotToQuote ="He said, \"I'm here\"";

        /* Other common Escape Codes
         * \n : Newline
         * \b : Backspace
         * \' : Apostrophe
         * \" : Quote
         * \\ : Backslash
         */

        // You combine Strings with a +
        System.out.println(randomString + " " + gotToQuote);

        // You can add other data type to the string with a +
        int numTwo = 2;
        System.out.println(randomString + " " + numTwo);

        /* You convert primitive types to a string with toString
         * String byteString = Byte.toString(bigByte);
         * String shortString = Short.toString(bigByte);
         * String intString = Integer.toString(bigInt);
         * String longString = Long.toString(bigByte);
         * String floatString = Float.toString(bigByte);
         * String doubleString = Double.toString(bigByte);
         * String booleanString = Boolean.toString(bigByte);
         *
         * You convert from String to primitives with parse
         * int stringToInt = Integer.parseInt(intString);
         * parseSort, parseLong, parseByte, parseDouble,
         * parseBoolean, parseFloat
         */

        // You compare strings with equals or equalsIgnoreCase
        String uppercaseStr = "BIG";
        String lowercaseStr = "big";

        if(uppercaseStr.equals(lowercaseStr))
        {
            System.out.println("They're equal");
        }

```

```
056         if(uppercaseStr.equalsIgnoreCase(lowercaseStr))
057         {
058             System.out.println("Same letters");
059         }
060 
061         String letters = "abcde";
062         String moreLetters = "fghijk";
063 
064         // charAt returns the character in a string
065         System.out.println("2nd Character: " + letters.charAt(1));
066 
067         // compareTo returns 0 if strings are equal
068         // Returns a negative number if letters comes before
    moreLetters
069         // Returns a positive number if letters comes after
    moreLetters
070         // There is also a compareToIgnoreCase()
071         System.out.println(letters.compareTo(moreLetters));
072 
073         // contains() returns a boolean depending on whether the
074         // String contains the String you pass it
075         System.out.println(letters.contains("abc"));
076 
077         // endsWith() checks if the String ends with the String you
    pass
078         System.out.println(letters.endsWith("de"));
079 
080         // startsWith() works similar to endsWith()
081 
082         // indexOf() returns the 1st index that matches the String
    passed
083         System.out.println(letters.indexOf("cd"));
084 
085         // You can also specify the index to start searching from
086         // indexOf(StringToLookFor, IndexStartPosition)
087 
088         // lastIndexOf() works like indexOf except it starts from
    the
089         // end of the String you are searching
090 
091         // length() returns the number of characters in a String
092         System.out.println("Length of string: " +
    letters.length());
093 
094         // replace() replaces every occurrence of the first String
    with
095         // the second String you provide
096         System.out.println(letters.replace("abc", "xy"));
097 
098         // You can create an array of Strings with split()
099         // You define how to break up the String using a delimiter
100         // If you had 123,456 and used the delimiter "," you would
101         // create the array [123,456]
102         String[] letterArray = letters.split("");
103 
104         // toString() converts the array into a String to print it
105         System.out.println(Arrays.toString(letterArray));
```

```
106
107         // toCharArray() inserts every character in the string into
108         // separate indexes in an array
109         char[] charArray = letters.toCharArray();
110
111         System.out.println(Arrays.toString(charArray));
112
113         // substring() returns a String starting at the first index
114         // through the last index provided
115         System.out.println(letters.substring(1,4));
116
117         // toUpperCase() converts all letters into uppercase
118         // toLowerCase() does the opposite
119         System.out.println(letters.toUpperCase());
120
121         String randString = "   abc   ";
122
123         // trim() gets rid of leading and trailing white space
124         System.out.println(randString.trim());
125
126         // A String is immutable, which means every time you change
127         // a String a new version is created in memory.
128         // If you manipulate Strings allot use a StringBuilder
129
130         // How to create a StringBuilder
131         // It has a fixed space in memory
132         StringBuilder randSB = new StringBuilder("A random
    string");
133
134         // append() adds anything to the end of a SB
135         System.out.println(randSB.append(" again"));
136
137         // append() permanently effected the StringBuilder
138         System.out.println(randSB);
139
140         // delete() removes part of the SB from first index to the
    last
141         System.out.println(randSB.delete(15, 21));
142
143         // deleteCharAt(index) is used to delete individual chars
144
145         // capacity() returns the number of indexs for the SB
146         System.out.println(randSB.capacity());
147
148         // ensureCapacity() increases the capacity for the SB
149         randSB.ensureCapacity(60);
150         System.out.println(randSB.capacity());
151
152         // length() returns the number of characters in the SB
153         System.out.println(randSB.length());
154
155         // trimToSize() forces capacity to equal length
156         randSB.trimToSize();
157
158         // insert() inserts at the index you provide anything
159         System.out.println(randSB.insert(1, "nother"));
160
```

```
161         // toString converts a SB into a String
162         String oldSB = randSB.toString();
163 
164         /* StringBuilders also have the same methods as Strings
165          * charAt(), indexOf(), lastIndexOf(), subString()
166          */
167     }
168 
169 }
```

```
001 // Swing allows you to create Graphical User Interfaces
002 // You need the Swing library to create GUI interfaces
003 import java.awt.Dimension;
004 import java.awt.Toolkit;
005
006 import javax.swing.*;
007
008 // You must extend the JFrame class to make a frame
009
010 public class LessonTwenty extends JFrame{
011
012     public static void main(String[] args){
013
014         new LessonTwenty();
015
016     }
017
018     public LessonTwenty(){
019
020         // Define the size of the frame
021         this.setSize(400, 400);
022
023         // Opens the frame in the middle of the screen----------
    ------------------------------------
024         // You could also define position based on a
    component                    |
025
026         // this.setLocationRelativeTo(null);
027
028
029         // Toolkit is the super class for the Abstract Window
    Toolkit
030         // It allows us to ask questions of the OS
031
032         Toolkit tk = Toolkit.getDefaultToolkit();
033
034         // A Dimension can hold the width and height of a component
035         // getScreenSize returns the size of the screen
036
037         Dimension dim = tk.getScreenSize();
038
039         // dim.width returns the width of the screen
040         // this.getWidth returns the width of the frame you are
    making
041
042          int xPos = (dim.width / 2) - (this.getWidth() / 2);
043          int yPos = (dim.height / 2) - (this.getHeight() / 2);
044
045         // You could also define the x, y position of the frame
046
047          this.setLocation(xPos, yPos);
048
049
050         // Define how the user exits the program
051         // This closes when they click the close button
052
```

```
053         // Define if the user can resize the frame (true by
    default)
054         this.setResizable(false);
055
056         // Define how the frame exits (Click the Close Button)
057         // Without this Java will eventually close the app
058
059         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
060
061         // Define the title for the frame
062
063         this.setTitle("My First Frame");
064
065         // The JPanel contains all of the components for your frame
066
067         JPanel thePanel = new JPanel();
068
069         // How to create a label with its text ----------
070
071         JLabel label1 = new JLabel("Tell me something");
072
073         // How to change the text for the label
074
075         label1.setText("New Text");
076
077         // How to create a tool tip for the label
078
079         label1.setToolTipText("Doesn't do anything");
080
081         // How to add the label to the panel
082
083         thePanel.add(label1);
084
085         // How to create a button -----------------------
086
087         JButton button1 = new JButton("Send");
088
089         // How to hide the button border (Default True)
090
091         button1.setBorderPainted(false);
092
093         // How to hide the button background (Default True)
094
095         button1.setContentAreaFilled(false);
096
097         // How to change the text for the label
098
099         button1.setText("New Button");
100
101         // How to create a tool tip for the label
102
103         button1.setToolTipText("Doesn't do anything either");
104
105         thePanel.add(button1);
106
107         // How to add a textfield ----------------------
108
```

```
109         JTextField textField1 = new JTextField("Type Here", 15);
110 
111         // Change the size of the text field
112 
113         textField1.setColumns(10);
114 
115         // Change the initial value of the text field
116 
117         textField1.setText("New Text Here");
118 
119         // Change the tool tip for the text field
120 
121         textField1.setToolTipText("More of nothing");
122 
123         thePanel.add(textField1);
124 
125         // How to add a text area ----------------------
126 
127         JTextArea textArea1 = new JTextArea(15, 20);
128 
129         // Set the default text for the text area
130 
131         textArea1.setText("Just a whole bunch of text that is
    important\n");
132 
133         // If text doesn't fit on a line, jump to the next
134 
135         textArea1.setLineWrap(true);
136 
137         // Makes sure that words stay intact if a line wrap occurs
138 
139         textArea1.setWrapStyleWord(true);
140 
141         // Gets the number of newlines in the text
142 
143         int numOfLines = textArea1.getLineCount();
144 
145         // Appends text after the current text
146 
147         textArea1.append(" number of lines: " + numOfLines);
148 
149         // Adds scroll bars to the text area ----------
150 
151         JScrollPane scrollbar1 = new JScrollPane(textArea1,
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
152 
153         // Other options: VERTICAL_SCROLLBAR_ALWAYS,
    VERTICAL_SCROLLBAR_NEVER
154 
155         thePanel.add(scrollbar1);
156 
157         // How to add the panel to the frame
158 
159         this.add(thePanel);
160 
161         // Makes the frame show on the screen
```

```
162
163         this.setVisible(true);
164
165         // Gives focus to the textfield
166
167         textField1.requestFocus();
168
169         // You can also request focus for the text array
170         // textArea1.requestFocus();
171
172     }
173
174 }
```

```
001 import javax.swing.*;
002
003 import java.awt.BorderLayout;
004 import java.awt.Dimension;
005 import java.awt.FlowLayout;
006
007 public class Lesson28 extends JFrame{
008
009     JButton button1, button2, button3, button4, button5;
010     String outputString = "";
011
012     public static void main(String[] args){
013
014         new Lesson28();
015
016     }
017
018     public Lesson28(){
019
020         // Create the frame, position it and handle closing it
021
022         this.setSize(400,400);
023         this.setLocationRelativeTo(null);
024         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
025         this.setTitle("My Sixth Frame");
026
027         /* FLOW LAYOUT
028         // Create a flow layout (Default)
029         JPanel thePanel = new JPanel();
030
031         // Define the flow layout alignment
032         // FlowLayout.RIGHT, FlowLayout.CENTER
033
034         thePanel.setLayout(new FlowLayout(FlowLayout.LEFT));
035
036         // You can also define the pixels that separate the
    components
037         // FlowLayout(alignment, horz gap, vertical gap)
038
039          button1 = new JButton("Button 1");
040          thePanel.add(button1);
041
042          END FLOW LAYOUT*/
043
044         /* BORDER LAYOUT */
045
046         JPanel thePanel = new JPanel();
047
048         thePanel.setLayout(new BorderLayout());
049
050         // Create buttons
051
052         button1 = new JButton("Button 1");
053         button2 = new JButton("Button 2");
054         button3 = new JButton("Button 3");
055         button4 = new JButton("Button 4");
```

```
056         button5 = new JButton("Button 5");
057
058         // If you put components in the same space the
059         // last one in stays and everything else goes
060         // EX.
061         // thePanel.add(button1, BorderLayout.NORTH);
062         // thePanel.add(button2, BorderLayout.NORTH);
063         // Only button2 shows
064         /*
065
066         thePanel.add(button1, BorderLayout.NORTH);
067         thePanel.add(button2, BorderLayout.SOUTH);
068         thePanel.add(button3, BorderLayout.EAST);
069         thePanel.add(button4, BorderLayout.WEST);
070         thePanel.add(button5, BorderLayout.CENTER);
071          */
072
073         /* If you want more than one component to show
074         // up in the same part of a border layout put
075         // them in a panel and then add the panel to
076         // the border layout panel
077
078
079         JPanel thePanel2 = new JPanel();
080
081         thePanel2.add(button1);
082         thePanel2.add(button2);
083
084         thePanel.add(thePanel2, BorderLayout.NORTH);
085         */
086
087         /* BOX LAYOUT */
088
089         Box theBox = Box.createHorizontalBox();
090
091         // You can also use Box theBox = Box.createVerticalBox();
092         /*
093         theBox.add(button1);
094         theBox.add(button2);
095         theBox.add(button3);
096         theBox.add(button4);
097         */
098
099         // You can also separate the components with struts
100         /*
101         theBox.add(button1);
102         theBox.add(Box.createHorizontalStrut(4));
103         theBox.add(button2);
104         theBox.add(Box.createHorizontalStrut(4));
105         theBox.add(button3);
106         theBox.add(Box.createHorizontalStrut(4));
107         theBox.add(button4);
108         */
109
110         // A rigid area gives you the option to space using
111         // horizontal and vertical spacing
112
```

```
113         theBox.add(button1);
114         theBox.add(button2);
115         // theBox.add(Box.createRigidArea(new Dimension(30, 20)));
116 
117         // When you use a glue you position the components as
118         // far apart as possible while remaining on the screen
119         // There is also a createVerticalGlue
120 
121         theBox.add(Box.createHorizontalGlue());
122         theBox.add(button3);
123 
124         this.add(theBox);
125 
126         // this.add(thePanel); // Don't use for BOX LAYOUT
127 
128         this.setVisible(true);
129     }
130 
131 }
```

```
001 import javax.swing.*;
002
003 import java.awt.event.*;
004
005 public class Lesson24 extends JFrame{
006
007     JComboBox favoriteShows;
008     JButton button1;
009     String infoOnComponent = "";
010
011     public static void main(String[] args){
012
013         new Lesson24();
014
015     }
016
017     public Lesson24(){
018
019         this.setSize(400,400);
020
021         this.setLocationRelativeTo(null);
022
023         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
024
025         this.setTitle("My Fourth Frame");
026
027         JPanel thePanel = new JPanel();
028
029         // Create an array that will be added to the combo box
030
031         String[] shows = {"Breaking Bad", "Life on Mars", "Doctor
    Who"};
032
033         // Create a combo box and add the array of shows
034
035         favoriteShows = new JComboBox(shows);
036
037         // Add an item to the combo box
038
039         favoriteShows.addItem("Pushing Daisies");
040
041         thePanel.add(favoriteShows);
042
043         // Create a button
044
045         button1 = new JButton("Get Answer");
046
047         ListenForButton lForButton = new ListenForButton();
048
049         button1.addActionListener(lForButton);
050
051         thePanel.add(button1);
052
053         this.add(thePanel);
054
055         this.setVisible(true);
```

```
056
057         // Add an item to a combo box at index 1
058
059         favoriteShows.insertItemAt("Dexter", 1);
060
061         // Only show 3 items at a time
062
063         favoriteShows.setMaximumRowCount(3);
064
065         // Remove the item named Dexter
066
067         //favoriteShows.removeItem("Dexter");
068
069         // Remove the item at index 1
070
071         //favoriteShows.removeItemAt(1);
072
073         // Remove all items
074
075         // favoriteShows.removeAllItems();
076
077             }
078
079         private class ListenForButton implements ActionListener{
080
081             public void actionPerformed(ActionEvent e){
082
083                 if(e.getSource() == button1){
084
085
086                     favoriteShows.setEditable(true);
087
088                     // Get item at index 0
089
090                     infoOnComponent = "Item at 0: " +
    favoriteShows.getItemAt(0) + "\n";
091
092                     // Get the number of items in the combo box
093
094                     infoOnComponent += "Num of Shows: " +
    favoriteShows.getItemCount() + "\n";
095
096                     // Get the index for the selected item
097
098                     infoOnComponent += "Selected ID: " +
    favoriteShows.getSelectedIndex() + "\n";
099
100                     // Get the value for the selected item
101
102                     infoOnComponent += "Selected Show: " +
    favoriteShows.getSelectedItem() + "\n";
103
104                     // Find out if the values in the combo box are
    editable
105
106                     infoOnComponent += "Combo Box Editable: " +
    favoriteShows.isEditable() + "\n";
```

```
107
108                     JOptionPane.showMessageDialog(Lesson24.this,
    infoOnComponent, "Information", JOptionPane.INFORMATION_MESSAGE);
109
110                     infoOnComponent = "";
111
112                 }
113             }
114         }
115
116 }
```

```
001 import java.awt.*;
002
003 import javax.swing.*;
004 import javax.swing.event.*;
005
006 import java.util.Calendar;
007 import java.util.Date;
008
009
010 public class Lesson30 extends JFrame{
011
012     JLabel nameLabel, streetLabel, stateLabel, dateLabel,
013         ageLabel, sexLabel, optionsLabel, aboutLabel;
014     JTextField nameText, streetText;
015     JComboBox stateList;
016     JSpinner dateSpin;
017     JSlider ageSlider;
018     JRadioButton maleRadio, femaleRadio;
019     ButtonGroup sexGroup;
020     JCheckBox morningCheck, afterNCheck, eveningCheck;
021     JTextArea aboutYou;
022
023
024     public static void main(String[] args){
025
026         new Lesson30();
027
028     }
029
030     public Lesson30(){
031
032         // Create the frame, position it and handle closing it
033
034         this.setLocationRelativeTo(null);
035         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
036         this.setTitle("Random Layout");
037
038         JPanel thePanel = new JPanel();
039
040         thePanel.setLayout(new GridBagLayout());
041
042         nameLabel = new JLabel("  Name:");
043
044         addComp(thePanel, nameLabel, 0, 0, 1, 1,
    GridBagConstraints.EAST, GridBagConstraints.NONE);
045
046         nameText = new JTextField(30);
047
048         addComp(thePanel, nameText, 1, 0, 2, 1,
    GridBagConstraints.WEST, GridBagConstraints.NONE);
049
050         streetLabel = new JLabel("  Street:");
051
052         addComp(thePanel, streetLabel, 0, 1, 1, 1,
    GridBagConstraints.EAST, GridBagConstraints.NONE);
053
```

```
054         streetText = new JTextField(30);
055 
056         addComp(thePanel, streetText, 1, 1, 2, 1,
    GridBagConstraints.WEST, GridBagConstraints.NONE);
057 
058         // Create a set of radio buttons and wrap them in a group
059 
060         Box sexBox = Box.createVerticalBox();
061         maleRadio = new JRadioButton("Male");
062         femaleRadio = new JRadioButton("Female");
063         ButtonGroup sexGroup = new ButtonGroup();
064         sexGroup.add(maleRadio);
065         sexGroup.add(femaleRadio);
066         sexBox.add(maleRadio);
067         sexBox.add(femaleRadio);
068         sexBox.setBorder(BorderFactory.createTitledBorder("Sex"));
069         addComp(thePanel, sexBox, 3, 0, 2, 1,
    GridBagConstraints.WEST, GridBagConstraints.NONE);
070 
071         // Create a flow layout panel and space components 10px
    apart
072 
073         JPanel statePanel = new JPanel();
074         statePanel.setLayout(new FlowLayout(FlowLayout.LEFT, 10,
    0));
075 
076         stateLabel = new JLabel("State");
077 
078         statePanel.add(stateLabel);
079 
080         // Create a combo box that will hold the states
081 
082         String[] states = {"PA", "NY", "OH", "WV", "NJ"};
083 
084         stateList = new JComboBox(states);
085 
086         statePanel.add(stateList);
087 
088         dateLabel = new JLabel("Appointment Date");
089 
090         statePanel.add(dateLabel);
091 
092         // Get todays date
093 
094         Date todaysDate = new Date();
095 
096         // Create a date spinner & set default to today, no
    minimum, or max
097         // Increment the days on button presses
098         // Can also increment YEAR, MONTH, or DAY_OF_MONTH
099 
100         dateSpin = new JSpinner(new SpinnerDateModel(todaysDate,
    null, null,
101                         Calendar.DAY_OF_MONTH));
102 
103         // DateEditor is an editor that handles displaying &
    editing the dates
```

```
104
105         JSpinner.DateEditor dateEditor = new
    JSpinner.DateEditor(dateSpin, "dd/MM/yy");
106         dateSpin.setEditor(dateEditor);
107
108         statePanel.add(dateSpin);
109
110         ageLabel = new JLabel("Age: 50");
111
112         statePanel.add(ageLabel);
113
114         // Creates a slider with a min value of 1 thru 100
115         // and an initial value of 1
116
117         ageSlider = new JSlider(1, 99, 50);
118
119         // Create an instance of ListenForEvents to handle events
120
121         ListenForSlider lForSlider = new ListenForSlider();
122
123         // Tell Java that you want to be alerted when an event
124         // occurs on the slider
125
126         ageSlider.addChangeListener(lForSlider);
127
128         statePanel.add(ageSlider);
129
130         addComp(thePanel, statePanel, 1, 2, 5, 1,
    GridBagConstraints.EAST, GridBagConstraints.NONE);
131
132         // Create check boxs and assign them to a group
133
134         JCheckBox morningCheck, afterNCheck, eveningCheck;
135
136         Box optionBox = Box.createVerticalBox();
137         morningCheck = new JCheckBox("Morning");
138         afterNCheck = new JCheckBox("Afternoon");
139         eveningCheck = new JCheckBox("Evening");
140
141         optionBox.add(morningCheck);
142         optionBox.add(afterNCheck);
143         optionBox.add(eveningCheck);
144         optionBox.setBorder(BorderFactory.createTitledBorder("Time
    of Day"));
145         addComp(thePanel, optionBox, 1, 3, 2, 1,
    GridBagConstraints.NORTHWEST, GridBagConstraints.NONE);
146
147         // Create a text area that is 6 columns high and 40 long
148
149         aboutYou = new JTextArea(6, 40);
150
151         // Set the default text for the text area
152
153         aboutYou.setText("Tell us something about you");
154
155         // If text doesn't fit on a line, jump to the next
156
```

```
157         aboutYou.setLineWrap(true);
158 
159         // Makes sure that words stay intact if a line wrap occurs
160 
161         aboutYou.setWrapStyleWord(true);
162 
163         // Adds scroll bars to the text area ----------
164 
165         JScrollPane scrollbar1 = new JScrollPane(aboutYou,
    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
166 
167         // Other options: VERTICAL_SCROLLBAR_ALWAYS,
    VERTICAL_SCROLLBAR_NEVER
168 
169         addComp(thePanel, scrollbar1, 2, 3, 3, 1,
    GridBagConstraints.EAST, GridBagConstraints.NONE);
170 
171         this.add(thePanel);
172 
173         // Adjusts the size of the frame to best work for the
    components
174 
175         this.pack();
176 
177         this.setVisible(true);
178 
179         // Define if the user can resize the frame (true by
    default)
180         this.setResizable(false);
181 
182     }
183 
184     // Sets the rules for a component destined for a GridBagLayout
185     // and then adds it
186 
187     private void addComp(JPanel thePanel, JComponent comp, int
    xPos, int yPos, int compWidth, int compHeight, int place, int
    stretch){
188 
189         GridBagConstraints gridConstraints = new
    GridBagConstraints();
190 
191         gridConstraints.gridx = xPos;
192         gridConstraints.gridy = yPos;
193         gridConstraints.gridwidth = compWidth;
194         gridConstraints.gridheight = compHeight;
195         gridConstraints.weightx = 100;
196         gridConstraints.weighty = 100;
197         gridConstraints.insets = new Insets(5,5,5,5);
198         gridConstraints.anchor = place;
199         gridConstraints.fill = stretch;
200 
201         thePanel.add(comp, gridConstraints);
202 
203     }
204 
```

```
205     // Implements ActionListener so it can react to events on
    components
206
207     private class ListenForSlider implements ChangeListener{
208
209     // Called when the spinner is changed
210         public void stateChanged(ChangeEvent e) {
211
212             // Check if the source of the event was the button
213
214             if(e.getSource() == ageSlider){
215
216                 // Change the value for the label next to the
    spinner
217
218                 ageLabel.setText("Age: " + ageSlider.getValue() );
219
220
221             }
222
223         }
224
225
226
227     }
228
229 }
```

```
001 import java.awt.Dimension;
002 import java.awt.Toolkit;
003 import javax.swing.*;
004
005 import java.awt.event.*;
006
007 // Extends JFrame so it can create frames
008
009 public class LessonTwentyOne extends JFrame{
010
011     JButton button1;
012     JTextField textField1;
013     JTextArea textArea1;
014     int buttonClicked;
015
016     public static void main(String[] args){
017
018         new LessonTwentyOne();
019
020     }
021
022     public LessonTwentyOne(){
023
024         // Define the size of the frame
025         this.setSize(400, 400);
026
027         // Toolkit is the super class for the Abstract Window
    Toolkit
028         // It allows us to ask questions of the OS
029
030         Toolkit tk = Toolkit.getDefaultToolkit();
031
032         // A Dimension can hold the width and height of a component
033         // getScreenSize returns the size of the screen
034
035         Dimension dim = tk.getScreenSize();
036
037         // dim.width returns the width of the screen
038         // this.getWidth returns the width of the frame you are
    making
039
040         int xPos = (dim.width / 2) - (this.getWidth() / 2);
041         int yPos = (dim.height / 2) - (this.getHeight() / 2);
042
043         // You could also define the x, y position of the frame
044
045         this.setLocation(xPos, yPos);
046
047         // Define how the frame exits (Click the Close Button)
048         // Without this Java will eventually close the app
049
050         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
051
052         // Define the title for the frame
053
054         this.setTitle("My First Frame");
```

```
055
056         // The JPanel contains all of the components for your frame
057
058         JPanel thePanel = new JPanel();
059
060         // Create a button with Click Here on it
061
062         button1 = new JButton("Click Here");
063
064         // Create an instance of ListenForEvents to handle events
065
066         ListenForButton lForButton = new ListenForButton();
067
068         // Tell Java that you want to be alerted when an event
069         // occurs on the button
070
071         button1.addActionListener(lForButton);
072
073         thePanel.add(button1);
074
075         // How to add a text field ----------------------
076
077         textField1 = new JTextField("Type Here", 15);
078
079         ListenForKeys lForKeys = new ListenForKeys();
080
081         textField1.addKeyListener(lForKeys);
082
083         thePanel.add(textField1);
084
085         // How to add a text area ----------------------
086
087         textArea1 = new JTextArea(15, 20);
088
089         // Set the default text for the text area
090
091         textArea1.setText("Tracking Events\n");
092
093         // If text doesn't fit on a line, jump to the next
094
095         textArea1.setLineWrap(true);
096
097         // Makes sure that words stay intact if a line wrap occurs
098
099         textArea1.setWrapStyleWord(true);
100
101         // Adds scroll bars to the text area ----------
102
103         JScrollPane scrollbar1 = new JScrollPane(textArea1,
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
    JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
104
105         // Other options: VERTICAL_SCROLLBAR_ALWAYS,
    VERTICAL_SCROLLBAR_NEVER
106
107         thePanel.add(scrollbar1);
108
```

```
109         this.add(thePanel);
110 
111         ListenForWindow lForWindow = new ListenForWindow();
112 
113         this.addWindowListener(lForWindow);
114 
115         this.setVisible(true);
116 
117         // Track the mouse if it is inside of the panel
118 
119         ListenForMouse lForMouse = new ListenForMouse();
120 
121         thePanel.addMouseListener(lForMouse);
122 
123 
124 
125     }
126 
127     // Implements ActionListener so it can react to events on
    components
128 
129     private class ListenForButton implements ActionListener{
130 
131     // This method is called when an event occurs
132 
133     public void actionPerformed(ActionEvent e){
134 
135         // Check if the source of the event was the button
136 
137         if(e.getSource() == button1){
138 
139             buttonClicked++;
140 
141             // Change the text for the label
142 
143             textArea1.append("Button clicked " + buttonClicked + "
    times\n" );
144 
145             // e.getSource().toString() returns information on the
    button
146             // and the event that occurred
147 
148         }
149 
150     }
151 
152     }
153 
154     // By using KeyListener you can track keys on the keyboard
155 
156     private class ListenForKeys implements KeyListener{
157 
158         // Handle the key typed event from the text field.
159         public void keyTyped(KeyEvent e) {
160             textArea1.append("Key Hit: " + e.getKeyChar() + "\n");
161         }
162 
```

```
163         // Handle the key-pressed event from the text field.
164         public void keyPressed(KeyEvent e) {
165 
166         }
167 
168         // Handle the key-released event from the text field.
169         public void keyReleased(KeyEvent e) {
170 
171         }
172 
173     }
174 
175     private class ListenForMouse implements MouseListener{
176 
177         // Called when a mouse button is clicked
178 
179         public void mouseClicked(MouseEvent e) {
180 
181             textArea1.append("Mouse Panel Pos: " + e.getX() + " " +
    e.getY() + "\n");
182             textArea1.append("Mouse Screen Pos: " +
    e.getXOnScreen() + " " + e.getYOnScreen() + "\n");
183             textArea1.append("Mouse Button: " + e.getButton()  +
    "\n");
184             textArea1.append("Mouse Clicks: " + e.getClickCount()
    + "\n");
185 
186         }
187 
188         // Called when the mouse enters the component assigned
189         // the MouseListener
190 
191         public void mouseEntered(MouseEvent arg0) {
192             // TODO Auto-generated method stub
193 
194         }
195 
196         // Called when the mouse leaves the component assigned
197         // the MouseListener
198 
199         public void mouseExited(MouseEvent arg0) {
200             // TODO Auto-generated method stub
201 
202         }
203 
204         // Mouse button pressed
205 
206         public void mousePressed(MouseEvent arg0) {
207             // TODO Auto-generated method stub
208 
209         }
210 
211         // Mouse button released
212 
213         public void mouseReleased(MouseEvent arg0) {
214             // TODO Auto-generated method stub
215 
```

```
216         }
217
218     }
219
220     private class ListenForWindow implements WindowListener{
221
222         // Called when window is the active window
223
224         public void windowActivated(WindowEvent e) {
225             textArea1.append("Window Activated\n");
226
227         }
228
229         // Called when window is closed using dispose
230         // this.dispose(); can be used to close a window
231
232         public void windowClosed(WindowEvent arg0) {
233             // TODO Auto-generated method stub
234
235         }
236
237         // Called when the window is closed from the menu
238
239         public void windowClosing(WindowEvent arg0) {
240             // TODO Auto-generated method stub
241
242         }
243
244         // Called when a window is no longer the active window
245
246         public void windowDeactivated(WindowEvent e) {
247             textArea1.append("Window Activated\n");
248
249         }
250
251         // Called when the window goes from minimized to a normal
    state
252
253         public void windowDeiconified(WindowEvent arg0) {
254             textArea1.append("Window in Normal State\n");
255
256         }
257
258         // Called when the window goes from normal to a minimized
    state
259
260         public void windowIconified(WindowEvent arg0) {
261             textArea1.append("Window Minimized\n");
262
263         }
264
265         // Called when the window is first created
266
267         public void windowOpened(WindowEvent arg0) {
268             textArea1.append("Window Created\n");
269
270         }
```

```
271
272         }
273
274 }
```

```
001 import java.awt.Font;
002 import java.awt.GridBagConstraints;
003 import java.awt.GridBagLayout;
004 import java.awt.GridLayout;
005 import java.awt.Insets;
006
007 import javax.swing.*;
008
009 public class Lesson29 extends JFrame{
010
011     JButton but1, but2, but3, but4, but5, but6,
012         but7, but8, but9, but0, butPlus, butMinus,
013         clearAll;
014
015     JTextField textResult;
016
017     int num1, num2;
018
019     public static void main(String[] args){
020
021         new Lesson29();
022
023     }
024
025     public Lesson29(){
026
027         // Create the frame, position it and handle closing it
028
029         this.setSize(400,400);
030         this.setLocationRelativeTo(null);
031         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
032         this.setTitle("Calculator");
033
034         JPanel thePanel = new JPanel();
035
036         // Create a Grid Layout with as many rows as needed
037         // and 3 columns. The last 2 parameters create a
038         // horizontal gap of 2 pixels and a vertical gap of
039         // 2 pixels.
040
041         // thePanel.setLayout(new GridLayout(0,3,2,2));
042
043         thePanel.setLayout(new GridBagLayout());
044
045         // You create a GridBagContraints object that defines
046         // defaults for your components
047
048         GridBagConstraints gridConstraints = new
    GridBagConstraints();
049
050         // Define the x position of the component
051
052         gridConstraints.gridx = 1;
053
054         // Define the y position of the component
055
```

```
056         gridConstraints.gridy = 1;
057
058         // Number of columns the component takes up
059
060         gridConstraints.gridwidth = 1;
061
062         // Number of rows the component takes up
063
064         gridConstraints.gridheight = 1;
065
066         // Gives the layout manager a hint on how to adjust
067         // component width (0 equals fixed)
068
069         gridConstraints.weightx = 50;
070
071         // Gives the layout manager a hint on how to adjust
072         // component height (0 equals fixed)
073
074         gridConstraints.weighty = 100;
075
076         // Defines padding top, left, bottom, right
077
078         gridConstraints.insets = new Insets(5,5,5,5);
079
080         // Defines where to place components if they don't
081         // fill the space: CENTER, NORTH, SOUTH, EAST, WEST
082         // NORTHEAST, etc.
083
084         gridConstraints.anchor = GridBagConstraints.CENTER;
085
086         // How should the component be stretched to fill the
087         // space: NONE, HORIZONTAL, VERTICAL, BOTH
088
089         gridConstraints.fill = GridBagConstraints.BOTH;
090
091         textResult = new JTextField("0",20);
092
093         // Defines the font to use in the text field
094
095         Font font = new Font("Helvetica", Font.PLAIN, 18);
096         textResult.setFont(font);
097
098         but1 = new JButton("1");
099         but2 = new JButton("2");
100         but3 = new JButton("3");
101         but4 = new JButton("4");
102         but5 = new JButton("5");
103         but6 = new JButton("6");
104         but7 = new JButton("7");
105         but8 = new JButton("8");
106         but9 = new JButton("9");
107         butPlus = new JButton("+");
108         but0 = new JButton("0");
109         butMinus = new JButton("-");
110         clearAll = new JButton("C");
111
112         thePanel.add(clearAll,gridConstraints);
```

```
113         gridConstraints.gridwidth = 20;
114         gridConstraints.gridx = 5;
115         thePanel.add(textResult,gridConstraints);
116         gridConstraints.gridwidth = 1;
117         gridConstraints.gridx = 1;
118         gridConstraints.gridy = 2;
119         thePanel.add(but1,gridConstraints);
120         gridConstraints.gridx = 5;
121         thePanel.add(but2,gridConstraints);
122         gridConstraints.gridx = 9;
123         thePanel.add(but3,gridConstraints);
124         gridConstraints.gridx = 1;
125         gridConstraints.gridy = 3;
126         thePanel.add(but4,gridConstraints);
127         gridConstraints.gridx = 5;
128         thePanel.add(but5,gridConstraints);
129         gridConstraints.gridx = 9;
130         thePanel.add(but6,gridConstraints);
131         gridConstraints.gridx = 1;
132         gridConstraints.gridy = 4;
133         thePanel.add(but7,gridConstraints);
134         gridConstraints.gridx = 5;
135         thePanel.add(but8,gridConstraints);
136         gridConstraints.gridx = 9;
137         thePanel.add(but9,gridConstraints);
138         gridConstraints.gridx = 1;
139         gridConstraints.gridy = 5;
140         thePanel.add(butPlus,gridConstraints);
141         gridConstraints.gridx = 5;
142         thePanel.add(but0,gridConstraints);
143         gridConstraints.gridx = 9;
144         thePanel.add(butMinus,gridConstraints);
145
146
147         // Adding buttons using the Grid Layout
148
149         /*
150         thePanel.add(but1);
151         thePanel.add(but2);
152         thePanel.add(but3);
153         thePanel.add(but4);
154         thePanel.add(but5);
155         thePanel.add(but6);
156         thePanel.add(but7);
157         thePanel.add(but8);
158         thePanel.add(but9);
159         thePanel.add(butPlus);
160         thePanel.add(but0);
161         thePanel.add(butMinus);
162         */
163
164         this.add(thePanel);
165
166         this.setVisible(true);
167
168     } // END OF Lesson29 CONSTRUCTOR
169
```

```
170 } // END OF Lesson29 CLASS
```

```
001 import javax.swing.*;
002
003 import java.awt.event.*;
004
005 public class Lesson25 extends JFrame{
006
007     JButton button1;
008     String infoOnComponent = "";
009     JList favoriteMovies, favoriteColors;
010     DefaultListModel defListModel = new DefaultListModel();
011     JScrollPane scroller;
012
013     public static void main(String[] args){
014
015         new Lesson25();
016
017     }
018
019     public Lesson25(){
020
021         this.setSize(400,400);
022
023         this.setLocationRelativeTo(null);
024
025         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
026
027         this.setTitle("My Fifth Frame");
028
029         JPanel thePanel = new JPanel();
030
031         // Create a button
032
033         button1 = new JButton("Get Answer");
034
035         ListenForButton lForButton = new ListenForButton();
036
037         button1.addActionListener(lForButton);
038
039         thePanel.add(button1);
040
041         String[] movies = {"Matrix", "Minority Report", "Big"};
042
043         // Creating a List Box
044
045         favoriteMovies = new JList(movies);
046
047         // Define the height of each cell
048
049         favoriteMovies.setFixedCellHeight(30);
050
051         // Define the width of each cell
052
053         favoriteMovies.setFixedCellWidth(150);
054
055         // Define how many selections can be made
056         // MULTIPLE_INTERVAL_SELECTION: Select what ever you want
057         // SINGLE_SELECTION: Select only one
058         // SINGLE_INTERVAL_SELECTION: Select as many as you want if in order
059
060         favoriteMovies.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
061
062         // All the methods for lists
063         /*
064          * getSelectedIndex(): returns the index for the first selected item
065          * getSelectedIndexes(): returns every selection in a list
066          * getSelectedValue(): returns the value of the first selected
067          * getSelectedValues(): returns an array of all values
068          * isSelectedIndex(): returns true if index is selected
```

```
069          */
070
071         // You can't change items in a list unless you store the items
072         // in a DefaultListModel
073
074         String[] colors = {"Black", "Blue", "White", "Green", "Orange", "Gray",
    "Pink"};
075
076         // How to load a String array into a DefaultListModel
077
078         for(String color: colors){
079             defListModel.addElement(color);
080         }
081
082         // Add item named Purple to index number 2
083
084         defListModel.add(2, "Purple");
085
086         // Create a List box filled with items in the DefaultListModel
087
088         favoriteColors = new JList(defListModel);
089
090         // Only display 4 items at a time
091
092         favoriteColors.setVisibleRowCount(4);
093
094         // Create a scroll bar panel to hold the list box
095
096         scroller = new JScrollPane(favoriteColors,
097                 JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
098                 JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
099
100         // Define the height of each cell
101
102         favoriteColors.setFixedCellHeight(30);
103
104         // Define the width of each cell
105
106         favoriteColors.setFixedCellWidth(150);
107
108         thePanel.add(favoriteMovies);
109
110         // You add the scroll bar container, not the list
111
112         thePanel.add(scroller);
113
114         this.add(thePanel);
115
116         this.setVisible(true);
117
118     }
119
120         private class ListenForButton implements ActionListener{
121
122             public void actionPerformed(ActionEvent e){
123
124                 if(e.getSource() == button1){
125
126                     // contains returns true if the item is in the list
127
128                     if(defListModel.contains("Black")) infoOnComponent += "Black is
    here\n";
129
130                     // Check if the list isn't empty
131
132                     if(!defListModel.isEmpty()) infoOnComponent += "Isn't Empty\n";
133
134                     // return the number of items in the DefaultListModel
135
```

```
136                     infoOnComponent += "Elements in the list " + defListModel.size()
    + "\n";
137
138                     // return the first element in the list
139
140                     infoOnComponent += "Last Element " + defListModel.firstElement()
    + "\n";
141
142                     // return the last element in the list
143
144                     infoOnComponent += "Last Element " + defListModel.lastElement()
    + "\n";
145
146                     // return the last element in the list
147
148                     infoOnComponent += "Element in index 1 " + defListModel.get(1) +
    "\n";
149
150                     // Remove the item in index 0
151
152                     defListModel.remove(0);
153
154                     // Remove the item named Big
155
156                     defListModel.removeElement("Blue");
157
158                     // Create an array filled with the list items
159
160                     Object[] arrayOfList = defListModel.toArray();
161
162                     // Iterate through the array
163
164                     for(Object color: arrayOfList){
165                         infoOnComponent += color + "\n";
166                     }
167
168                     JOptionPane.showMessageDialog(Lesson25.this, infoOnComponent,
    "Information", JOptionPane.INFORMATION_MESSAGE);
169
170                     infoOnComponent = "";
171
172                 }
173             }
174         }
175
176 }
```

```
001 import javax.swing.*;
002
003 import java.awt.event.*;
004
005 // Object that allows me to use height & width units
006
007 import java.awt.Dimension;
008
009 // Used to get todays date to use with Calendar
010
011 import java.util.Date;
012
013 // Editor for JSpinner that allows easy incrementing of dates
014
015 import javax.swing.SpinnerDateModel;
016
017 // Calendar provides methods that make it easy to work with dates
018
019 import java.util.Calendar;
020
021 public class Lesson26 extends JFrame{
022
023     JButton button1;
024     JSpinner spinner1, spinner2, spinner3, spinner4;
025     String outputString = "";
026
027     public static void main(String[] args){
028
029         new Lesson26();
030
031     }
032
033     public Lesson26(){
034
035         this.setSize(400,400);
036
037         this.setLocationRelativeTo(null);
038
039         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
040
041         this.setTitle("My Fifth Frame");
042
043         JPanel thePanel = new JPanel();
044
045         // Create a button
046
047         button1 = new JButton("Get Answer");
048
049         ListenForButton lForButton = new ListenForButton();
050
051         button1.addActionListener(lForButton);
052
053         thePanel.add(button1);
054
055         // Create a basic 1-9 number spinner
056
```

```
057         spinner1 = new JSpinner();
058 
059         thePanel.add(spinner1);
060 
061         // Create a spinner with initial number, starting number,
062         // max number, increment with each click
063 
064         spinner2 = new JSpinner(new SpinnerNumberModel(1, 1, 100,
    1));
065 
066         thePanel.add(spinner2);
067 
068         // Create a spinner using default values
069 
070         String[] weekDays = {"Mon", "Tues", "Weds", "Thurs",
    "Fri"};
071 
072         spinner3 = new JSpinner(new SpinnerListModel(weekDays));
073 
074         // Set the size for the spinner so that everything fits
075 
076         Dimension d = spinner3.getPreferredSize();
077         d.width = 80;
078         spinner3.setPreferredSize(d);
079 
080         thePanel.add(spinner3);
081 
082         // Get todays date
083 
084         Date todaysDate = new Date();
085 
086         // Create a date spinner & set default to today, no
    minimum, or max
087         // Increment the days on button presses
088         // Can also increment YEAR, MONTH, or DAY_OF_MONTH
089 
090         spinner4 = new JSpinner(new SpinnerDateModel(todaysDate,
    null, null,
091                 Calendar.DAY_OF_MONTH));
092 
093         // DateEditor is an editor that handles displaying &
    editing the dates
094 
095         JSpinner.DateEditor dateEditor = new
    JSpinner.DateEditor(spinner4, "dd/MM/yy");
096         spinner4.setEditor(dateEditor);
097 
098         thePanel.add(spinner4);
099 
100         /*
101         You can add a change listener with Spinners as well
102 
103         ListenForSpinner lForSpinner = new ListenForSpinner();
104 
105         Tell Java that you want to be alerted when an event
106         occurs on the spinner
107 
```

```
108         spinner4.addChangeListener(lForSpinner);
109         */
110
111         this.add(thePanel);
112
113         this.setVisible(true);
114
115     }
116
117     private class ListenForButton implements ActionListener{
118
119         public void actionPerformed(ActionEvent e){
120
121             if(e.getSource() == button1){
122
123                 outputString += "Spinner 1 Value: " +
    spinner1.getValue() + "\n";
124                 outputString += "Spinner 2 Value: " +
    spinner2.getValue() + "\n";
125                 outputString += "Spinner 3 Value: " +
    spinner3.getValue() + "\n";
126                 outputString += "Spinner 4 Value: " +
    spinner4.getValue() + "\n";
127
128                 JOptionPane.showMessageDialog(Lesson26.this,
    outputString, "Information", JOptionPane.INFORMATION_MESSAGE);
129
130                 outputString = "";
131
132             }
133         }
134     }
135
136 }
```

```
001 import javax.swing.*;
002
003 import java.awt.event.*;
004
005 import java.awt.Dimension;
006
007 // Enumerations are used to store related items together
008
009 import java.util.Enumeration;
010
011 import javax.swing.tree.*;
012
013 public class Lesson27 extends JFrame{
014
015     JButton button1;
016     String outputString = "";
017
018     // A Tree contains nodes that can contain other nodes
019
020     JTree theTree;
021
022     // If a node holds other nodes it is called a parent node
023     // The nodes inside of a parent node are children nodes
024     // Nodes on the same level are called siblings
025
026     DefaultMutableTreeNode documents, work, games, emails;
027
028     DefaultMutableTreeNode fileSystem = new DefaultMutableTreeNode("C Drive");
029
030     public static void main(String[] args){
031
032         new Lesson27();
033
034     }
035
036     public Lesson27(){
037
038         this.setSize(400,400);
039
040         this.setLocationRelativeTo(null);
041
042         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
043
044         this.setTitle("My Sixth Frame");
045
046         JPanel thePanel = new JPanel();
047
048         // Create a button
049
050         button1 = new JButton("Get Answer");
051
052         ListenForButton lForButton = new ListenForButton();
053
054         button1.addActionListener(lForButton);
055
056         thePanel.add(button1);
057
058         // Create the JTree by passing it the top tree node
059
060         theTree = new JTree(fileSystem);
061
062         // Makes sure only one item can be selected at a time
063         // By default you can make multiple selections
064
065         theTree.getSelectionModel().setSelectionMode(TreeSelectionModel.SINGLE_TREE_SELECTION);
066
067         // Only show 8 rows of the tree at a time
068
069         theTree.setVisibleRowCount(8);
070
071         // Add the items to the tree documents, work, games
072         // We first add the documents parent node
073
074         documents = addAFile("Docs", fileSystem);
075
```

```
076         // Now we add children nodes to the documents parent node
077
078         addAFile("Taxes.exl", documents);
079         emails = addAFile("Emails", documents);
080         addAFile("Story.txt", documents);
081         addAFile("Schedule.txt", documents);
082
083         // Add a child node to the email node
084
085         addAFile("CallBob.txt", emails);
086
087         // Create the work node and its children
088
089         work = addAFile("Work Applications", fileSystem);
090         addAFile("Spreadsheet.exe", work);
091         addAFile("Wordprocessor.exe", work);
092         addAFile("Presentation.exe", work);
093
094         // Create the games node and its children
095
096         games = addAFile("Games", fileSystem);
097         addAFile("SpaceInvaders.exe", games);
098         addAFile("PacMan.exe", games);
099
100         // Put the tree in a scroll component
101
102         JScrollPane scrollBox = new JScrollPane(theTree);
103
104         // Set the size for the JScrollPane so that everything fits
105
106         Dimension d = scrollBox.getPreferredSize();
107         d.width = 200;
108         scrollBox.setPreferredSize(d);
109
110         thePanel.add(scrollBox);
111
112         this.add(thePanel);
113
114         this.setVisible(true);
115
116     }
117
118     private DefaultMutableTreeNode addAFile(String fileName, DefaultMutableTreeNode
    parentFolder){
119
120         // Creates a new node for the tree
121         DefaultMutableTreeNode newFile = new DefaultMutableTreeNode(fileName);
122
123         // Add attaches a name to the node
124
125         parentFolder.add(newFile);
126
127         // return the new node
128
129         return newFile;
130
131     }
132
133     private class ListenForButton implements ActionListener{
134
135         public void actionPerformed(ActionEvent e){
136
137             if(e.getSource() == button1){
138
139                 // How to get the selected node
140                 // Returns the last selected node in the tree
141                 Object treeObject = theTree.getLastSelectedPathComponent();
142
143                 // Cast the Object into a DefaultMutableTreeNode
144
145                 DefaultMutableTreeNode theFile = (DefaultMutableTreeNode) treeObject;
146
147                 // Returns the object stored in this node and casts it to a string
148
149                 String treeNode = (String) theFile.getUserObject();
150
```

```
151                 outputString = "The Selected Node: " + treeNode + "\n";
152 
153                 // Get the number of children this node has
154 
155                 outputString += "Number of Children: " + theFile.getChildCount() + "\n";
156 
157                 // Get the number of siblings
158 
159                 outputString += "Number of Siblings: " + theFile.getSiblingCount() + "\n";
160 
161                 // Get the parent of this node
162 
163                 outputString += "The parent: " + theFile.getParent() + "\n";
164 
165                 // Get the next node
166 
167                 outputString += "Next Node: " + theFile.getNextNode() + "\n";
168 
169                 // Get the previous node
170 
171                 outputString += "Next Node: " + theFile.getPreviousNode() + "\n";
172 
173                 // Get the children for the node
174 
175                 outputString += "\nChildren of Node\n";
176 
177                 // children() returns an enumeration that contains all the children
178                 // This for loop will continue to run as long as there are more elements
179                 // nextElement() returns the next element in the list
180 
181                 for (Enumeration enumValue = theFile.children(); enumValue.hasMoreElements();
    ) {
182 
183                     outputString += enumValue.nextElement() + "\n";
184 
185                 }
186 
187                 // Get the path from the root
188 
189                 outputString += "\nPath From Root\n";
190 
191                 // getPath returns an array of TreeNodes
192 
193                 TreeNode[] pathNodes = theFile.getPath();
194 
195                 // Cycle through the TreeNodes
196 
197                 for(TreeNode indivNodes: pathNodes){
198                     outputString += indivNodes + "\n";
199                 }
200 
201                 JOptionPane.showMessageDialog(Lesson27.this, outputString, "Information",
    JOptionPane.INFORMATION_MESSAGE);
202 
203                 outputString = "";
204 
205             }
206         }
207     }
208 
209 }
```

```
001 import javax.swing.*;
002
003 import java.awt.event.*;
004
005 // New event listener that monitors changing values for components
006
007 import javax.swing.event.ChangeEvent;
008 import javax.swing.event.ChangeListener;
009
010 // Allows me to format the numbers
011
012 import java.text.NumberFormat;
013
014 // Allows me to edit borders on panels
015
016 import javax.swing.border.*;
017
018
019 public class Lesson22 extends JFrame{
020
021     JButton button1;
022     JLabel label1, label2, label3;
023     JTextField textField1, textField2;
024     JCheckBox dollarSign, commaSeparator;
025     JRadioButton addNums, subtractNums, multNums, divideNums;
026     JSlider howManyTimes;
027
028     double number1, number2, totalCalc;
029
030     public static void main(String[] args){
031
032         new Lesson22();
033
034     }
035
036     public Lesson22(){
037
038         // Define the size of the frame
039
040         this.setSize(400, 400);
041
042         // Opens the frame in the middle of the screen
043
044         this.setLocationRelativeTo(null);
045
046         // Define how the frame exits (Click the Close Button)
047
048         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
049
050         // Define the title for the frame
051
052         this.setTitle("My Third Frame");
053
054         // The JPanel contains all of the components for your frame
055
056         JPanel thePanel = new JPanel();
```

```
057
058         // ----------------------------------------------------
    -
059
060         // Create a button with Click Here on it
061
062         button1 = new JButton("Calculate");
063
064         // Create an instance of ListenForEvents to handle events
065
066         ListenForButton lForButton = new ListenForButton();
067
068         // Tell Java that you want to be alerted when an event
069         // occurs on the button
070
071         button1.addActionListener(lForButton);
072
073         thePanel.add(button1);
074
075         // How to add a label -------------------------
076
077         label1 = new JLabel("Number 1");
078
079         thePanel.add(label1);
080
081         // How to add a text field ---------------------
082
083         textField1 = new JTextField("", 5);
084
085         thePanel.add(textField1);
086
087         // How to add a label -------------------------
088
089         label2 = new JLabel("Number 2");
090
091         thePanel.add(label2);
092
093         // How to add a text field ---------------------
094
095         textField2 = new JTextField("", 5);
096
097         thePanel.add(textField2);
098
099         // How to add checkboxes ----------------------
100
101         dollarSign = new JCheckBox("Dollars");
102         commaSeparator = new JCheckBox("Commas");
103
104         thePanel.add(dollarSign);
105
106         // By putting true in here it is selected by default
107
108         thePanel.add(commaSeparator, true);
109
110         // Creates radio buttons with default labels
111
112         addNums = new JRadioButton("Add");
```

```
113         subtractNums = new JRadioButton("Subtract");
114         multNums = new JRadioButton("Multiply");
115         divideNums = new JRadioButton("Divide");
116
117         // Creates a group that will contain radio buttons
118         // You do this so that when 1 is selected the others
119         // are deselected
120
121         ButtonGroup operation = new ButtonGroup();
122
123         // Add radio buttons to the group
124
125         operation.add(addNums);
126         operation.add(subtractNums);
127         operation.add(multNums);
128         operation.add(divideNums);
129
130         // Create a new panel to hold radio buttons
131
132         JPanel operPanel = new JPanel();
133
134         // Surround radio button panel with a border
135         // You can define different types of borders
136         // createEtchedBorder, createLineBorder, createTitledBorder
137         // createLoweredBevelBorder, createRaisedBevelBorder
138
139         Border operBorder =
    BorderFactory.createTitledBorder("Operation");
140
141         // Set the border for the panel
142
143         operPanel.setBorder(operBorder);
144
145         // Add the radio buttons to the panel
146
147         operPanel.add(addNums);
148         operPanel.add(subtractNums);
149         operPanel.add(multNums);
150         operPanel.add(divideNums);
151
152         // Selects the add radio button by default
153
154         addNums.setSelected(true);
155
156         // Add the panel to the main panel
157         // You don't add the group
158
159         thePanel.add(operPanel);
160
161         // How to create a slider ----------------
162
163         label3 = new JLabel("Perform How Many Times?");
164
165         thePanel.add(label3);
166
167         // Creates a slider with a min value of 0 thru 99
168         // and an initial value of 1
```

```
169
170         howManyTimes = new JSlider(0, 99, 1);
171
172         // Defines the minimum space between ticks
173
174         howManyTimes.setMinorTickSpacing(1);
175
176         // Defines the minimum space between major ticks
177
178         howManyTimes.setMajorTickSpacing(10);
179
180         // Says to draw the ticks on the slider
181
182         howManyTimes.setPaintTicks(true);
183
184         // Says to draw the tick labels on the slider
185
186         howManyTimes.setPaintLabels(true);
187
188         // Create an instance of ListenForEvents to handle events
189
190         ListenForSlider lForSlider = new ListenForSlider();
191
192         // Tell Java that you want to be alerted when an event
193         // occurs on the slider
194
195         howManyTimes.addChangeListener(lForSlider);
196
197
198         thePanel.add(howManyTimes);
199
200         this.add(thePanel);
201
202         this.setVisible(true);
203
204         // Gives focus to the textfield
205
206         textField1.requestFocus();
207
208     }
209
210     private class ListenForButton implements ActionListener{
211
212         // This method is called when an event occurs
213
214         public void actionPerformed(ActionEvent e){
215
216             // Check if the source of the event was the button
217
218             if(e.getSource() == button1){
219
220                 // getText returns a String so you have to parse it
221                 // into a double in this situation
222
223                 try{
224                     number1 =
    Double.parseDouble(textField1.getText());
```

```
225                     number2 =
    Double.parseDouble(textField2.getText());
226                 }
227
228                 catch(NumberFormatException excep){
229
230                     // JOptionPane displays a popup on the screen
231                     // (parentComponent, message, title, error
    icon)
232                     // Error Icons: WARNING_MESSAGE,
    QUESTION_MESSAGE, PLAIN_MESSAGE
233
234                     JOptionPane.showMessageDialog(Lesson22.this,
    "Please Enter the Right Info", "Error", JOptionPane.ERROR_MESSAGE);
235                     System.exit(0); // Closes the Java app
236                 }
237
238
239                 if(addNums.isSelected()) { totalCalc =
    addNumbers(number1, number2, howManyTimes.getValue());
240
241                 } else if(subtractNums.isSelected()) { totalCalc =
    subtractNumbers(number1, number2, howManyTimes.getValue());
242
243                 } else if(multNums.isSelected()) { totalCalc =
    multiplyNumbers(number1, number2, howManyTimes.getValue());
244
245                 } else { totalCalc = divideNumbers(number1,
    number2, howManyTimes.getValue()); }
246
247                 // If the dollar is selected in the checkbox print
    the number as currency
248
249                 if(dollarSign.isSelected()) {
250
251                     // Defines that you want to format a number
    with $ and commas
252
253                     NumberFormat numFormat =
    NumberFormat.getCurrencyInstance();
254
255                     JOptionPane.showMessageDialog(Lesson22.this,
    numFormat.format(totalCalc), "Solution",
    JOptionPane.INFORMATION_MESSAGE);
256
257                 }
258
259                 // If the comma is selected in the checkbox print
    the number with commas
260
261                 else if(commaSeparator.isSelected()) {
262
263                     // Defines that you want to format a number
    with commas
264
265                     NumberFormat numFormat =
    NumberFormat.getNumberInstance();
```

```
266
267                     JOptionPane.showMessageDialog(Lesson22.this,
    numFormat.format(totalCalc), "Solution",
    JOptionPane.INFORMATION_MESSAGE);
268
269                 } else {
270
271                 JOptionPane.showMessageDialog(Lesson22.this,
    totalCalc, "Solution", JOptionPane.INFORMATION_MESSAGE);
272
273                 }
274
275             }
276
277         }
278
279     }
280
281     // Implements ActionListener so it can react to events on
    components
282
283         private class ListenForSlider implements ChangeListener{
284
285             @Override
286             public void stateChanged(ChangeEvent e) {
287
288                 // Check if the source of the event was the button
289
290                 if(e.getSource() == howManyTimes){
291
292                     label3.setText("Perform How Many Times? " +
    howManyTimes.getValue() );
293
294
295                 }
296
297             }
298
299
300
301         }
302
303         public static double addNumbers(double number1, double
    number2, int howMany){
304
305             double total = 0;
306
307             int i = 1;
308
309             while(i <= howMany){
310                 total = total + (number1 + number2);
311                 i++;
312             }
313
314             return total;
315
316         }
```

```
317
318         public static double subtractNumbers(double number1, double
    number2, int howMany){
319
320             double total = 0;
321
322             int i = 1;
323
324             while(i <= howMany){
325                 total = total + (number1 - number2);
326                 i++;
327             }
328
329             return total;
330
331         }
332
333         public static double multiplyNumbers(double number1, double
    number2, int howMany){
334
335             double total = 0;
336
337             int i = 1;
338
339             while(i <= howMany){
340                 total = total + (number1 * number2);
341                 i++;
342             }
343
344             return total;
345
346         }
347
348         public static double divideNumbers(double number1, double
    number2, int howMany){
349
350             double total = 0;
351
352             int i = 1;
353
354             while(i <= howMany){
355                 total = total + (number1 / number2);
356                 i++;
357             }
358
359             return total;
360
361         }
362
363 }
```

```
// In the last tutorial I coordinated threads using
// a timing method. Here I show you how to execute code based
// on a predefined time schedule and much more

// Used to schedule when certain events should be triggered
import java.util.concurrent.ScheduledThreadPoolExecutor;

// Used to tell Java what unit of time I want to use
import static java.util.concurrent.TimeUnit.*;

public class LessonEighteen{

	public static void main(String[] args){

		addThreadsToPool();

	}

	public static void addThreadsToPool(){

		// Allows you to schedule code execution at some time in the future
		// You can also have code execute repetitively based on a time
period
		// It must be big enough to hold all potential Threads

		ScheduledThreadPoolExecutor eventPool = new
ScheduledThreadPoolExecutor(5);

		// Adds a Thread to the pool. Tells that thread to start executing
		// after 0 seconds (immediately) and then execute every 2 seconds

		eventPool.scheduleAtFixedRate(new CheckSystemTime(), 0, 2,
SECONDS);

		eventPool.scheduleAtFixedRate(new PerformSystemCheck("Mail"),
5, 5, SECONDS);

		eventPool.scheduleAtFixedRate(new
PerformSystemCheck("Calendar"), 10,10, SECONDS);
```

```
        // Thread.activeCount returns the number of threads running

        System.out.println("Number of Threads: " +Thread.activeCount());

        // Quiz: Why does it say there are 4 threads when we expect 3?

        // Create an array of threads with enough spaces for all active
threads

        Thread[] listOfThreads = new Thread[Thread.activeCount()];

        // enumerate fills the Thread array with all active threads

        Thread.enumerate(listOfThreads);

        // Cycle through all the active threads and print out their names

        for(Thread i : listOfThreads){
                System.out.println(i.getName());
        }

        // Get priority of all the threads (Priority is equal to the thread
        // that created the new thread. Top priority is 10, lowest priority is 1

        for(Thread i : listOfThreads){
                System.out.println(i.getPriority());
        }

        // threadName.setPriority can be used to set the priority

        // This allows the above threads to run for approximately 20
seconds

        try{
                Thread.sleep(20000);
        }
        catch(InterruptedException e)
        {}

        // Shuts down all threads in the pool
```

```
                // eventPool.shutdown();

        }

}



import java.text.DateFormat;
import java.util.*;

public class CheckSystemTime implements Runnable{

        public void run(){

                Date rightNow;
                Locale currentLocale;
                DateFormat timeFormatter;
                String timeOutput;

                rightNow = new Date();
                currentLocale = new Locale("en");

                timeFormatter =
DateFormat.getTimeInstance(DateFormat.DEFAULT, currentLocale);
                timeOutput = timeFormatter.format(rightNow);

                System.out.println("Time: " + timeOutput);


        }

}



// You could also lock down a method and then unlock it
// when you are finished with it. This library does that
```

```
import java.util.concurrent.locks.ReentrantLock;

public class PerformSystemCheck implements Runnable{

	private String checkWhat;

	// Creates a lock for your method

	ReentrantLock lock = new ReentrantLock();

	public PerformSystemCheck(String checkWhat){

		this.checkWhat = checkWhat;

	}

	// By putting synchronized before a method, you make
	// sure only one thread at a time can execute it.
	// Synchronizing slows down Java, so it should only
	// be used when necessary.

	/* synchronized */ public void run(){

		// this locks down the method just like synchronized
		// You can't use synchronized and lock, that's why
		// synchronized is commented out above

		lock.lock();

		System.out.println("Checking " + this.checkWhat);

		// this unlocks the method just like synchronized

		lock.unlock();

	}

}
```

```java
// To import an external class you use import
// You can import whole libraries of classes like this import java.util.*;
import java.util.Scanner;

public class LessonTwo
{
    /* static means that only a class can call for this function to execute
    * Creates a new scanner object named userInput
    * You create the Scanner object by calling new and passing the Scanner constructor
    * the input stream to look at (System.in = keyboard input)
    */
    static Scanner userInput = new Scanner(System.in);

    public static void main(String[] args)
    {
        System.out.print("Your favorite number: "); // Same as println without a newline

        /* The if statement will only execute the code that lies between {} if the value inside () is true
         * userInput.hasNextDouble() returns true if the next value entered is a Double
         * There are similar methods for the other data types
         * hasNextInt() : Integer input
         * hasNextFloat() : Float input
         * There are others for Boolean, Byte, Long, and Short
         */

        if (userInput.hasNextInt())
        {

            int numberEntered = userInput.nextInt();
            /* userInput.nextDouble() receives user input and stores it in the variable numberEntered
             * You have to use a different method based on the type of input
             * nextInt() : Works for Integers
             * nextDouble() : Works for Doubles
             * nextFloat() : Works for Floats
             * nextLine() : Works for Strings
             * There are others for Boolean, Byte, Long, and Short
             * If the user enters a value of the wrong type the program crashes
             */

            System.out.println("You entered " + numberEntered);

            // Here I perform basic mathematics calculations
            int numEnteredTimes2 = numberEntered + numberEntered;
            System.out.println(numberEntered + " + " + numberEntered + " = " + numEnteredTimes2);

            int numEnteredMinus2 = numberEntered - 2;
```

```
047             System.out.println(numberEntered + " - 2 " + " = " +
    numEnteredMinus2);
048
049             int numEnteredTimesSelf = numberEntered *
    numberEntered;
050             System.out.println(numberEntered + " * " +
    numberEntered + " = " + numEnteredTimesSelf);
051
052             // Without the cast the value wouldn't consider
    fractions
053             double numEnteredDivide2 = (double)numberEntered / 2;
054             System.out.println(numberEntered + " / 2 " + " = " +
    numEnteredDivide2);
055
056             // % Modulus returns the remainder of a division
057             int numEnteredRemainder = numberEntered % 2;
058             System.out.println(numberEntered + " % 2 " + " = " +
    numEnteredRemainder);
059
060             // Shorthand way to add to 2 to a variable and assign
    the result to self
061             numberEntered += 2; // *=   /=  %=  Also work
062             numberEntered -= 2;
063
064             // Shorthand way to add 1 to a variable
065             numberEntered++;
066
067             // Shorthand way to subtract 1 from a variable
068             numberEntered--;
069
070             int numEnteredABS = Math.abs(numberEntered); // Returns
    the absolute value
071
072             // Returns the larger of the two arguments (They must
    be of the same type)
073             int whichIsBigger = Math.max(5, 7);
074
075             // Returns the smaller of the two arguments (They must
    be of the same type)
076             int whichIsSmaller = Math.min(5, 7);
077
078             // Returns the square root argument
079             double numSqrt = Math.sqrt(5.23);
080
081             // Rounds the number provided up
082             int numCeiling = (int) Math.ceil(5.23);
083             System.out.println("Ceiling: " + numCeiling);
084
085             // Rounds the number provided down
086             int numFloor = (int) Math.floor(5.23);
087             System.out.println("Floor: " + numFloor);
088
089             // Rounds the number based on the fraction
090             int numRound = (int) Math.round(5.23);
091             System.out.println("Rounded: " + numRound);
092
093             // Generates random numbers between 0 to 9
```

```
094             int randomNumber = (int) (Math.random() * 10);
095             System.out.println("A random number " + randomNumber);
096 
097         // If the above condition is false, the code following else
    is executed
098         } else {
099             System.out.println("Sorry you must enter an integer");
100         }
101 
102 
103     }
104 
105 }
```

```
public class LessonSeventeen{

	public static void main(String[] args){

		// Create a new Thread that executes the code in GetTime20

		Thread getTime = new GetTime20();

		// Create a new Thread created using the Runnable interface
		// Execute the code in run after 10 seconds

		Runnable getMail = new GetTheMail(10);

		Runnable getMailAgain = new GetTheMail(20);

		// Call for the code in the method run to execute

		getTime.start();

		new Thread(getMail).start();
		new Thread(getMailAgain).start();

	}

}



// By using threads you can execute multiple blocks
// of code at the same time. This program will output
// the current time and then at a specific time execute
// other code without stopping the time output

// Need this for Date and Locale classes
import java.util.*;

// Need this to format the dates
import java.text.DateFormat;

// By extending the Thread class you can run your code
```

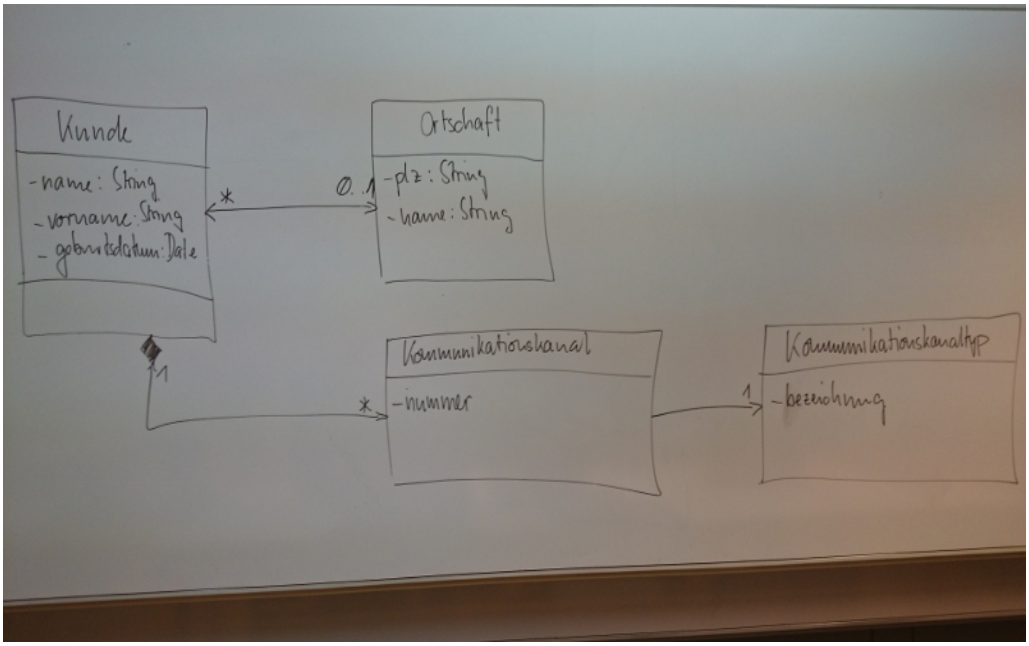
```
// concurrently with other threads
public class GetTime20 extends Thread{

	// All of the code that the thread executes must be
	// in the run method, or be in a method called for
	// from inside of the run method
	public void run(){

		// Creating fields that will contain date info
		Date rightNow;
		Locale currentLocale;
		DateFormat timeFormatter;
		DateFormat dateFormatter;
		String timeOutput;
		String dateOutput;

		// Output the current date and time 20 times
		for(int i = 1; i <= 20; i++){

			// A Date object contains date and time data
			rightNow = new Date();

			// Locale defines time formats depending on location
			currentLocale = new Locale("en", "US");

			// DateFormat allows you to define dates / times using
predefined
			// styles DEFAULT, SHORT, MEDIUM, LONG, or FULL
			// getTimeInstance only outputs time information
			timeFormatter =
DateFormat.getTimeInstance(DateFormat.DEFAULT, currentLocale);

			// getDateInstance only outputs time information
			dateFormatter =
DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);

			// Convert the time and date into Strings
			timeOutput = timeFormatter.format(rightNow);
			dateOutput = dateFormatter.format(rightNow);
```

```
                        System.out.println(timeOutput);
                        System.out.println(dateOutput);
                        System.out.println();

                        // You must wrap the sleep method in error handling
                        // code to catch the InterruptedException exception
                        // sleep pauses thread execution for 2 seconds below

                        try {
                                Thread.sleep(2000);
                        }
                        catch(InterruptedException e)
                        {}
                }

        }

}


// By using threads you can execute multiple blocks
// of code at the same time. This program will output
// the current time and then at a specific time execute
// other code without stopping the time output

// Need this for Date and Locale classes
import java.util.*;

// Need this to format the dates
import java.text.DateFormat;

// By extending the Thread class you can run your code
// concurrently with other threads
public class GetTime20 extends Thread{

        // All of the code that the thread executes must be
        // in the run method, or be in a method called for
        // from inside of the run method
        public void run(){
```

```
        // Creating fields that will contain date info
        Date rightNow;
        Locale currentLocale;
        DateFormat timeFormatter;
        DateFormat dateFormatter;
        String timeOutput;
        String dateOutput;

        // Output the current date and time 20 times
        for(int i = 1; i <= 20; i++){

            // A Date object contains date and time data
            rightNow = new Date();

            // Locale defines time formats depending on location
            currentLocale = new Locale("en", "US");

            // DateFormat allows you to define dates / times using
predefined
            // styles DEFAULT, SHORT, MEDIUM, LONG, or FULL
            // getTimeInstance only outputs time information
            timeFormatter =
DateFormat.getTimeInstance(DateFormat.DEFAULT, currentLocale);

            // getDateInstance only outputs time information
            dateFormatter =
DateFormat.getDateInstance(DateFormat.DEFAULT, currentLocale);

            // Convert the time and date into Strings
            timeOutput = timeFormatter.format(rightNow);
            dateOutput = dateFormatter.format(rightNow);

            System.out.println(timeOutput);
            System.out.println(dateOutput);
            System.out.println();

            // You must wrap the sleep method in error handling
            // code to catch the InterruptedException exception
            // sleep pauses thread execution for 2 seconds below
```

```
                try {
                        Thread.sleep(2000);
                }
                catch(InterruptedException e)
                {}
        }

    }

}


// You can use the Runnable interface instead of
// wasting your 1 class extension.

public class GetTheMail implements Runnable {

    // Stores the number of seconds before the code
    // will be executed
    private int startTime;

    // Constructor that sets the wait time for each
    // new Thread
    public GetTheMail(int startTime){
        this.startTime = startTime;
    }

    // All of the code that the thread executes must be
    // in the run method, or be in a method called for
    // from inside of the run method
    public void run(){

        try
        {
            // Don't execute until 10 seconds has passed if
            // startTime equals 10
            Thread.sleep(startTime * 1000);
        }

        catch(InterruptedException e)
```

```
                {}
                System.out.println("Checking for Mail");
        }

}
```




```
import java.util.Date;
import java.util.List;
import java.util.ArrayList;

public class Kunde {

    private String name;
    private String vorname;
    private Date geburtsdatum;
    private Ortschaft ortschaft;
    private List<Kommunikationskanal> kommunikationskanaele = new
ArrayList<>();

    public String getName() { return name; }
    public String getVorname() { return vorname; }
    public Date getGeburtsdatum() { return geburtsdatum; }
    public Ortschaft getOrtschaft() { return ortschaft; }
    // Um die Konsistenz der Liste zu wahren und
sicherzustellen, dass keine ungewollte Manipulation der Liste
möglich ist,
// wird im Getter mit einer Listenkopie gearbeitet.
    public List<Kommunikationskanal> getKommunikationskanaele() {
return new ArrayList<Kommunikationskanal>(kommunikationskanaele);
}
```

```
public void setName(String name) { this.name = name; }
	public void setVorname(String vorname) { this.vorname =
vorname; }
	public void setGeburtsdatum(Date geburtsdatum)
{ this.geburtsdatum = geburtsdatum; }
	// Kein Setter für Kommunikationskanaele

	public void addKommunikationskanal(String nummer,
Kommunikationskanaltyp kommunikationskanaltyp) {
		kommunikationskanaele.add(new
Kommunikationskanal(nummer, kommunikationskanaltyp));
	}

	public void removeKommunikationskanal(Kommunikationskanal
kommunikationskanal) {
		kommunikationskanaele.remove(kommunikationskanal);
	}

/*  Variante mit Ortschaftskardinalität 0..1

	public void setOrtschaft(Ortschaft o) {
		if (ortschaft != null)
			ortschaft.removeKunde(this);
		ortschaft = o;

		if (ortschaft != null)
			ortschaft.addKunde(this);	}	*/

	// Variante mit Ortschaftskardinalität 1
public void setOrtschaft(Ortschaft o) throws Exception {
	if (o == null)
	throw new Exception("Ortschaft muss zwingend definiert werden");
		if (ortschaft != null)
			ortschaft.removeKunde(this);
		ortschaft = o;
		ortschaft.addKunde(this);  }	}
```

```
import java.util.List;
import java.util.ArrayList;

public class Ortschaft {

    private String plz;
    private String name;
    private List<Kunde> kunden = new ArrayList<>();

    public String getPlz() { return plz; }
    public String getName() { return name; }
    public List<Kunde> getKunden() { return kunden; }

    public void setPlz(String plz) { this.plz = plz; }
    public void setName(String name) { this.name = name; }
    public void setKunden(List<Kunde> kunden) { this.kunden =
kunden; }

    public void addKunde(Kunde kunde) {
        kunden.add(kunde);
    }

    public void removeKunde(Kunde kunde) {
        kunden.remove(kunde);
    }
}



public class Kommunikationskanal {

    private String nummer;
    private Kommunikationskanaltyp kommunikationskanaltyp;

    public Kommunikationskanal(String nummer,
Kommunikationskanaltyp kommunikationskanaltyp) {
        this.nummer = nummer;
        this.kommunikationskanaltyp = kommunikationskanaltyp;
```

```
	}

	public String getNummer() { return nummer; }
	public Kommunikationskanaltyp getKommunikationskanaltyp()
{ return kommunikationskanaltyp; }

	public void setNummer(String nummer) { this.nummer = nummer; }
	public void setKommunikationskanaltyp(Kommunikationskanaltyp
kommunikationskanaltyp) { this.kommunikationskanaltyp =
kommunikationskanaltyp; }
}
```

public class Kommunikationskanaltyp {

```
	private String bezeichnung;

	public Kommunikationskanaltyp(String bezeichnung) {
		this.bezeichnung = bezeichnung;
	}

	public String getBezeichnung() { return bezeichnung; }

	public void setBezeichnung(String bezeichnung) {
 this.bezeichnung = bezeichnung;
}
}
```

```
// The Model performs all the calculations needed
// and that is it. It doesn't know the View
// exists

public class CalculatorModel {

	// Holds the value of the sum of the numbers
	// entered in the view

	private int calculationValue;

public void addTwoNumbers(int firstNumber, int secondNumber){

    calculationValue = firstNumber + secondNumber;

	}

	public int getCalculationValue(){

		return calculationValue;

	}

}


// This is the View
// Its only job is to display what the user sees
// It performs no calculations, but instead passes
// information entered by the user to whomever needs
// it.

import java.awt.event.ActionListener;
```

```
import javax.swing.*;

public class CalculatorView extends JFrame{

	private JTextField firstNumber  = new JTextField(10);
	private JLabel additionLabel = new JLabel("+");
	private JTextField secondNumber = new JTextField(10);
	private JButton calculateButton = new JButton("Calculate");
	private JTextField calcSolution = new JTextField(10);

	CalculatorView(){

		// Sets up the view and adds the components

		JPanel calcPanel = new JPanel();

		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setSize(600, 200);

		calcPanel.add(firstNumber);
		calcPanel.add(additionLabel);
		calcPanel.add(secondNumber);
		calcPanel.add(calculateButton);
		calcPanel.add(calcSolution);

		this.add(calcPanel);

		// End of setting up the components --------

	}

	public int getFirstNumber(){
```

```
        return Integer.parseInt(firstNumber.getText());
    }
    public int getSecondNumber(){
        return Integer.parseInt(secondNumber.getText());
    }
    public int getCalcSolution(){
        return Integer.parseInt(calcSolution.getText());
    }
    public void setCalcSolution(int solution){
        calcSolution.setText(Integer.toString(solution));
    }
    // If the calculateButton is clicked execute a method
    // in the Controller named actionPerformed
    void addCalculateListener(ActionListener listenForCalcButton){
        calculateButton.addActionListener(listenForCalcButton);
    }
    // Open a popup that contains the error message passed
```

```
	void displayErrorMessage(String errorMessage){

		JOptionPane.showMessageDialog(this, errorMessage);

	}

}



import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

// The Controller coordinates interactions
// between the View and Model

public class CalculatorController {

	private CalculatorView theView;
	private CalculatorModel theModel;

	public CalculatorController(CalculatorView theView,
CalculatorModel theModel) {
		this.theView = theView;
		this.theModel = theModel;

		// Tell the View that when ever the calculate button
		// is clicked to execute the actionPerformed method
		// in the CalculateListener inner class

	this.theView.addCalculateListener(new CalculateListener());
	}
```

```
class CalculateListener implements ActionListener{

	public void actionPerformed(ActionEvent e) {

		int firstNumber, secondNumber = 0;

		// Surround interactions with the view with
		// a try block in case numbers weren't
		// properly entered

		try{

			firstNumber = theView.getFirstNumber();
		secondNumber = theView.getSecondNumber();

theModel.addTwoNumbers(firstNumber, secondNumber);


theView.setCalcSolution(theModel.getCalculationValue());

		}

		catch(NumberFormatException ex){

			System.out.println(ex);

theView.displayErrorMessage("You Need to Enter 2 Integers");

		}

	}
```

```
        }

}
public class MVCCalculator {

    public static void main(String[] args) {

        CalculatorView theView = new CalculatorView();

        CalculatorModel theModel = new CalculatorModel();
CalculatorController theController = new
CalculatorController(theView,theModel);

      theView.setVisible(true);

    }
}
```

```
001 import java.util.Arrays;
002
003 import org.apache.commons.lang3.ArrayUtils;
004
005 // Basic class definition
006 // public means this class can be used by other classes
007 // Class names should begin with a capital letter
008 // A file can't contain two public classes. It can contain classes
    that are not public
009 // If you place class files in the same folder the java compiler
    will be able to find them
010
011 public class MonsterTwo{
012     // Creates a multidimensional array of chars
013     static char[][] battleBoard = new char[10][10];
014
015     // This static method builds an empty battle board
016     public static void buildBattleBoard(){
017
018         // Cycles through the array and gives a default value of *
    to everything
019
020         for(char[] row : battleBoard)
021             Arrays.fill(row, '*');
022     }
023
024     // Redraws the board
025     public static void redrawBoard()
026     {
027
028         int k = 1;
029         while(k <= 30){ System.out.print('-'); k++; }
030         System.out.println();
031
032         for(int i = 0; i < battleBoard.length; i++)
033         {
034             for(int j = 0; j < battleBoard[i].length; j++)
035             {
036                 System.out.print("|" + battleBoard[i][j] + "|");
037             }
038             System.out.println();
039
040         }
041         k = 1;
042         while(k <= 30){ System.out.print('-'); k++; }
043         System.out.println();
044     }
045
046     // Class Variables or Fields
047     // You declare constants with final
048
049     public final String TOMBSTONE = "Here Lies a Dead monster";
050
051     // private fields are not visible outside of the class
052     private int health = 500;
053     private int attack = 20;
```

```
054     private int movement = 2;
055 
056     // Monitors whether the monster is alive or dead
057     private boolean alive = true;
058 
059     // public variables are visible outside of the class
060     // You should have as few as possible public fields
061     public String name = "Big Monster";
062     public int xPosition = 0;
063     public int yPosition = 0;
064     public char nameChar1 = 'B';
065     public static int numOfMonsters = 0;
066 
067 
068     // Class Methods
069     // Accessor Methods are used to get and set the values of
    private fields
070 
071     public int getAttack()
072     {
073         return attack;
074     }
075 
076     public int getMovement()
077     {
078         return movement;
079     }
080 
081     public int getHealth()
082     {
083         return health;
084     }
085 
086     public boolean getAlive()
087     {
088         return alive;
089     }
090 
091     // You can create multiple versions using the same method name
092     // Now setHealth can except an attack that contains decimals
093     // When overloading a method you can't just change the return
    type
094     // Focus on creating methods that except different parameters
095 
096     public void setHealth(int decreaseHealth)
097     {
098         health = health - decreaseHealth;
099         if (health < 0)
100         {
101             alive = false;
102         }
103     }
104 
105     public void setHealth(double decreaseHealth)
106     {
107         int intDecreaseHealth = (int) decreaseHealth;
108         health = health - intDecreaseHealth;
```

```
109         if (health < 0)
110         {
111             alive = false;
112         }
113     }
114 
115     public void moveMonster(MonsterTwo[] monster, int
    arrayItemIndex)
116     {
117         // isSpaceOpen will be used to track whether the space the
118         // monster plans to move into is occupied
119         boolean isSpaceOpen = true;
120 
121         // Define the maximum x and y for the battle board
122         // It's 1 less because the array index starts at 0
123         int maxXBoardSpace = battleBoard.length - 1;
124         int maxYBoardSpace = battleBoard[0].length - 1;
125 
126 
127         // while loop used to make sure I don't move a monster
128         // into an occupied space
129         while(isSpaceOpen)
130         {
131 
132         // Randomly generate move direction N, S, E, or W
133         int randMoveDirection = (int) (Math.random() * 4);
134 
135         // Randomly generate move distance based on max move
    distance
136         int randMoveDistance = (int) (Math.random() *
    (this.getMovement() + 1));
137 
138         // Prints move distance and move direction
139         System.out.println(randMoveDistance + " " +
    randMoveDirection);
140 
141         // Erase monsters character on the board by replacing it
    with a *
142 
143         battleBoard[this.yPosition][this.xPosition] = '*';
144 
145 
146 
147         if(randMoveDirection == 0)
148         {
149             // Find new xPosition & yPosition based on the current
    position on the board
150             // If statements won't allow monster to move off the
    board
151 
152             if((this.yPosition - randMoveDistance) < 0)
153             {
154                 this.yPosition = 0;
155             } else {
156                 this.yPosition = this.yPosition - randMoveDistance;
157             }
158         } else if (randMoveDirection == 1) {
```

```
159             if((this.xPosition + randMoveDistance) > maxXBoardSpace)
160             {
161                 this.xPosition = maxXBoardSpace;
162             } else {
163                 this.xPosition = this.xPosition + randMoveDistance;
164             }
165         } else if (randMoveDirection == 2) {
166             if((this.yPosition + randMoveDistance) > maxYBoardSpace)
167             {
168                 this.yPosition = maxYBoardSpace;
169             } else {
170                 this.yPosition = this.yPosition + randMoveDistance;
171             }
172         } else {
173             if((this.xPosition - randMoveDistance) < 0)
174             {
175                 this.xPosition = 0;
176             } else {
177                 this.xPosition = this.xPosition - randMoveDistance;
178             }
179         }
180 
181         // monster.length returns the number of items in the array monster
182         for (int i = 0; i < monster.length; i++)
183         {
184             // if statement skips checking the same monster position against itself
185 
186             if (i == arrayItemIndex)
187             {
188                 continue;
189             }
190 
191             // onMySpace receives the monster array, index for the object I'm
192             // checking currently, and the index for the monster sent to
193             // this function
194 
195             if(onMySpace(monster, i, arrayItemIndex))
196             {
197                 // If a monster tries to move to an occupied space the
198                 // while loop repeats after I break out of the for loop
199 
200                 isSpaceOpen = true;
201                 break;
202             } else {
203                 // There was no monster in the space so end the while loop
204                 isSpaceOpen = false;
205 
206             }
```

```
207
208         }
209
210         } // End of while loop
211
212         // Set the value in the array to the first letter of the
    monster
213         battleBoard[this.yPosition][this.xPosition] =
    this.nameChar1;
214     }
215
216     // Checks if one monster is trying to move into the same x/y
    position as
217     // another monster
218     public boolean onMySpace(MonsterTwo[] monster, int indexToChk1,
    int indexToChk2)
219     {
220         // Checks if the 2 monsters have the same x/y position
221         if((monster[indexToChk1].xPosition)==
    (monster[indexToChk2].xPosition)&&
    (monster[indexToChk1].yPosition)==(monster[indexToChk2].yPosition))
222         {
223             // If they are equal return true so a new x/y position
    is calculated
224
225             return true;
226
227         } else {
228
229             // If false I know the x/y position isn't occupied
230             return false;
231         }
232     }
233
234
235     /* The Constructor
236      * Code that is executed when an object is created from this
    class definition
237      * The method name is the same as the class
238      * The constructor is only executed once per object
239      * The constructor can't return a value
240      */
241
242     public MonsterTwo(int health, int attack, int movement, String
    name)
243     {
244         this.health = health;
245         this.attack = attack;
246         this.movement = movement;
247         this.name = name;
248         /* If the attributes had the same names as the class
    health, attack, movement
249          * You could refer to the objects fields by proceeding them
    with this
250          * this.health = health;
251          * this.attack = attack;
252          * objectFieldName = attributeFieldName;
```

```
253          */
254
255         // Define the maximum x and y for the battle board
256         // It's 1 less because the array index starts at 0
257         int maxXBoardSpace = battleBoard.length - 1;
258         int maxYBoardSpace = battleBoard[0].length - 1;
259
260         // The random starting position for a monster
261         int randNumX, randNumY;
262
263         // We use a do loop because we always want to define a
    start
264         // position for each monster
265
266         do {
267         // Calculate start position based on max board space
268         randNumX = (int) (Math.random() * maxXBoardSpace);
269         randNumY = (int) (Math.random() * maxYBoardSpace);
270         } while(battleBoard[randNumY][randNumX] != '*');
271         // Only allow monster to start on a space with a * on it
272
273         // Assign x and y position to the object that called this
    method
274         this.xPosition = randNumX;
275         this.yPosition = randNumY;
276
277         // Assign character in the array based on the first initial
278         // of the monsters name charAt(0) returns first letter of
    name
279         this.nameChar1 = this.name.charAt(0);
280
281         // Put first character of monster in the array
282         battleBoard[this.yPosition][this.xPosition] =
    this.nameChar1;
283
284         numOfMonsters++; // Adds 1 to the number of monsters on the
    board
285     }
286
287     // You can overload constructors like any other method
288     // The following constructor is the one provided by default if
    you don't create any other constructors
289     // Default Constructor
290
291     public MonsterTwo()
292     {
293         numOfMonsters++; // Adds 1 to the number of monsters on the
    board
294     }
295
296 public static void main(String[] args)
297 {
298
299
300
301 }
302
```

```
303 }
```

```
001 // Layout used by the JPanel
002 import java.awt.BorderLayout;
003
004 // Define color of shapes
005 import java.awt.Color;
006
007 // Allows me to draw and render shapes on components
008 import java.awt.Graphics;
009 import java.awt.Graphics2D;
010 import java.awt.RenderingHints;
011
012 import java.awt.event.KeyEvent;
013 import java.awt.event.KeyListener;
014 import java.awt.geom.AffineTransform;
015
016 // Will hold all of my Rock objects
017 import java.util.ArrayList;
018
019 // Runs commands after a given delay
020 import java.util.concurrent.ScheduledThreadPoolExecutor;
021
022 // Defines time units. In this case TimeUnit.MILLISECONDS
023 import java.util.concurrent.TimeUnit;
024
025 import javax.swing.JComponent;
026 import javax.swing.JFrame;
027
028 public class GameBoard extends JFrame{
029
030     // Height and width of the game board
031
032     public static int boardWidth = 1000;
033     public static int boardHeight = 800;
034
035     // Used to check if a key is being held down
036
037     public static boolean keyHeld = false;
038
039     // Gets the keycode for the key being held down
040
041     public static int keyHeldCode;
042
043     public static void main(String [] args)
044     {
045             new GameBoard();
046
047     }
048
049     public GameBoard()
050     {
051         // Define the defaults for the JFrame
052
053         this.setSize(boardWidth, boardHeight);
054         this.setTitle("Java Asteroids");
055         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
056
057         // Handles executing code based on keys being pressed
058
059         addKeyListener(new KeyListener() {
060
061             @Override
062             public void keyTyped(KeyEvent e) {
063                 // TODO Auto-generated method stub
064
065             }
066
067             @Override
068             public void keyPressed(KeyEvent e) {
069                 if (e.getKeyCode()==87)
070                 {
071                     System.out.println("Forward");
072
073                     keyHeldCode = e.getKeyCode();
```

```
074                     keyHeld = true;
075
076                 } else if (e.getKeyCode()==83){
077                     System.out.println("Backward");
078
079                     keyHeldCode = e.getKeyCode();
080                     keyHeld = true;
081
082                 }
083
084                 // Id the d key is pressed set keyHeld as if it
085                 // was being held down. This will cause the ship to
086                 // constantly rotate. keyHeldCode stores the keyCode for d
087
088                 else if (e.getKeyCode()==68){
089                     System.out.println("Rotate Right");
090
091                     keyHeldCode = e.getKeyCode();
092                     keyHeld = true;
093
094                 }
095
096                 // Same thing is done here as was done with the last
097                 // 65 is the keyCode for a
098
099                 else if (e.getKeyCode()==65){
100                     System.out.println("Rotate Left");
101
102                     keyHeldCode = e.getKeyCode();
103                     keyHeld = true;
104                 }
105
106             }
107
108             // When the key is released this informs the code that
109             // the key isn't being held down
110
111             public void keyReleased(KeyEvent e) {
112
113                 keyHeld = false;
114
115             }
116
117         });
118
119
120         GameDrawingPanel2 gamePanel = new GameDrawingPanel2();
121
122      // Make the drawing area take up the rest of the frame
123
124         this.add(gamePanel, BorderLayout.CENTER);
125
126         // Used to execute code after a given delay
127         // The attribute is corePoolSize - the number of threads to keep in
128         // the pool, even if they are idle
129
130         ScheduledThreadPoolExecutor executor = new ScheduledThreadPoolExecutor(5);
131
132         // Method to execute, initial delay, subsequent delay, time unit
133
134         executor.scheduleAtFixedRate(new RepaintTheBoard2(this), 0L, 20L,
    TimeUnit.MILLISECONDS);
135
136         // Show the frame
137
138         this.setVisible(true);
139     }
140
141 }
142
143
144 // Class implements the runnable interface
145 // By creating this thread we can continually redraw the screen
146 // while other code continues to execute
```

```
147
148 class RepaintTheBoard2 implements Runnable{
149
150     GameBoard theBoard;
151
152     public RepaintTheBoard2(GameBoard theBoard){
153         this.theBoard = theBoard;
154     }
155
156     @Override
157     public void run() {
158
159         // Redraws the game board
160
161         theBoard.repaint();
162
163     }
164
165 }
166
167 @SuppressWarnings("serial")
168
169 // GameDrawingPanel is what we are drawing on
170
171 class GameDrawingPanel2 extends JComponent {
172
173     // Holds every Rock I create
174
175     public static ArrayList<Rock> rocks = new ArrayList<Rock>();
176
177     // Get the original x & y points for the polygon
178
179     int[] polyXArray = Rock.sPolyXArray;
180     int[] polyYArray = Rock.sPolyYArray;
181
182     // Create a SpaceShip
183     SpaceShip theShip = new SpaceShip();
184
185     // Gets the game board height and weight
186
187     int width = GameBoard.boardWidth;
188     int height = GameBoard.boardHeight;
189
190     // Creates 50 Rock objects and stores them in the ArrayList
191     // Suppress warnings when I clone the rocks array
192
193     public GameDrawingPanel2() {
194
195         for(int i = 0; i < 10; i++){
196
197             // Find a random x & y starting point
198             // The -40 part is on there to keep the Rock on the screen
199
200             int randomStartXPos = (int) (Math.random() * (GameBoard.boardWidth - 40) + 1);
201             int randomStartYPos = (int) (Math.random() * (GameBoard.boardHeight - 40) +
    1);
202
203             // Add the Rock object to the ArrayList based on the attributes sent
204
205             rocks.add(new Rock(Rock.getpolyXArray(randomStartXPos),
    Rock.getpolyYArray(randomStartYPos), 13, randomStartXPos, randomStartYPos));
206
207             Rock.rocks = rocks; // NEW
208
209         }
210
211     }
212
213     public void paint(Graphics g) {
214
215         // Allows me to make many settings changes in regards to graphics
216
217         Graphics2D graphicSettings = (Graphics2D)g;
218
```

```
219         AffineTransform identity = new AffineTransform();
220 
221         // Draw a black background that is as big as the game board
222 
223         graphicSettings.setColor(Color.BLACK);
224         graphicSettings.fillRect(0, 0, getWidth(), getHeight());
225 
226         // Set rendering rules
227 
228         graphicSettings.setRenderingHint( RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
229 
230         // Set the drawing color to white
231 
232         graphicSettings.setPaint( Color.WHITE );
233 
234         // Cycle through all of the Rock objects
235 
236         for(Rock rock : rocks){
237 
238             // Move the Rock polygon
239 
240             rock.move();
241 
242             // Stroke the polygon Rock on the screen
243 
244             graphicSettings.draw(rock);
245 
246         }
247 
248         // Handles spinning the ship in the clockwise direction when the D key
249         // is pressed and held
250 
251         if(GameBoard.keyHeld == true && GameBoard.keyHeldCode == 68){
252 
253             theShip.increaseRotationAngle();
254 
255             System.out.println("Ship Angle: " + theShip.getRotationAngle());
256 
257         } else
258 
259         // Continues to rotate the ship counter clockwise if the A key is held
260 
261         if(GameBoard.keyHeld == true && GameBoard.keyHeldCode == 65){
262 
263             theShip.decreaseRotationAngle();
264 
265             System.out.println("Ship Angle: " + theShip.getRotationAngle());
266 
267         } else
268 
269         if (GameBoard.keyHeld == true && GameBoard.keyHeldCode == 87){
270 
271             // Set movement angle to the current rotation angle
272             // This is done so that the ship rotation can be set by the A & D keys
273             // but when the throttle is hit the ship knows what direction to go
274 
275             theShip.setMovingAngle(theShip.getRotationAngle());
276 
277             // Changes the values of x & y based on the angle of the ship. This way it
    knows if it        should increase or decrease x & y. By putting .01 in here we can slowly
    increase the velocity
278 
279             theShip.increaseXVelocity(theShip.shipXMoveAngle(theShip.getMovingAngle())*0.1);
280             theShip.increaseYVelocity(theShip.shipYMoveAngle(theShip.getMovingAngle())*0.1);
281 
282         }
283 
284         theShip.move();
285 
286         // Sets the origin on the screen so rotation occurs properly
287 
288         graphicSettings.setTransform(identity);
289 
```

```
290         // Moves the ship to the center of the screen
291
292         graphicSettings.translate(theShip.getXCenter(),theShip.getYCenter());
293
294         // Rotates the ship
295
296         graphicSettings.rotate(Math.toRadians(theShip.getRotationAngle()));
297
298         graphicSettings.draw(theShip);
299
300
301     }
302
303 }
```

```
001 Features of SpaceShip
002
003 1. Move the ship around the screen
004 2. Rotate the ship clockwise when D is pressed
005 3. Rotate the ship counter clockwise when A is pressed
006 4. I want the ship to rotate in any direction 360 degrees
007 5. Increase velocity slowly when W is pressed
008 6. If the ship goes off the board have it appear on the opposite
    side of the board
009 7. If the ship hits something it explodes
010 8. I want to be able to rotate the ship while it continues on its
    current path
011 9. Use encapsulation so the application is more flexible
012
013 Use Case Diagram
014
015 The Player
016
017 1. Starts the game
018 2. Moves the ship
019     a. Rotates the ship
020     b. Increases velocity on ship
021 3. If ship crashes start over
022
023 Risks / Problem Code
024
025 1. 360 degree rotation means we have to use doubles
026 2. Polygons are built using int arrays
027 3. Floating point calculations lose precision
028 4. How do we slowly increase velocity
029 5. How can they rotate the ship while maintaining the current
    course
030 6. How do we move the ship when it goes off the board
031
032 We've already solved collision problems, so put that on the back
    burner
033
034 AffineTransform allows us to:
035
036 1. Use doubles for rotation
037 2. Let the polygon be an int[] but have the center be a double
038 3. When the ship goes off the board just change the center point
039
040 Solves: 1, 2, 3, 6
041
042 Create velocity methods that increase and decrease velocity
043 Solves: 4
044
045 Have a rotation angle for current movement angle and one for ship
    rotation
046 Solves: 5
047
048 Scenario
049
050 1. Draw polygon in the center of the screen with points surrounding
    the center points.
```

```
051
052 a. GameBoard width and height
053 b. Center points for polygon x & y
054 c. Polygon x & y int array
055 d. Might need upper left hand x & y?
056
057 2. Player increase the ship velocity
058
059 a. x & y velocity
060
061 3. Player rotates the ship after acceleration
062
063 a. Rotation angle
064 b. Movement angle
065 c. Calculate movement angle on the fly
066
067 4. Player hits a rock and blows up
068
069 a. GetBounds method
070
071 SpaceShip : UML Class Diagram
072
073 gBWidth : int
074 gBHeight : int
075 centerX : double
076 centerY : double
077 polyXArray : int[]
078 polyYArray : int[]
079 shipWidth : int
080 shipHeight : int
081 uLeftXPos : double
082 uLeftYPos : double
083 xVelocity : double
084 yVelocity : double
085 rotationAngle : double
086 movingAngle : double
087
088 SpaceShip( int[], int[], int)
089
090 getXCenter() : double
091 getYCenter() : double
092 setXCenter(double) : void
093 setYCenter(double) : void
094 increaseXPos(double) : void
095 increaseYPos(double) : void
096 getuLeftXPos() : double
097 getuLeftYPos() : double
098 setuLeftXPos(double) : void
099 setuLeftYPos(double) : void
100 getShipWidth() : int
101 getShipHeight() : int
102 getXVelocity() : double
103 getYVelocity() : double
104 setXVelocity(double) : void
105 setYVelocity(double) : void
106 increaseXVelocity(double) : void
107 increaseYVelocity(double) : void
```

```
decreaseXVelocity(double) : void
decreaseYVelocity(double) : void
setMovingAngle(double) : void
getMovingAngle() : double
increaseMovingAngle(double) : void
shipXMoveAngle(double) : double
shipYMoveAngle(double) : double
getRotationAngle() : double
increaseRotationAngle() : void
decreaseRotationAngle() : void
getBounds() : Rectangle
move() : void
```

Höhere Fachschule für Technik Informatik III LE 03
des Kantons Solothurn Umsetzung Klassendiagramm

Umsetzung des Klassendiagramms in Java *Einleitung*

Sie kennen nun praktisch alle Notationselemente des Klassendiagramms der UML. Ausserdem haben wir die Konzepte der Objektorientierung und die Vorgehensweise zur Erarbeitung der Artefakte erarbeitet und vertieft. Mit etwas Übung sind Sie damit in der Lage, aus einer Problemstellung die statischen Teile des Systems zu modellieren.
Das ist alles gut und recht, nur ist mit einem Klassendiagramm alleine noch keine Applikation erstellt. Wir müssen uns also noch damit befassen, wie ein Klassendiagramm mit Hilfe der Programmiersprache Java umgesetzt werden kann. Einen Teil davon haben wir bereits in Informatik I behandelt. Wir haben einzelne Klassen bereits in Code umgesetzt. Was daraus entsteht ist nur gerade ein Gerüst für die Umsetzung – eben die statischen Teile der Klasse. Wir wissen aber nicht, wie Assoziationen, Aggregationen und Kompositionen umgesetzt werden. Ausser kann die Kardinalität auch immer wieder für Kopfzerberchen sorgen.
Wir wollen zuerst das Bekannte in diesem Zusammenhang kurz repetieren, um einen optimalen Einstieg in das Thema zu erhalten.

Repetition Umsetzung einer einzelnen Klasse

Die unten abgebildete Klasse beschreibt einen Bus zum Transport von Passagieren. Sie enthält alle Artefakte, die
in einer Java Klasse auftreten können.

Bus
+PASSAGIERBEZEICHNUNG: String = "PAX" -totalPassagiere: int = 0 -totalUmsatz: double = 0.0 -linie: String -ticketPreis: double -umsatz: double -anzahlPassagiere: int
+Bus(linie: String, ticketPreis: double) +getTotalPassagiere(): int

+getTotalUmsatz(): double
+getLinie(): String
+getTicketPreis(): double
+zusteigen(anzahlPassagiere: int): void
+aussteigen(anzahlPassagiere : int): void
+getUmsatz(): double

Bei **PASSAGIERBEZEICHNUNG** handelt es sich um eine Konstante. Konstanten können nur beim Laden der Klasse durch den Classloader initialisiert werden. Anschliessend kann der Wert nicht mehr abgeändert werden. Konstanten sind gleichzeitig Klassenattribute, haben also für alle Objekte den gleichen Wert. Die Sichtbarkeit dieser Konstante ist **public**, kann also über den Klassennamen angesprochen werden. Das macht aus der Sicht der Objektorientierung durchaus Sinn, da der Wert nicht verändert werden kann und damit nicht noch zusätzlich durch Kapselung geschützt werden muss. Für die Umsetzung der Unveränderbarkeit verwendet Java das Schlüsselwort **final** bei der Deklaration des Attributs. Bei der Deklaration muss das Attribut gleichzeitig initialisiert werden. Der Initialwert sollte im Klassendiagramm immer erwähnt sein. Da eine Konstante der Klasse und nicht einer Instanz zugeordnet ist, findet ausserdem das Schlüsselwort **static** Anwendung. Unsere Konstante wird also wie folgt definiert:
public static final String PASSAGIERBEZEICHNUNG = "PAX";
Die Attribute **totalPassagiere** und **totalUmsatz** sind Klassenattribute (**static**). Sie sind mit der Sichtbarkeit **private** modelliert. Die Attribute dürfen also nicht direkt über den Klassennamen oder – was leider in Java auch möglich ist – über eine Referenz angesprochen werden. Die Idee hinter diesen beiden Klassenattributen ist, dass sie für alle Busse bei jedem Zusteigen von Passagieren aktualisiert werden. Klassenattribute müssen in Java jeweils bei der Deklaration oder im Static-Initializer initialisiert werden. Wir definieren also die beiden Attribute wie folgt:

private static int totalPassagiere = 0;
private static double totalUmsatz = 0.0;
In der Gesamtübersicht weiter unten zeige ich Ihnen die Initialisierung über den Static-Initializer.
Die Attribute **linie**, **ticketPreis**, **umsatz** und **anzahlPassagiere** sind Instanzattribute. Sie sind gemäss dem Geheimhaltungsprinzip mit der Sichtbarkeit **private** modelliert. Jedes Objekt der Klasse Bus (jeder Bus) fährt auf einer bestimmten Linie im städtischen Netz, hat einen bestimmten Ticketpreis, egal wie lang die gefahrene Strecke ist, macht durch den Transport der Passagiere einen Umsatz der den zugestiegenen Passagieren mal dem Ticketpreis entspricht und hat eine zu einem bestimmten Zeitpunkt gerade aktuelle Anzahl Passagiere an Bord.

Bei Instanzattributen **fehlt** das Schlüsselwort **static**.
private String line;
private double ticketPreis;
private double umsatz;
private int anzahlPassagiere;
Bei der **Operation Bus** handelt es sich um den Konstrukor der Klasse. Eine Klasse kann mehrere Konstruktor-Operationen anbieten, allerdings können diese sich nur in der Anzahl und der Reihenfolge der Parameter unterscheiden. In Java gilt die strenge Konvention, dass der Name des Konstruktors gleich dem Namen der Klasse sein muss. Konstruktoren definieren keinen Rückgabewert und sind in der Regel **public**. Für Letzteres bildet z.B. das Singleton-Muster der GoF1 eine Ausnahme. Dort ist der Konstruktor private (ev. Protected) deklariert und die Objekterzeugung wird durch eine Klassenoperation (getInstance) kontrolliert. Der abgebildete Konstruktor mit zwei Übergabeparametern wird wie folgt definiert:
public Bus(String linie, **double** ticketPreis)
Die Operationen **getTotalPassagiere** und **getTotalUmsatz** sind Klassenoperationen – daher **static** – und mit der Sichtbarkeit **public** modelliert. Sie dienen als Accessor Operations (siehe LE02) und haben die Aufgabe, den Wert des entsprechenden Attributs zurückzugeben. Der Name von Accessor Operations setzt sich immer aus dem Präfix "get" gefolgt vom Attributnamen (erster Buchstabe gross) zusammen. Dieses Namensschema gilt sowohl für Klassenattribute als

auch für Instanzattribute. Update Operations haben den gleichen Namensaufbau, beginnen aber mit dem Präfix "set". Die Operation getAnzahlPassagiere definiert als Rückgabetyp einen int, die Operation getTotalUmsatz einen double. Die Verknüpfung zum Datentypen des gekapselten Attributs ist offensichtlich.

public static int getTotalPassagiere()
public static double getTotalUmsatz()
Bei den restlichen Operationen handelt es sich um Instanzoperationen. Ihnen **fehlt** also das Schlüsselwort **static**.
Die Operationen arbeiten direkt auf einem Objekt, können aber intern z.B. auch auf Klassenattribute zugreifen und diese manipulieren. Beachten Sie dazu die Beispielimplementierung der abgebildeten Klasse.

public String getLinie()
public double getTicketPreis()
public void zusteigen(**int** anzahlPassagiere)
public void aussteigen(**int** anzahlPassagiere)
public double getUmsatz()
Auch hier handelt es sich bei getLinie, getTicketPreis und getUmsatz um Accessor Operations, natürlich nicht um Klassenoperationen sondern eben um Instanzoperationen. Zum Schluss soll die Beispielimplementierung ein Gesamtbild über die Klasse Bus (in der Datei Bus.java) zeigen.

```
public class Bus {
public static final String PASSAGIERBEZEICHNUNG = "PAX";
private static double totalUmsatz;
private static int totalPassagiere;
private int anzahlPassagiere;
private double umsatz;
private double ticketPreis;
private String linie;
static {
totalUmsatz = 0.0;
totalPassagiere = 0;
}
public Bus(String linie, double ticketPreis) {
```

```
this.linie = linie;
this.ticketPreis = ticketPreis;
}
public void aussteigen(int anzahlPassagiere) {
this.anzahlPassagiere -= anzahlPassagiere;
}
public void zusteigen(int anzahlPassagiere) {
this.anzahlPassagiere += anzahlPassagiere;
this.umsatz = anzahlPassagiere * ticketPreis;
totalUmsatz += anzahlPassagiere * ticketPreis;
totalPassagiere += anzahlPassagiere;
}
public double getTicketPreis() {
return ticketPreis;
}
public String getLinie() {
return linie;
}
public static double getTotalUmsatz() {
return totalUmsatz;
}
public static int getTotalPassagiere() {
return totalPassagiere;
}
}
```

Listing 1: Umsetzung der Klasse Bus

Beziehungen

Eine **Beziehung**, egal in welcher Ausprägung, beschreibt den **Link** zwischen **Objekten** auf der **Ebene** der **Klasse**.

Ein solcher Link **sagt aus**, dass **sich** die **beiden Objekte** kennen. Dies **muss nachhaltig sein**, ist also genauso eine Eigenschaft des Objekts wie ein anderes Attribut auch. **Links** werden in der **Implementierung als Attribute realisiert**. Der **Name** des **Attributs** ergibt sich **aus** dem **Rollennamen**, oder wenn dieser **fehlt**, aus dem **Namen** der **Klasse** am anderen **Ende** der **Beziehung**. Handelt es sich um eine Beziehung zu mehreren Objekten

(Kardinalität > 1), wird diese mit einem Container aus dem Collection-Framework realisiert. Die Collection muss typisiert werden. Der Name des Attributs ergibt sich aus dem Rollennamen oder aus der Pluralform des Namens der referenzierten Klasse.
Von zentraler Bedeutung sind die Kardinalitäten der Beziehung.
Handelt es sich um Muss-Beziehungen, sind Sie bei der Implementierung verpflichtet, diese Vorgabe zu kontrollieren, so dass das Modell nicht in einen inkonsistenten Zustand gelangt.

Das Klassendiagramm aus Abbildung 1 dient als Fallbeispiel. Das Beispiel kommt in späteren Modulen im Zusammenhang mit Datenbank-Modellierung und Java Persistenzmechanismen wieder.
Es enthält alle sieben Beziehungstypen, welche im Rahmen des O/R-Mappings vorkommen und wurde für die Betrachtungen in diesem Arbeitsblatt erweitert. Bei den gezeigten Implementierungen handelt es sich um Vorschläge.
Selbstverständlich dürfen Sie sich auch eigene Strategien überlegen, vor allem was die Umsetzung von Constraints angeht (was eine Muss-Beziehung auch ist). Ich möchte das Modell und einige nicht sichtbare Vorgaben nun auch noch in einer Modellbeschreibung verdeutlichen.
Im Modell geht es um eine Projektverwaltung. Im Zentrum steht die Klasse Projekt mit ihren beiden Subklassen ProjektExtern und ProjektIntern.
Einem Projekt können beliebig viele *Bearbeiter*, welche Objekte der Klasse Mitarbeiter sind, zugeteilt werden. Umgekehrt kann ein Mitarbeiter in beliebig vielen Projekten mitarbeiten. In einem Projekt werden von verschiedenen Mitarbeitern Leistungen erbracht. Die Besonderheit an dieser Aggregation ist, dass jedes Projekt mindestens über ein Leistungsobjekt verfügt. Dies wirkt zwar etwas künstlich, ist aber nötig, da ich die Problematik der Muss-Beziehungen mit beliebig vielen Objekten aufzeigen möchte. In der Praxis könnte dies so argumentiert werden, dass die Eröffnung des Projekts im System immer eine Leistung darstellt und somit angelegt werden muss. Eine Leistung wird von einem bestimmten Mitarbeiter erbracht (*erbringer* der Leistung). Bei internen Projekten kann der *Projektleiter* erfasst werden, muss aber nicht.
Ein Mitarbeiter gehört zu genau einer Abteilung, die Abteilung hat keine, einen oder mehrere Mitarbeiter und einen *Abteilungsleiter*. Ein Mitarbeiter hat keinen (der Chef) oder maximal einen *Vorgesetzten*, der nicht

unbedingt der Abteilungsleiter seiner Abteilung sein muss. Daraus ergibt sich, dass ein Mitarbeiter in seiner Eigenschaft als Vorgesetzter keinen, einen oder mehrere *Untergeben* haben kann. Aus Gründen der Wiederverwendung wurde die Adresse als eigene Klasse ausmodelliert. Ein Mitarbeiter hat sein ***Domizil*** an einer bestimmten Adresse. Bei den drei Attributen handelt es sich um zwingende Attribute.
Beziehungen mit einem **x** auf der Linie sind in die Richtung des x nicht navigierbar. Das bedeutet, dass z.B. Der Mitarbeiter nicht herausfinden kann, ob er Abteilungsleiter einer bestimmten Abteilung ist. Eine Beziehung die in beiden Richtungen offene Pfeile aufweist, ist in beide Richtungen explizit navigierbar. Diese Beziehungsart könnte ebenfalls mit zwei unidirektionalen Beziehung ersetzt werden. Zwischen Mitarbeiter und Abteilung besteht die Beziehung, welche die Abteilungszugehörigkeit modelliert. Sie ist explizit bidirektional. Eine gleichartige Beziehung (allerdings mit anderer Kardinalität) modelliert die Tatsache, dass ein Mitarbeiter Bearbeiter in einem Projekt ist. Diese Beziehung wurde als Beispiel mit zwei unidirektionale Beziehungen abgebildet.

Beachten Sie, dass die Kardinalität 1 nicht angegeben wird oder umgekehrt formuliert ist überall dort, wo die Angabe der Kardinalität fehlt, die Kardinalität gleich 1. Ist die Beziehung nicht navigierbar, ist die Kardinalität unwichtig und wird ebenfalls nicht angegeben.
Ich verwende bei auftretenden Problemen die ProjektException, um dem Client dieser Klassen das Problem mitzuteilen.

Together unterstützt sie nur bedingt bei der Umsetzung und verhält sich in einigen Spezialsituation sogar falsch.
Lassen Sie sich nicht davon beirren.

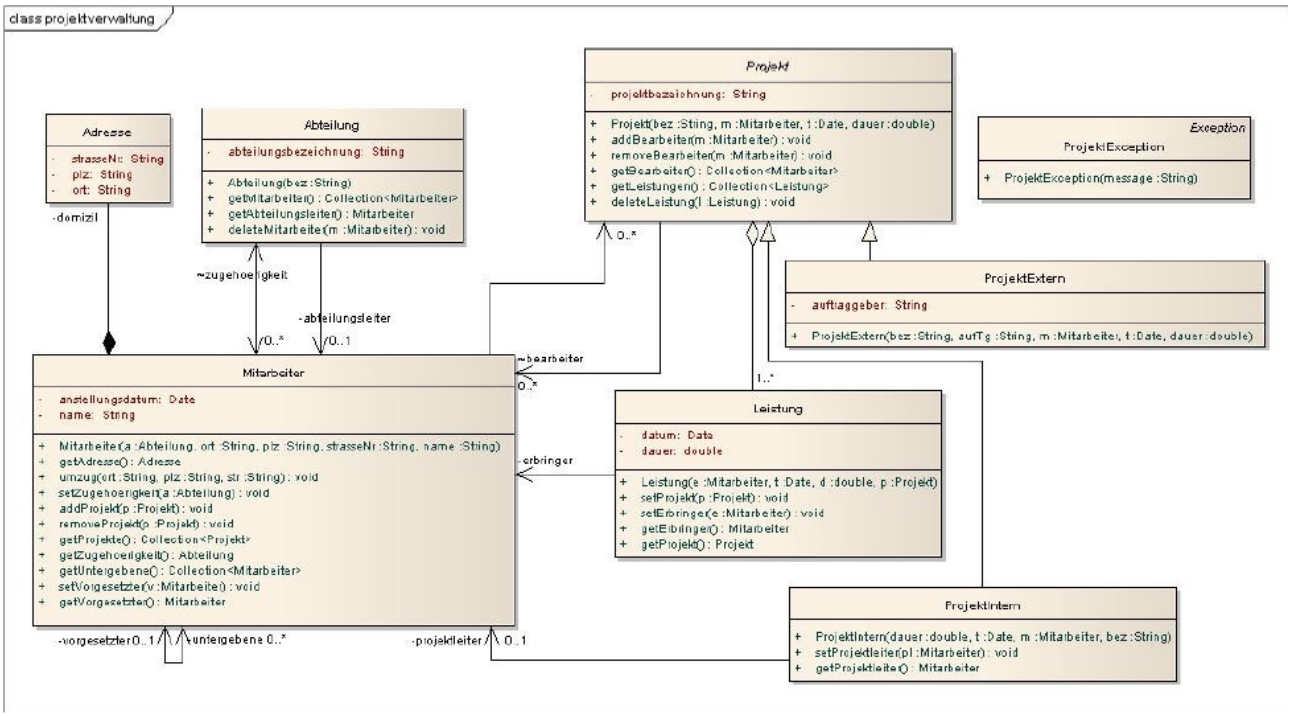


Abbildung 1: Klassendiagramm Übersicht "Projektverwaltung"

Abbildung 1: Klassendiagramm Übersicht "Projektverwaltung"

Sicherstellen von Muss-Beziehungen

Als Entwickler sind Sie dafür verantwortlich, dass die Beziehungen gemäss den Design-Vorgaben umgesetzt werden. Unangenehm – weil aufwändig – ist die Implementierung von Muss-Beziehungen. Glücklicherweise treten Sie nicht so häufig auf. Sie sind vor allem in der Umsetzung von Kompositionen von grosser Bedeutung.

In der Projektverwaltung finden Sie zwei Beispiele: Die Komposition zwischen Mitarbeiter und Adresse und die **Aggregation** zwischen **Projekt** und **Leistung**. Die Angabe einer Adresse ist zwingend, da die Kardinalität genau 1 ist. Vergessen wir nicht, dass die drei **Attribute der Klasse Adresse zwingende Angaben sind**. Durch die Modellierung einer **Komposition** wird die Verantwortung des Lebenszyklus der Adresse ausserdem an die Klasse Mitarbeiter delegiert. **Streng genommen darf kein Client ein Objekt der Klasse Adresse erzeugen**, bzw. eigentlich Kenntnis dieser Struktur haben. Letztere Anforderung liesse sich durch eine innere Klasse realisieren. Um einen möglichst grossen Profit aus der Extraktion der Adresse aus Mitarbeiter zu erhalten, wird zwar eine öffentliche Klasse verwendet, das angelegte **Adress-Objekt** aber nie an den Client zurückgegeben. Damit ist eine Modifikation von "aussen" nicht möglich und die Wahrung der

Konsistenz kann durch die Klasse Mitarbeiter effektiv wahrgenommen werden, d.h. Die Modifikation der Adresse kann nur über die Klasse Mitarbeiter erfolgen. Betrachten Sie die folgende Implementierung.

```
public class Adresse {
private String ort;
private String plz;
private String strasseNr;
public Adresse(String str, String plz, String ort) throws ProjektException {
setOrt(ort);
setPlz(plz);
setStrasseNr(str);
}
public String getOrt() {
return ort;
}
public String getPlz() {
return plz;
}
public String getStrasseNr() {
return strasseNr;
}
public void setOrt(String o) throws ProjektException {
if (o == null || o.equals(""))
throw new ProjektException("Der Ort ist eine zwingende Angabe");
ort = o;
}
public void setPlz(String p) throws ProjektException {
if (p == null || p.equals(""))
throw new ProjektException("Die PLZ ist eine zwingende Angabe");
plz = p;
}
public void setStrasseNr(String s) throws ProjektException {
if (s == null || s.equals(""))
throw new ProjektException("Die Strasse ist eine zwingende Angabe");
strasseNr = s;
}
}
```

Listing 2: Implementierung der Klasse Adresse

Die "Bewachung" der Konsistenz der Attribute strasseNr, plz und ort wird in den **Setter-Methoden** vollzogen.
Jeder Versuch, das Attribut mit **null** oder mit einem **Leerstring** zu belegen wird mit dem Werfen der **ProjektException** beantwortet. Beachten Sie den etwas technischen Unterschied zwischen einer Null-Referenz
und einem Leerstring. Wenn Benutzer von Muss-Attributen reden, werden Sie diesen Unterschied nie machen können. Für sie bedeutet ein Muss-Attribut ein Attribut, das einen (gültigen) Wert beinhalten muss. Der Konstruktor verwendet natürlich ebenfalls die Setter-Methoden zur Initialisierung der Werte, damit auch bei der Erzeugung des Objekts die Objektkonsistenz gewahrt ist und wirft die Exception weiter, falls einer der Werte ungültig sein sollte.
Die Komposition zwischen Mitarbeiter und Adresse ist unidirektional, was bei Kompositionen die Regel ist. Daher kennt die Adresse den Mitarbeiter nicht. Die Komposition mit einer Muss-Kardinalität zur Adresse bedeutet aber aus der Sicht des Mitarbeiters auch, dass ein Mitarbeiter nur gültig ist, wenn auch ein Adress- Objekt angelegt werden kann. Dies bedeutet wiederum, dass ein Mitarbeiter-Objekt die zwingenden Werte für das Adress-Objekt bereits im Konstruktor erhalten muss.

```
public class Mitarbeiter {
private Adresse domizil;
private Date anstellungsdatum;
private String name;
...
public Mitarbeiter(String name, String strasseNr, String plz,
String ort, Abteilung a) throws ProjektException {
domizil = new Adresse(strasseNr, plz, ort);
anstellungsdatum = new Date();
untergebene = new LinkedList<Mitarbeiter>();
setZugehoerigkeit(a);
}
public Adresse getAdresse() throws ProjektException {
Adresse a = new Adresse(domizil.getStrasseNr(),
domizil.getPlz(), domizil.getOrt());
```

```
return a;
}
public void umzug(String strasseNr, String plz, String ort)
throws ProjektException {
domizil.setStrasseNr(strasseNr);
domizil.setPlz(plz);
domizil.setOrt(ort);
}

}
```

Listing 3: Code-Ausschnitt der Mitarbeiter-Klasse (Adresse betreffend)

Die Realisierung der Beziehung erfolgt über das Attribut domizil. Wird ein Mitarbeiter-Objekt erzeugt, wird dieser Referenz automatisch auch ein Objekt zugewiesen. Sollte das Adress-Objekt aus den bekannten Gründen nicht erzeugt werden können, wird durch das Weiterreichen der Exception auch die Erzeugung des Mitarbeiter-Objekts verhindert.

In **der Regel** liefert eine **Getter-Methode** nur gerade den **Attribut-Wert** zurück der im **angefragten Objekt** gekapselt wird. Um sicherzustellen, dass eine **Manipulation von Aussen gänzlich unmöglich ist** – denn die darf aufgrund der **Komposition** nur über das **Mitarbeiter-Objekt** erfolgen – liefert die Methode ***getAdresse*** nur eine **Kopie des Adress-Objekts**. Die Änderung der Adresse wird über die Methode ***umzug*** zur Verfügung gestellt.

Bidirektionale Muss-Beziehungen

Die Beziehung zwischen Mitarbeiter und Abteilung, welche die **Abteilungszugehörigkeit** eines Mitarbeiters beschreibt ist eine **Muss-Beziehung**. Ein Mitarbeiter **muss** zu einer **Abteilung** gehören. Allerdings liegen hier die Dinge etwas anders als im obigen Beispiel einer **Muss-Beziehung**. Ein Mitarbeiter kann durchaus die **Abteilung wechseln**. Da die Beziehung bidirektional ist, muss die K**onsistenz der Beziehung sichergestellt** werden.

Referenziert ein **Mitarbeiter** eine **Abteilung**, **muss er umgekehrt** in der **Collection aller Mitarbeiter dieser Abteilung** aufgenommen werden. **Wechselt** der Mitarbeiter die **Abteilung über das Mitarbeiter-Objekt**, **muss er** bei der **"alten"** Abteilung auch aus der **Collection** der Abteilungsmitarbeiter **entfernt** und bei der **neuen** Abteilung

hinzugefügt werden. Wird ein Mitarbeiter über die **Abteilung derselben** hinzugefügt, **muss er bei der alten Abteilung auch entfernt** werden. Die Pflege dieser Beziehung ist äusserst fehleranfällig. Deshalb liefert die **Getter-Methode der Collection** auf der Abteilungsseite nur eine **Kopie** der Collection. Damit ist die Beziehung vor unqualifizierten Modifikationen geschützt.

```
public class Mitarbeiter {
Abteilung zugehoerigkeit;
...
public Mitarbeiter(String name, String str, String plz, String ort,
Abteilung a) throws ProjektException {
domizil = new Adresse(str, plz, ort);
setZugehoerigkeit(a);
this.name = name;
anstellungsdatum = new Date();
projekte = new LinkedList<Projekt>();
}
public void setZugehoerigkeit(Abteilung a) throws ProjektException {
if (a == null)
throw new ProjektException("Abteilung ist eine zwingende Angabe");
if (zugehoerigkeit != null)
zugehoerigkeit.mitarbeiter.remove(this);
zugehoerigkeit = a;
zugehoerigkeit.mitarbeiter.add(this);
}
public Abteilung getZugehoerigkeit() {
return zugehoerigkeit;
}
}
```

Listing 4: Code-Ausschnitte aus der Mitarbeiter-Klasse

Auch hier wird die Umsetzung des Muss-Attributs ***zugehoerigkeit*** durch die **Setter-Methode** sichergestellt. Sollte eine **Null-Referenz** übergeben werden, wird eine **ProjektException** geworfen. Diese wird z.B. im **Konstruktor** einfach nur **durchgereicht**. Damit gilt für das Mitarbeiter-Objekt eine weitere **Regel**: Die Abteilung muss bereits bei der Erzeugung mitgegeben

werden. Sollte die Setter-Methode aus einem anderen Kontext aufgerufen werden, ist die Referenz ***zugehoerigkeit*** in jedem Fall nicht null. Damit geht die Methode davon aus, dass der Mitarbeiter die Abteilung wechselt. Er muss also bei der alten Abteilung aus der Collection der Mitarbeiter gelöscht werden. Gleichzeitig wird er bei der neuen Abteilung hinzugefügt. Die Manipulation der Beziehung ist damit konsistent erfolgt und die Bidirektionalität ist gepflegt worden.
Bei **bidirektionalen Beziehungen** stellt sich immer wieder die Frage, wie die Pflege der Beziehung diese Bidirektionalität sicherstellt. Auf der einen Seite möchte man die beiden Beziehungsattribute (hier zugehoerigkeit und mitarbeiter) vor direkten **Manipulationen schützen**, andererseits ist ein Zugriff von der einen Klasse zur anderen notwendig und muss daher über einen **Accessor** auf das jeweilige Attribut zugreifen.
Eine mögliche Strategie ist die hier angewendete. Wenn wir davon ausgehen, dass alle Klassen des Modells (im Sinne des Entity-Models) im gleichen Paket sind, können wir die Beziehungsattribute als „**friendly**“ deklarieren.
Damit können andere Klassen aus dem Modell auf die Beziehungsattribute zugreifen. Jede an der Beziehung beteiligte Klasse stellt dann ihre Methoden zur Verfügung, um die zulässigen Manipulationen abzudecken.
Im vorliegenden Fall habe ich die Pflege der Beziehungsattribute vollständig über die Methoden Mitarbeiter.setZugehoerigkeit() und Abteilung.deleteMitarbeiter() abgedeckt. Funktional sind folgende Anwendungsfälle vorstellbar:

1. **Der Mitarbeiter wird neu erfasst.**
2. **Der Mitarbeiter wird gelöscht.**
3. **Der Mitarbeiter wechselt die Abteilung.**

Alle drei Fälle können technisch mit dem zur Verfügung gestellten Interface abgewickelt werden. Fall 1 ist mit dem Konstruktor abgedeckt. Intern wird **setZugehoerigkeit()** aufgerufen, was die Anwendung aller Constraints sicherstellt. Der zweite Fall wird über die Methode **Abteilung.deleteMitarbeiter()** abgedeckt. Fall drei kann ebenfalls über die Methode **Mitarbeiter.setZugehoerigkeit()** abgewickelt werden.
Der gelockerte Zugriffsschutz auf die Beziehungsattribute ist ein Kompromiss, der eingegangen werden kann.
Die Gepflogenheiten in der Paketstrukturierung einer Applikation isolieren die Entity-Klassen in einem eigenen Paket. Alle anderen Applikationsteile

wie Client, Businesslogik und Datenhaltung sind in einem anderen Paket abzulegen und haben daher keinen Zugriff auf die Beziehungsattribute. Beachten Sie den JavaDoc-Kommentar zu **deleteMitarbeiter()**. Es ist wichtig, dass die Methode wirklich nur beim Löschen eines Mitarbeiters aus dem System zur Anwendung kommt. Deshalb sollte man das dem Benutzer (der ja auch ein Entwickler ist) mitteilen.

```
public class Abteilung {
Collection<Mitarbeiter> mitarbeiter;
...
public Abteilung(String bez) {
mitarbeiter = new LinkedList<Mitarbeiter>();
abteilungsbezeichnung = bez;
}
public Collection<Mitarbeiter> getMitarbeiter() {
Collection<Mitarbeiter> kopie = new LinkedList<Mitarbeiter>();
for (Mitarbeiter m : mitarbeiter)
kopie.add(m);
return kopie;
}
/**
* Löschen eines Mitarbeiters aus der Abteilung. Das Mitarbeiterobjekt darf
* von Clients anschliessend nicht mehr verwendet werden, da die
Konsistenz
* nicht gewährleistet ist. Die Methode darf nur angewendet werden, wenn
* der Mitarbeiter anschliessend ganz aus dem System entfernt wird.
*
* @param m
* das zu entfernende Mitarbeiterobjekt
* @throws ProjektException
* falls das Objekt null ist
*/
public void deleteMitarbeiter(Mitarbeiter m) throws ProjektException {
if (m == null)
throw new ProjektException("Nonsense");
m.zugehoerigkeit.mitarbeiter.remove(m);
m.zugehoerigkeit = null;
```

```
}
...
}
```

Listing 5: Code-Ausschnitt aus der Klasse Abteilung

Die Abbildung der 0..* Kardinalität der Mitarbeiter-Beziehung erfolgt über eine Collection vom Typ Mitarbeiter.
Der Konstruktor initialisiert die Beziehung mit einer leeren Collection.
Reflexive (Kann-) Beziehungen Die reflexive Beziehung zur Abbildung von Vorgesetzen und Untergebenen weist auf der Seite der Vorgesetzten-Rolle die **Kardinalität 0..1** auf. Dies ist typisch für reflexive Beziehung. Eine reflexive Beziehung lässt sich immer als Baumstruktur darstellen. Der einzige Knoten im Baum, der keinen Vorgänger aufweist, ist der Root-Knoten. Allerdings weicht natürlich genau diese eine Ausnahme auch unser Modell sehr stark auf.
Zur Umsetzung der Beziehung benötigt der Mitarbeiter also zwei Attribute. Auf der einen Seite eine Collection von Mitarbeitern mit der Rolle (gleichzeitig Attributname) ***untergebene***, auf der anderen Seite eine Referenz auf einen Mitarbeiter, der die Rolle des **Vorgesetzten** inne hat. Die **Bidirektionalität** der Beziehung verlangt eine **strenge Kapselung der Beziehungsattribute**, damit keine **unvollständigen Modifikationen vorgenommen** werden können. Daher gibt die **Getter-Methode** für die Collection der **Untergebenen** nur eine **Kopie** der im Mitarbeiter-Objekt eingelagerten Beziehung zurück. Es gibt auch **keine Setter-Methode**, da diese **Operation sehr aufwändig** in der Implementierung ist. Da es sich beim Vorgesetzten um eine Kann-Beziehung handelt, sind **Null-Werte** zulässig. Aus Sicht einer **bidirektionalen Beziehung** kann ein Mitarbeiter aus der Liste der **Untergebenen** entfernt werden und der **Vorgesetzte** eines **Mitarbeiters** auf Null gesetzt werden. Die Pflege der Beziehung erfolgt ausschliesslich über die Methode setVorgesetzter. Die Implementierung kümmert sich sowohl um den vorherigen Vorgesetzten als auch um den

```
public class Mitarbeiter {
private Mitarbeiter vorgesetzter;
private Collection<Mitarbeiter> untergebene;
...
public Mitarbeiter(String name, String str, String plz, String ort,
Abteilung a) throws ProjektException {
```

```
domizil = new Adresse(str, plz, ort);
setZugehoerigkeit(a);
this.name = name;
anstellungsdatum = new Date();
projekte = new LinkedList<Projekt>();
untergebene = new LinkedList<Mitarbeiter>();
}
public Collection<Mitarbeiter> getUntergebene() {
Collection<Mitarbeiter> kopie = new LinkedList<Mitarbeiter>();
for (Mitarbeiter u : untergebene)
kopie.add(u);
return kopie;
}
public void setVorgesetzter(Mitarbeiter v) {
if (vorgesetzter != null)
vorgesetzter.untergebene.remove(this);
vorgesetzter = v;
if (v != null)
vorgesetzter.untergebene.add(this);
}
public Mitarbeiter getVorgesetzter() {
return vorgesetzter;
}
...
}
```

Listing 6: Code-Ausschnitt aus der Mitarbeiter-Klasse

Muss-Beziehung mit 1..* Kardinalität

Das abgebildete Beispiel zwischen Projekt und Leistung geht davon aus, dass der Aufwand für die Projekteröffnung oft vergessen wird und zwingt damit den Benutzer, diesen Aufwand immer zu spezifizieren.
Denkbar ist auch, dass in einem Client Vorgabewerte hinterlegt werden können. In der Umsetzung müssen also bereits im Konstruktor des Projekts die Werte zur Erstellung des Leistungsobjekts mitgegeben werden. Da es sich um eine **Aggregation handelt**, könnte das Leistungsobjekt auch vom Client her erstellt werden und dieses fertige Objekt dem Projekt-Konstruktor übergeben werden. C'est à vous de choisir! Da eine

Aggregation modelliert wurde, wird die Verantwortung für die Leistungsobjekte tendenziell den Projektobjekten übertragen.
Man geht – ähnlich wie bei der **Komposition** – davon aus, dass die Pflege der Beziehung über das **Aggregat** erfolgt. Die Ausnahme bestätigt aber auch hier die **Regel**: Die gezeigte Implementierung benutzt **Leistung.setProjekt()** zur Pflege der Beziehung.
Da die **Beziehung Bidirektional ist**, wird das **Beziehungsattribut gekapselt** und die **Getter-Methode liefert** nur gerade eine **Kopie** der eingelagerten **Leistungsreferenzen**. Die einzige Möglichkeit, eine Leistung vollständig zu löschen, bietet die Methode **deleteLeistung**. Auch hier ist ein entsprechender Kommentar im JavaDoc sinnvoll.

```
public abstract class Projekt {
Collection<Leistung> leistungen;
...
public Projekt(String bez, Mitarbeiter m, Date t, double dauer)
throws ProjektException {
this.projektbezeichnung = bez;
bearbeiter = new LinkedList<Mitarbeiter>();
leistungen = new LinkedList<Leistung>();
leistungen.add(new Leistung(m, t, dauer, this));
}
public Collection<Leistung> getLeistungen() {
Collection<Leistung> kopie = new LinkedList<Leistung>();
for (Leistung l : leistungen)
kopie.add(l);
return kopie;
}
/**
* Löschen einer Leistung. Das Leistungsobjekt darf von Clients
* anschliessend nicht mehr verwendet werden, da die Konsistenz nicht
* gewährleistet ist.
*
* @param l
* das zu entfernende Leistungsobjekt
* @throws ProjektException
* falls das Objekt null ist oder das letzte Objekt entfernt
```

```
* werden soll.
*/
public void deleteLeistung(Leistung l) throws ProjektException {
if (l == null)
throw new ProjektException("Nonsense");
if (leistungen.size() == 1) // Mindestens 1 Objekt muss vorhanden sein
throw new ProjektException(
"Es muss mindestens eine Leistung vorhanden sein.");
l.projekt = null;
if (leistungen.contains(l))
leistungen.remove(l);
}       }
```

Listing 7: Code-Ausschnitt aus der Projekt-Klasse


```
public class Leistung {
private Mitarbeiter erbringer;
Projekt projekt;
public Leistung(Mitarbeiter e, Date t, double d, Projekt p)
throws ProjektException {
setProjekt(p);
setErbringer(e);
datum = t;
dauer = d;
}
public void setProjekt(Projekt p) throws ProjektException {
if (p == null)
throw new ProjektException("Projekt ist eine zwingende Angabe");
if (projekt != null)
projekt.leistungen.remove(this);
projekt = p;
projekt.leistungen.add(this);
}
public void setErbringer(Mitarbeiter e) throws ProjektException {
if (e == null)
throw new ProjektException("Mitarbeiter ist eine zwingende Angabe");
```

```
erbringer = e;
}
public Projekt getProjekt() {
return projekt;
}
}
```

Listing 8: Code-Ausschnitt aus der Klasse Leistung

Kann-Beziehungen

Kann-Beziehungen sind in der Handhabung eher einfach. Sie stellen in der Regel für das Beziehungsattribut bei einer 0..1-Beziehung eine **Getter- und eine Setter-Methode zur Verfügung**, welche die Pflege der Beziehung erlaubt. Wollen Sie die Beziehung aufheben, übergeben Sie eine Null-Referenz. Bei einer **0..***-Beziehung reicht eigentlich das zur Verfügung stellen der **Setter- und Getter-Methoden** auch aus, da der Client die Modifikationen auch selber vornehmen kann.
Bei **bidirektionalen Beziehungen** sollte die **Collection aber gekapselt** werden. Dannbrauchen Sie eine **add- und eine remove-Methode**, die beide Seiten konsistent aktualisiert.

Modellkonsistenz in komplexeren Architekturen

Der bisher gezeigte Lösungsansatz zur Sicherstellung der im Modell definierten Constraints geht von einer einfachen Software-Architektur aus. Die hier gezeigten Entity-Klassen werden im Zusammenhang mit einer Mehrschichten-Architektur als Model-Klassen bezeichnet. Diese Architektur verfügt über eine sogenannte **Business-Logik-Schicht**. **Sämtliche Constraints** werden in dieser Architektur durch diese Schicht sichergestellt. Alle **Modifikationen an den Entity-Objekten** werden an diese **Schicht delegiert**. Damit werden alle Regeln an einem zentralen Ort abgehandelt. Dieses Verfahren hat aber auch Nachteile. So ist die Versuchung jeweils gross, direkt Entity-Objekte zu bearbeiten, da dies nur schwer (Java Security) verhindert werden kann. Eine Mehrschichten-Architektur ist trotzdem zu bevorzugen. Sie ist ein zentrales Thema in Informatik IV.

Im Zentrum steht die Forderung, dass alle Regeln, welche im Modell und dessen Beschreibung definiert werden, in der Implementierung auch wirklich umgesetzt werden müssen. Das fängt bei Pflichtattributen an, geht über die Sicherstellung der Bidirektionalität und reicht bis zum sauberen

Abbilden von **Muss-Beziehungen**. Die Mittel dazu sind **Kapselung** und **Exception-Handling**. **Ungültige Operationen** werden mit dem Werfen einer **Exception geahndet** und zwingen so den Client zur korrekten Benutzung der Entity-Objekte. **Kapselung hilft uns**, die **Beziehungsattribute**, seien es **direkte Referenzen** oder **Collections**, vor falschen **Manipulationen** zu **schützen**.

Die hier gezeigten Lösungen sind lediglich Vorschläge. Die gesteckten Ziele können durchaus auch anders erreicht werden. Denken Sie daran:

Viele Wege führen nach Rom, aber, Rom ist immer am gleichen Ort!

Live Example : Simple Class Program [Area of Rectangle]

```
class Rectangle {
  double length;
  double breadth;
}

// This class declares an object of type Rectangle.
class RectangleDemo {
  public static void main(String args[]) {
    Rectangle myrect = new Rectangle();
    double area;

    // assign values to myrect's instance variables
    myrect.length = 10;
    myrect.breadth = 20;

    // Compute Area of Rectangle
    area =  myrect.length * myrect.breadth ;

    System.out.println("Area is " + area);
  }
}
```

Output :

```
C:Priteshjava>java RectangleDemo
Area is 200.0
```

[336×280]

Explanation : Class Concept

1. Class Creation / Declaration :

Consider following class –

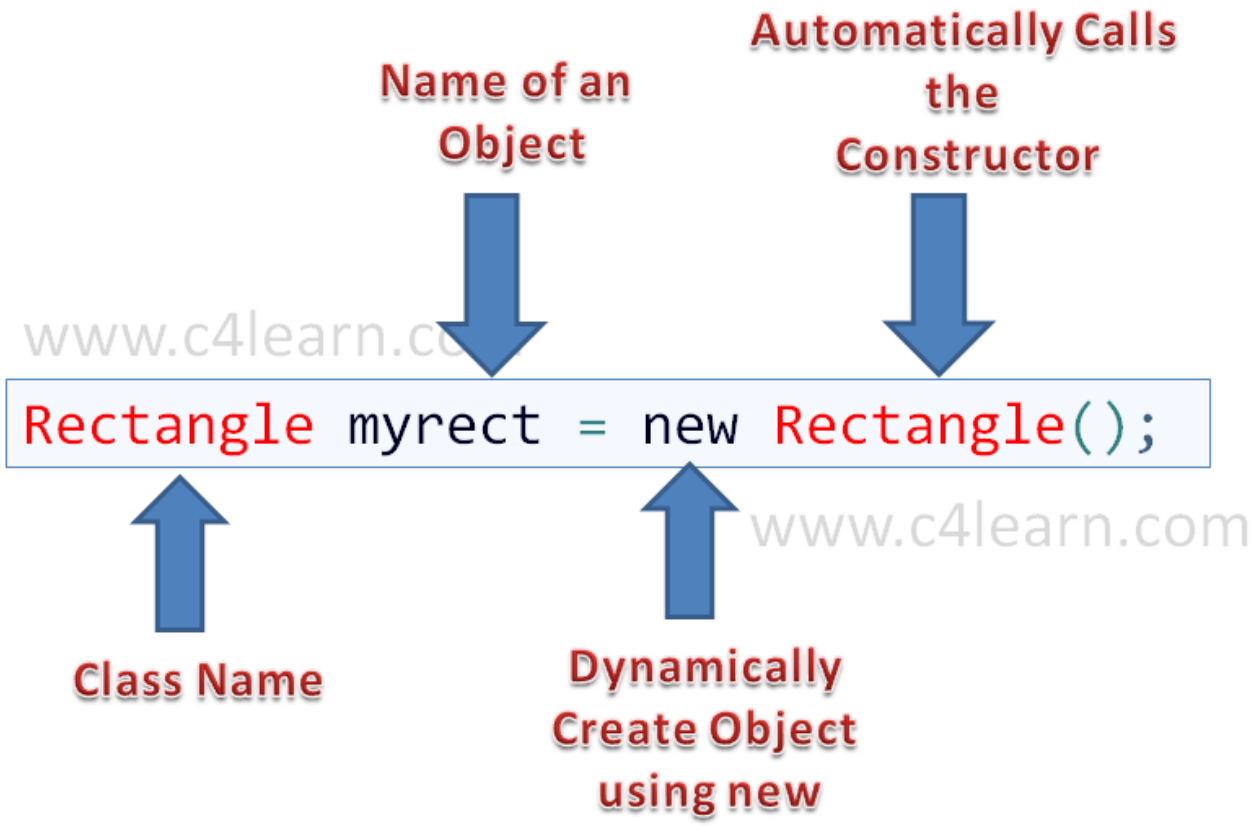
```
class Rectangle {
  double length;
  double breadth;
}
```

- **class** defines a **new type of data**.
- **Rectangle** is our new data type.
- **“Rectangle”** will be used to **declare objects of type Rectangle**.
- class declaration only creates a template. It does **not create an actual object**.

2. Creation of Object

```
Rectangle myrect = new Rectangle();
```

- Actual Creation of Object.
- **Memory is allocated for an object** after executing this statement.
- This Statement will create instance of the class “**Ractangle**” and name of instance is nothing but actual object “**myrect**“.
- **Fresh copy of Instance variables** gets created for Fresh Instance. [myrect now have its own instance variables -> length,breadth]
- Following fig shows one unknown term – **Constructor** (Don’t worry about this term , you will get more details in the incoming chapters)



3. Accessing Instance Variables Of Object/Instance using DOT Operator

```
myrect.length  = 10;
myrect.breadth = 20;
```

- As explained earlier , Each Instance/Object gets their **own copy of instance variables i.e length and breadth**.
- We can access "**myrect's**" copy of instance variable using "**DOT**" operator (as shown above).

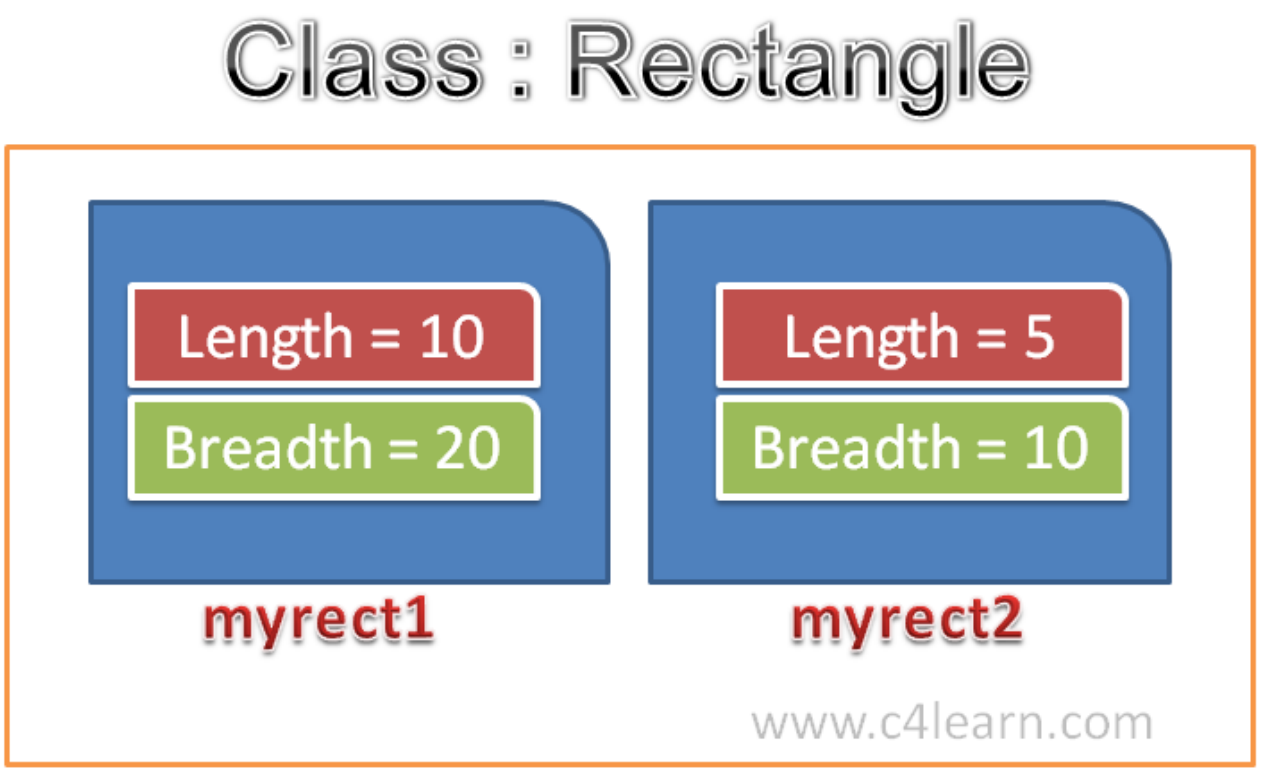
[468×60]

Steps to Run Above Program

1. Create RectangleDemo.java . **[suppose your file have multiple classes then you must save file with class name that contain main class]**
2. Copy Above Program into **RectangleDemo.java** and save it.
3. Run **MyRectangle.java** inside **Command Prompt**.

```java
class Rectangle {
  double length;
  double breadth;
}

// This class declares an object of type Rectangle.
class RectangleDemo {
  public static void main(String args[]) {
    Rectangle myrect1 = new Rectangle();
    Rectangle myrect2 = new Rectangle();
    double area1,area2;

    // assign values to myrect1's instance variables
    myrect1.length = 10;
    myrect1.breadth = 20;

    // Compute Area of Rectangle
    area1 =  myrect1.length * myrect1.breadth ;
    System.out.println("Area of Rectange 1 : " + area1);

    // assign values to myrect2's instance variables
    myrect2.length = 10;
    myrect2.breadth = 20;

    // Compute Area of Rectangle
    area1 =  myrect2.length * myrect2.breadth ;
    System.out.println("Area of Rectange 2 : " + area2);

  }
}
```

Explanation :



Each Object has its own set of Instance Variables :

1. In the above program we have created two objects of the class “**Rectangle**” , i.e myrect1,myrect2

```java
Rectangle myrect1 = new Rectangle();
Rectangle myrect2 = new Rectangle();
```

2. As soon as above two statements gets executed , two objects are created with specimen copy of their instance variables.
3. In short myrect1’s version of lemgth and breadth gets created . Similarly myrect2’s version of length and breadth gets created.
4. Using dot Operator we can access instance variable of respective object.

If you want to access instance variable of **myrect1** Object –

```java
myrect1.length  = 10;
myrect1.breadth = 20;
```

If you want to access instance variable of **myrect2** Object –

```java
myrect2.length  = 5;
myrect2.breadth = 10;
```

5. We can assign different values to instance variables of object. **Instance variable of different objects though have same name , they have different memory address , different value**.
6. It is important to understand that **changes to the instance variables of one object have no effect on the instance variables of another**.

Constructors : Initializing an Class Object in Java Programming

1. Objects contain there **own copy of Instance Variables**.
2. It is very difficult to **initialize each and every instance variable** of each and every object of Class.
3. Java allows **objects to initialize themselves when they are created**. Automatic initialization is performed through the use of a constructor.
4. A Constructor **initializes an object** as soon as object gets created.
5. Constructor gets **called automatically after creation of object and before completion of new Operator**.

Some Rules of Using Constructor :

1. Constructor **Initializes an Object**.
2. Constructor **cannot be called** like methods.
3. Constructors **are called automatically** as soon as object gets created.
4. Constructor **don't have any return Type.** (even Void)
5. Constructor name is same as that of "**Class Name**".
6. Constructor **can accept parameter**.

Live Example : How Constructor Works ?

```
class Rectangle {
  int length;
  int breadth;

  Rectangle()
  {
  length  = 20;
  breadth = 10;
  }

}

class RectangleDemo {
  public static void main(String args[]) {

  Rectangle r1 = new Rectangle();

  System.out.println("Length of Rectangle : " + r1.length);
  System.out.println("Breadth of Rectangle : " + r1.breadth);

  }
}
```

Output :

```
C:Priteshjava>javac RectangleDemo.java

C:Priteshjava>java RectangleDemo
Length  of Rectangle : 20
Breadth of Rectangle : 10
```

Explanation :

1. new Operator will create an object.
2. As soon as Object gets created it will call Constructor-

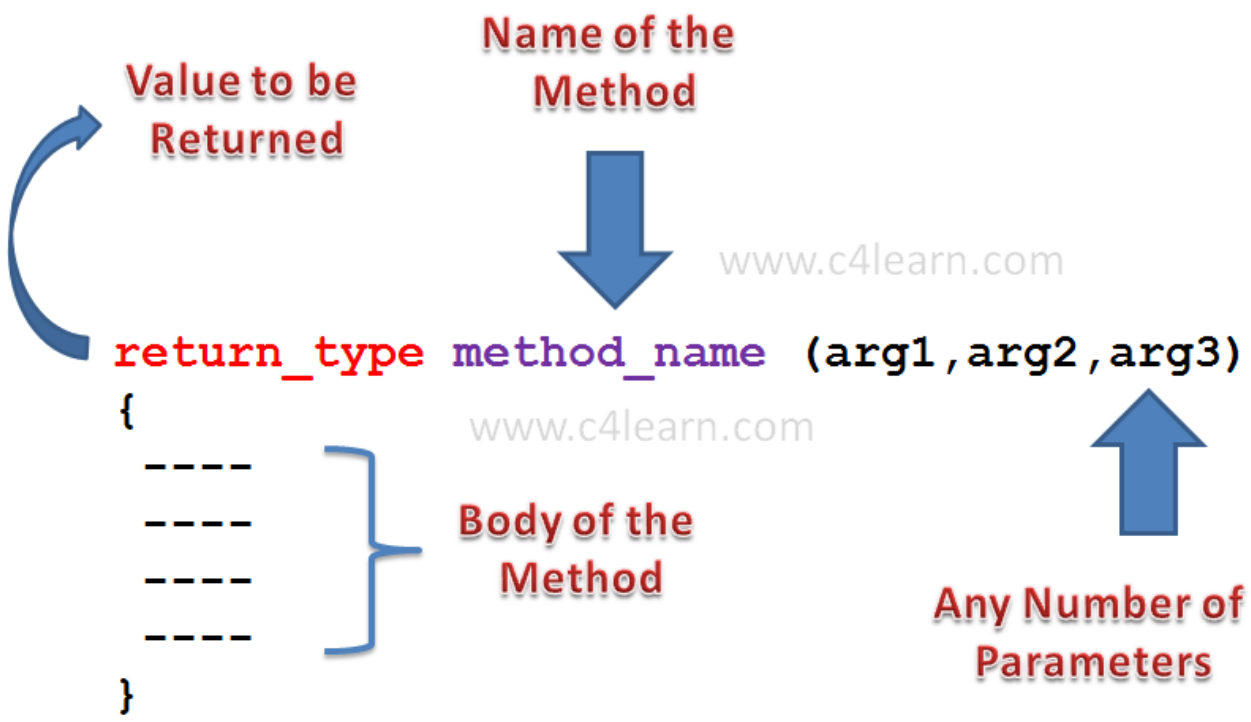
```
Rectangle()   //This is Constructor
  {
  length  = 20;
  breadth = 10;
  }
```

3. In the above Constructor Instance Variables of Object r1 gets their own values.
4. Thus Constructor **Initializes an Object as soon as after creation**.
5. It will print Values initialized by Constructor –

```
System.out.println("Length of Rectangle : " + r1.length);
System.out.println("Breadth of Rectangle : " + r1.breadth);
```

Introducing Methods in Java Class : Class Concept in Java

1. In Java Class , We can add user defined method.
2. Method is equivalent to Functions in C/C++ Programming.



Syntax : Methods in Java Classes

```
return_type method_name ( arg1 , arg2 , arg3 )
```

1. **return_type** is nothing but the value to be returned to an calling method.
2. **method_name** is an name of method that we are going to call through any method.
3. **arg1,arg2,arg3** are the different parameters that we are going to pass to a method.

Return Type of Method :

1. Method can return any type of value.
2. Method can return any Primitive data type

```
int sum (int num1,unt num2);
```

3. Method can return Object of Class Type.

```
Rectangle sum (int num1,unt num2);
```

4. Method sometimes may not return value.

```
void sum (int num1,unt num2);
```

Method Name :

1. Method name must be valid identifier.
2. All Variable naming rules are applicable for writing Method Name.

Parameter List :

1. Method can accept any number of parameters.
2. Method can accept any data type as parameter.
3. Method can accept Object as Parameter
4. Method can accept no Parameter.
5. Parameters are separated by Comma.
6. Parameter must have Data Type

Live Example : Introducing Method in Java Class

```
class Rectangle {
  double length;
  double breadth;

  void setLength(int len)
  {
  length = len;
  }
}

class RectangleDemo {
  public static void main(String args[]) {

  Rectangle r1 = new Rectangle();

  r1.length = 10;
  System.out.println("Before Function Length : " + r1.length);

  r1.setLength(20);
  System.out.println("After Function Length : " + r1.length);

  }
}
```

Output :

```
C:Priteshjava>java RectangleDemo
Before Function Length : 10.0
After Function Length : 20.0
```

Explanation :

Calling a Method :

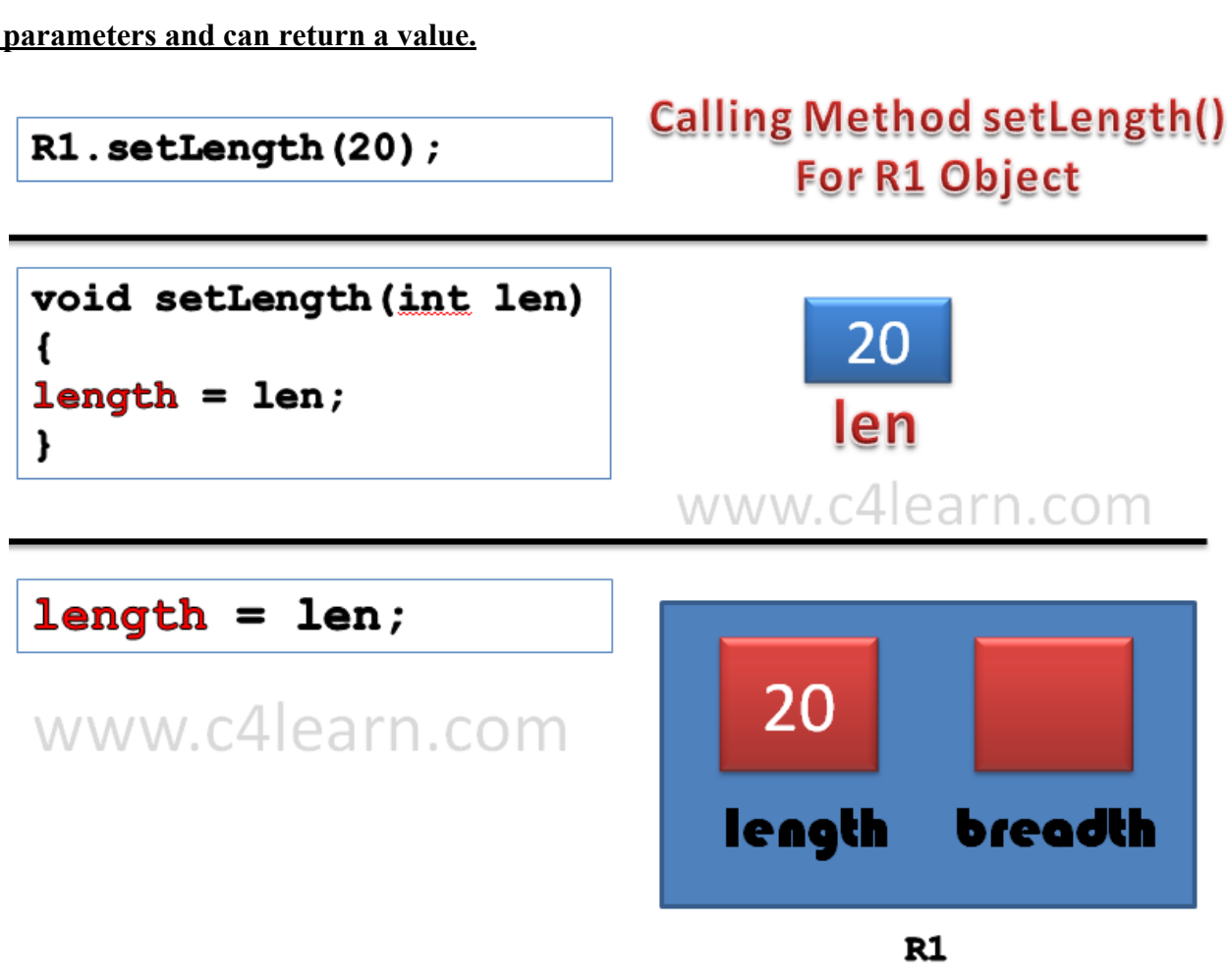
1. "**r1**" is an Object of Type Rectangle.
2. We are calling method "**setLength()**" by writing –

```
Object_Name [DOT] Method_Name ( Parameter List ) ;
```

3. Function call is always followed by **Semicolon**.

Method Definition :

1. Method Definition contain the **actual body of the method**.
2. Method **can take parameters and can return a value.**



Yet Another Constructor Example : More Detailed Example (Java Programming)

```
class Rectangle {
  int length;
  int breadth;

  Rectangle()
  {
  length  = 20;
  breadth = 10;
  }

  void setDiamentions()
  {
  length  = 40;
  breadth = 20;
  }

}

class RectangleDemo {
  public static void main(String args[]) {

  Rectangle r1 = new Rectangle();

  System.out.println("Length of Rectangle : " + r1.length);
  System.out.println("Breadth of Rectangle : " + r1.breadth);

  r1.setDiamentions();

  System.out.println("Length of Rectangle : " + r1.length);
  System.out.println("Breadth of Rectangle : " + r1.breadth);

  }
}
```

Output :

```
C:Priteshjava>java RectangleDemo
Length  of Rectangle : 20
Breadth of Rectangle : 10
Length  of Rectangle : 40
Breadth of Rectangle : 20
```

Explanation :

1. After the Creation of Object , Instance Variables have their own values inside.
2. As soon as we call method , **values are re-initialized**.

Parameterized Constructors : Constructor Taking Parameters

In this article we are talking about constructor that will take parameter. Constructor taking parameter is called as "**Parameterized Constructor**".

Parameterized Constructors :

1. Constructor Can Take Value , Value is Called as – "**Argument**".
2. Argument **can be of any type** i.e Integer,Character,Array or any Object.
3. Constructor **can take any number of Argument**.
4. See following example – How Parameterized Constructor Works ? –

Live Example : Constructor Taking Parameter in Java Programming

```
class Rectangle {
  int length;
  int breadth;

  Rectangle(int len,int bre)
  {
  length  = len;
  breadth = bre;
  }
}

class RectangleDemo {
  public static void main(String args[]) {

  Rectangle r1 = new Rectangle(20,10);

  System.out.println("Length of Rectangle : " + r1.length);
  System.out.println("Breadth of Rectangle : " + r1.breadth);

  }
}
```

Output :

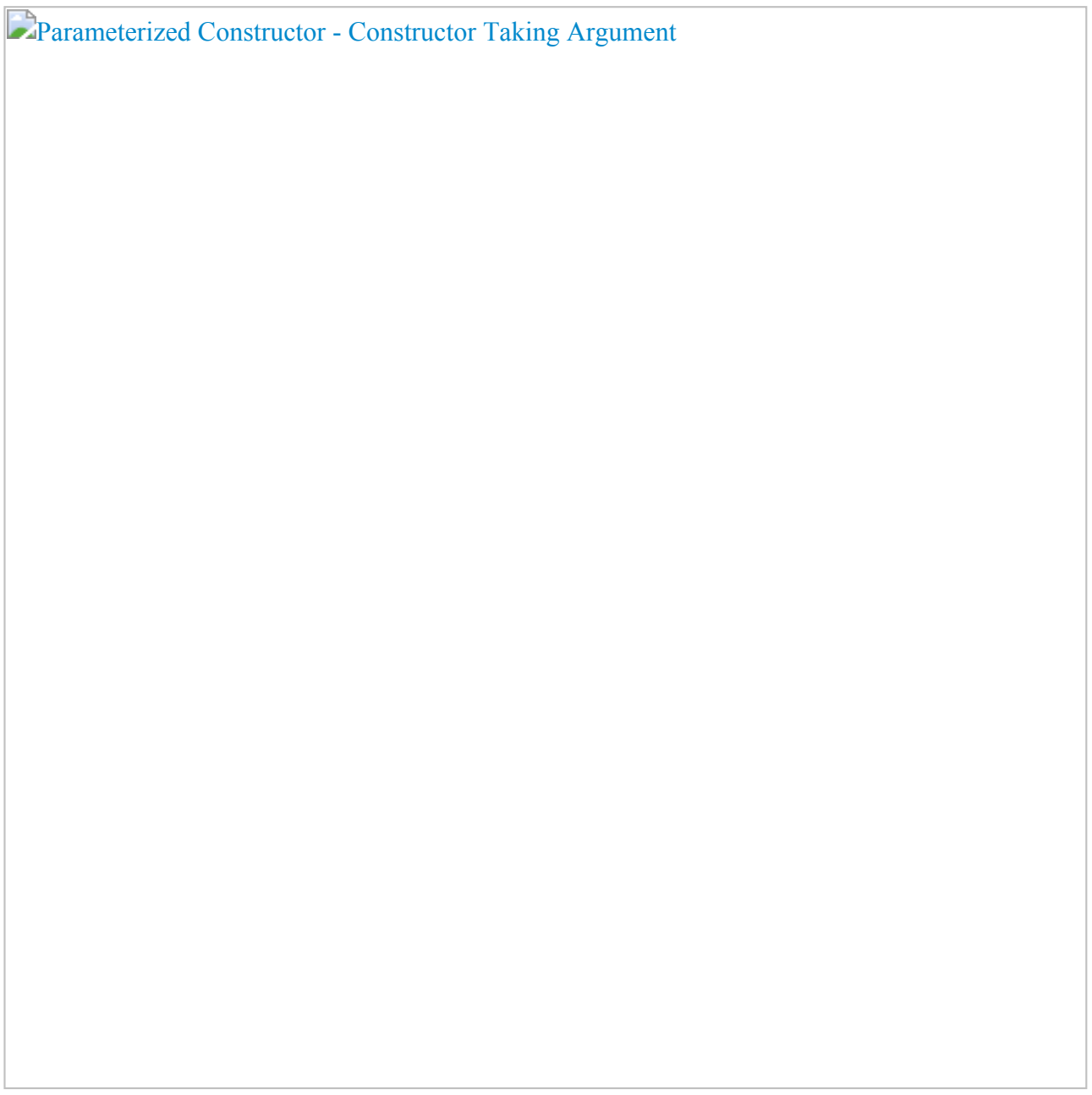
```
Length of Rectangle  : 20
Breadth of Rectangle : 10
```

Explanation :

Carefully observe above program – You will found something like this –

```
Rectangle r1 = new Rectangle(20,10);
```

This is Parameterized Constructor taking argument.These arguments are used for any purpose inside Constructor Body.



- New Operator is used to **Create Object**.
- We are passing **Parameter to Constructor** as 20,10.
- These parameters are assigned to **Instance Variables of the Class**.
- We can Write above statement like –

```
Rectangle(int length,int breadth)
  {
  length  = length;
  breadth = breadth;
  }
```

OR

```
Rectangle(int length,int breadth)
  {
  this.length  = length;
  this.breadth = breadth;
  }
```

But if we use Parameter name same as Instance variable then compiler will recognize instance variable and Parameter but user or programmer may confuse. Thus we have used "this keyword" to specify that "**Variable is Instance Variable of Object – r1**".

Passing Object as Parameter :

```
package com.pritesh.programs;

class Rectangle {
    int length;
    int width;

    Rectangle(int l, int b) {
        length = l;
        width = b;
    }

    void area(Rectangle r1) {
        int areaOfRectangle = r1.length * r1.width;
        System.out.println("Area of Rectangle : "
                           + areaOfRectangle);
    }
}

class RectangleDemo {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle(10, 20);
        r1.area(r1);
    }
}
```

Output of the program :

```
Area of Rectangle : 200
```

Explanation :

1. We can pass Object of any class as parameter to a method in java.
2. We can access the instance variables of the object passed inside the called method.

```
area = r1.length * r1.width
```

3. It is good practice to initialize instance variables of an object before passing object as parameter to method otherwise it will take default initial values.

Different Ways of Passing Object as Parameter :

Way 1 : By directly passing Object Name

```
void area(Rectangle r1) {
        int areaOfRectangle = r1.length * r1.width;
        System.out.println("Area of Rectangle : "
                           + areaOfRectangle);
    }

class RectangleDemo {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle(10, 20);
        r1.area(r1);
    }
```

Way 2 : By passing Instance Variables one by one

```
package com.pritesh.programs;

class Rectangle {
    int length;
    int width;

    void area(int length, int width) {
        int areaOfRectangle = length * width;
        System.out.println("Area of Rectangle : "
```

```
                    + areaOfRectangle);
    }
}

class RectangleDemo {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();

        r1.length = 20;
        r1.width = 10;

        r2.area(r1.length, r1.width);
    }
}
```

Actually this is not a way to pass the object to method. but this program will explain you how to pass instance variables of particular object to calling method.

Way 3 : We can pass only public data of object to the Method.

Suppose we made width variable of a class private then we cannot update value in a main method since it does not have permission to access it.

```
private int width;
```

after making width private –

```
class RectangleDemo {
    public static void main(String args[]) {
        Rectangle r1 = new Rectangle();
        Rectangle r2 = new Rectangle();

        r1.length = 20;
        r1.width = 10;

        r2.area(r1.length, r1.width);
    }
}
```

Returning the Object From Method

In Java Programming A method can return any type of data, including class types that you create.

For example, in the following program, the **getRectangleObject( )** method returns an object.

Java Program : Returning the Object From Method

```
package com.pritesh.programs;

import java.io.File;
import java.io.IOException;

class Rectangle {
      int length;
      int breadth;

      Rectangle(int l,int b) {
        length = l;
        breadth = b;
      }

      Rectangle getRectangleObject() {
        Rectangle rect = new Rectangle(10,20);
        return rect;
      }
}

class RetOb {
      public static void main(String args[]) {
        Rectangle ob1 = new Rectangle(40,50);
        Rectangle ob2;

        ob2 = ob1.getRectangleObject();
        System.out.println("ob1.length : " + ob1.length);
        System.out.println("ob1.breadth: " + ob1.breadth);

        System.out.println("ob2.length : " + ob2.length);
        System.out.println("ob2.breadth: " + ob2.breadth);

        }
}
```

Output of the Program :

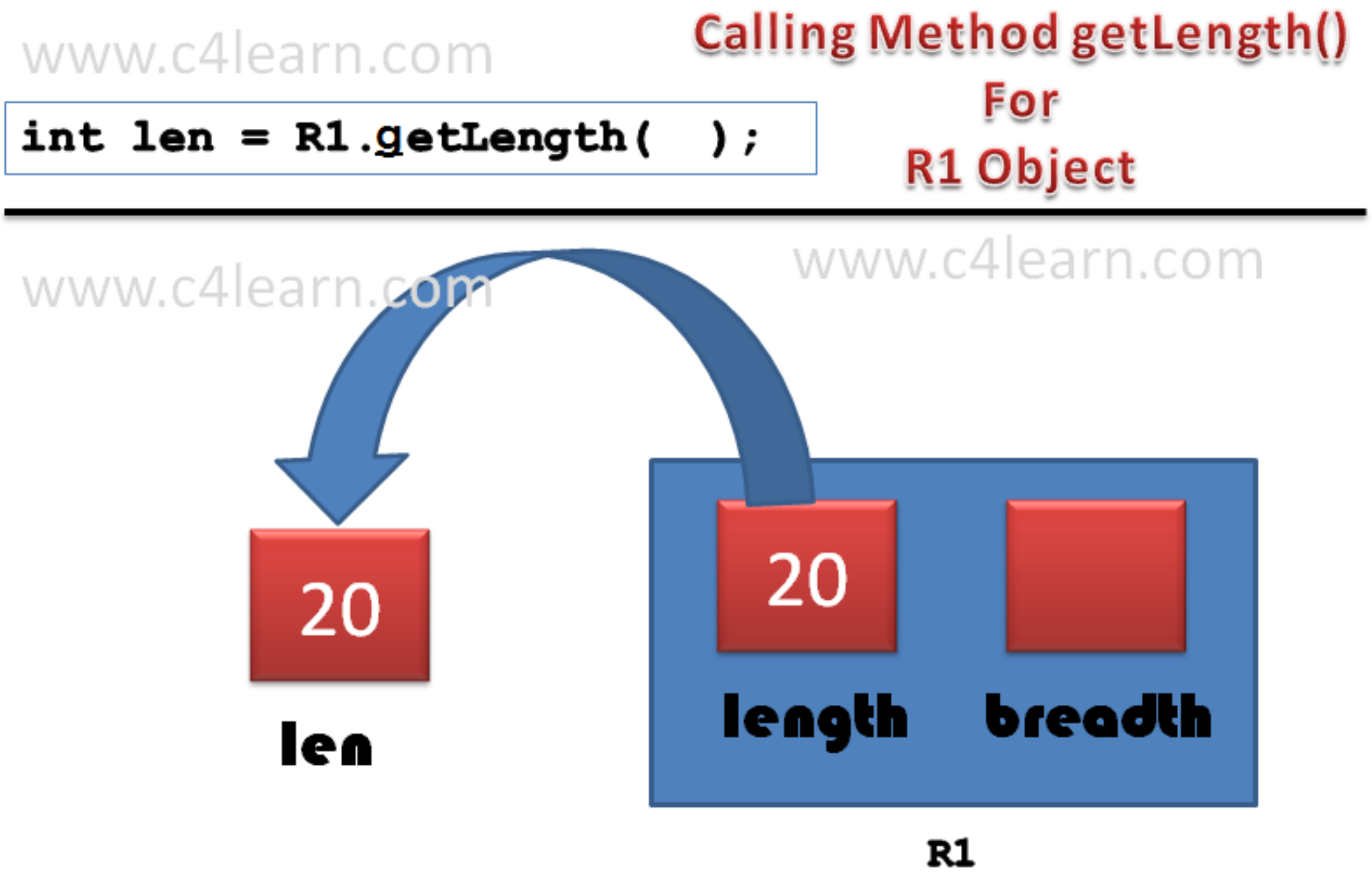
```
ob1.length : 40
ob1.breadth: 50
ob2.length : 10
ob2.breadth: 20
```

Explanation :

1. In the above program we have called a method **getRectangleObject**() and the method creates object of class from which it has been called.
2. All objects are dynamically allocated using **new**, you don’t need to worry about an object going out-of-scope because the method in which it was created terminates.
3. The object will continue to exist as long as there is a reference to it somewhere in your program. When there are no references to it, the object will be reclaimed the next time garbage collection takes place.

Returning Value From the Method :

1. We can specify return type of the method as **“Primitive Data Type” or “Class name”**.
2. Return Type can be “Void” means **it does not return any value**.
3. Method can return a value by using “**return**” keyword.



Live Example : Returning Value from the Method

```
class Rectangle {
  int length;
  int breadth;

  void setLength(int len)
  {
  length = len;
  }

  int getLength()
  {
  return length;
  }

}

class RectangleDemo {
  public static void main(String args[]) {

  Rectangle r1 = new Rectangle();

  r1.setLength(20);

  int len = r1.getLength();

  System.out.println("Length of Rectangle : " + len);

  }
}
```

Output :

```
C:Priteshjava>java RectangleDemo
Length of Rectangle : 20
```

There are two important things to understand about returning values :

1. The **type of data returned by a method must be compatible with the return type specified by the method**. For example, if the return type of some method is boolean, you could not return an integer.

```
boolean getLength()
  {
  int length = 10;
  return(length);
  }
```

2. **The variable receiving the value returned by a method (such as len, in this case) must also be compatible with the return type specified for the method**.

```
int getLength()
  {
  return length;
  }

boolean len = r1.getLength();
```

3. **Parameters should be passed in sequence and they must be accepted by method in the same sequence**.

```
void setParameters(String str,int len)
  {
  -----
  -----
  -----
  }

 r1.setParameters(12,"Pritesh");
```

Instead it should be like –

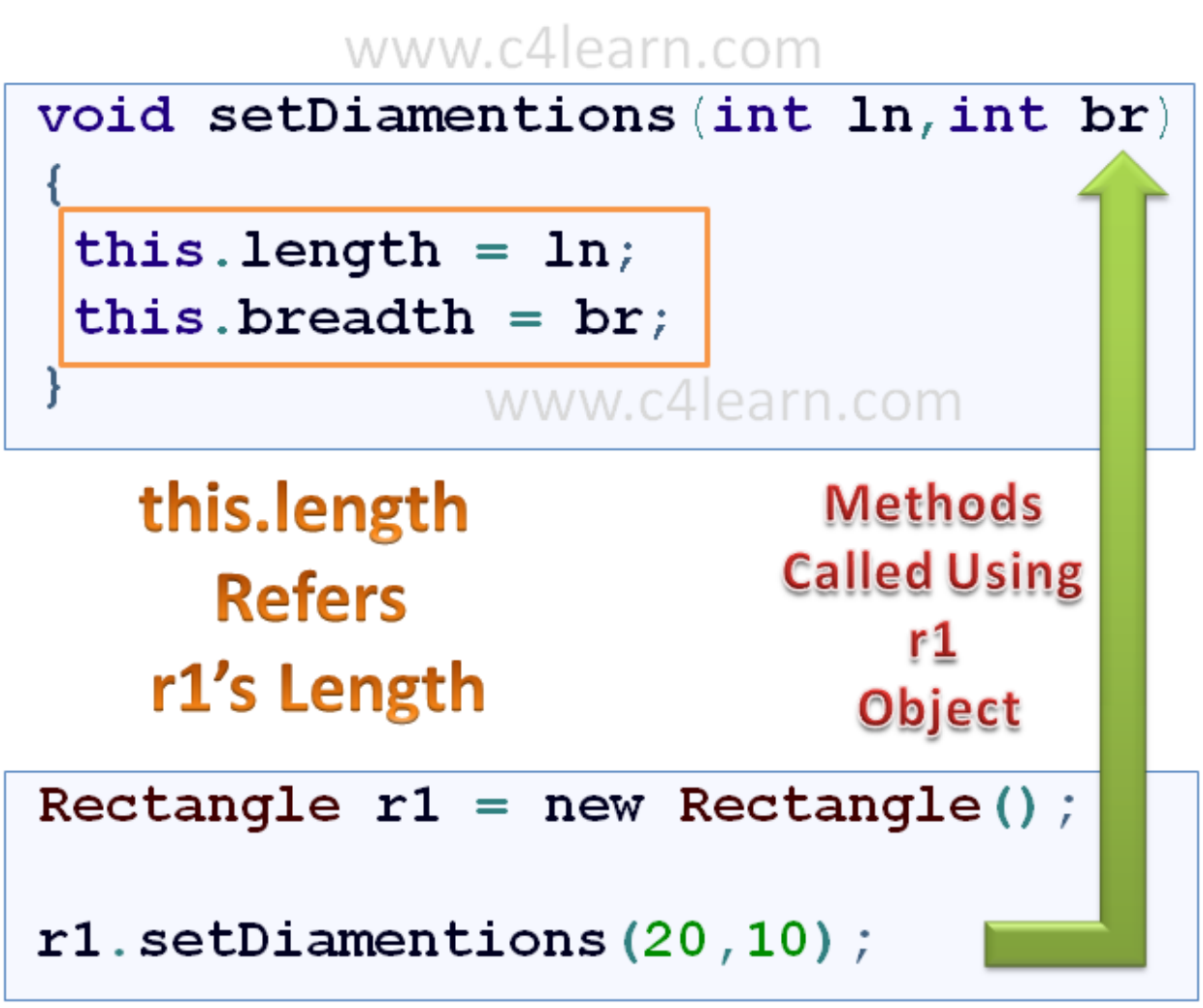
```
void setParameters(int length,String str)
  {
  -----
  -----
  -----
  }

 r1.setParameters(12,"Pritesh");
```

this keyword : Refer Current Object in Java Programming

1. **this** is keyword in Java.
2. We can use this keyword in any method or constructor.
3. this keyword used to **refer current object**.
4. Use **this keyword** from any method or constructor to refer to the **current object that calls a method or invokes constructor** .

Syntax : this Keyword

```
this.field
```



Live Example : this Keyword

```
class Rectangle {
  int length;
  int breadth;

  void setDiamentions(int ln,int br)
  {
  this.length  = ln;
  this.breadth = br;
  }

}

class RectangleDemo {
  public static void main(String args[]) {

  Rectangle r1 = new Rectangle();

  r1.setDiamentions(20,10);

  System.out.println("Length of Rectangle : " + r1.length);
  System.out.println("Breadth of Rectangle : " + r1.breadth);

  }
}
```

Output :

```
C:Priteshjava>java RectangleDemo
Length  of Rectangle : 20
Breadth of Rectangle : 10
```

this Keyword is used to hide Instance Variable :

```
void setDiamentions(int length,int breadth)
  {
  this.length  = length;
  this.breadth = breadth;
  }
```

- length,breadth are the **parameters that are passed to the method**.
- Same names are given to the **instance variables of an object**.
- In order to hide instance variable we can use this keyword. above syntax will clearly make**difference between instance variable and parameter**.

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups −

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators −

Assume integer variable A holds 10 and variable B holds 20, then −

Show Examples

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

The Relational Operators

There are following relational operators supported by Java language.

Assume variable A holds 10 and variable B holds 20, then −

Show Examples

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Bitwise Operators

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows −

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators −

Assume integer variable A holds 60 and variable B holds 13 then −

Show Examples

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000

>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators

The following table lists the logical operators −

Assume Boolean variables A holds true and variable B holds false, then −

Show Examples

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

The Assignment Operators

Following are the assignment operators supported by Java language −

Show Examples

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Bitwise AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Right shift AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2

|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Miscellaneous Operators

There are few other operators supported by Java Language.

Conditional Operator (? :)

Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as −

```
variable x = (expression) ? value if true : va
```

Following is an example −

Example

```
public class Test {

   public static void main(String args[]) {
      int a, b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is :

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is :
   }
}
```

This will produce the following result −

Output

```
Value of b is : 30
Value of b is : 20
```

instanceof Operator

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as −

```
( Object reference variable ) instanceof  (cla
```

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is an example −

Example

```
public class Test {

   public static void main(String args[]) {

      String name = "James";

      // following will return true since n
      boolean result = name instanceof Stri
      System.out.println( result );
   }
}
```

This will produce the following result −

Output

```
true
```

This operator will still return true, if the object being compared is the assignment compatible with the type on the right. Following is one more example −

Example

```
class Vehicle {}

public class Car extends Vehicle {

   public static void main(String args[]) {

      Vehicle a = new Car();
      boolean result =  a instanceof Car;
      System.out.println( result );
   }
}
```

This will produce the following result −

Output

```
true
```

Precedence of Java Operators

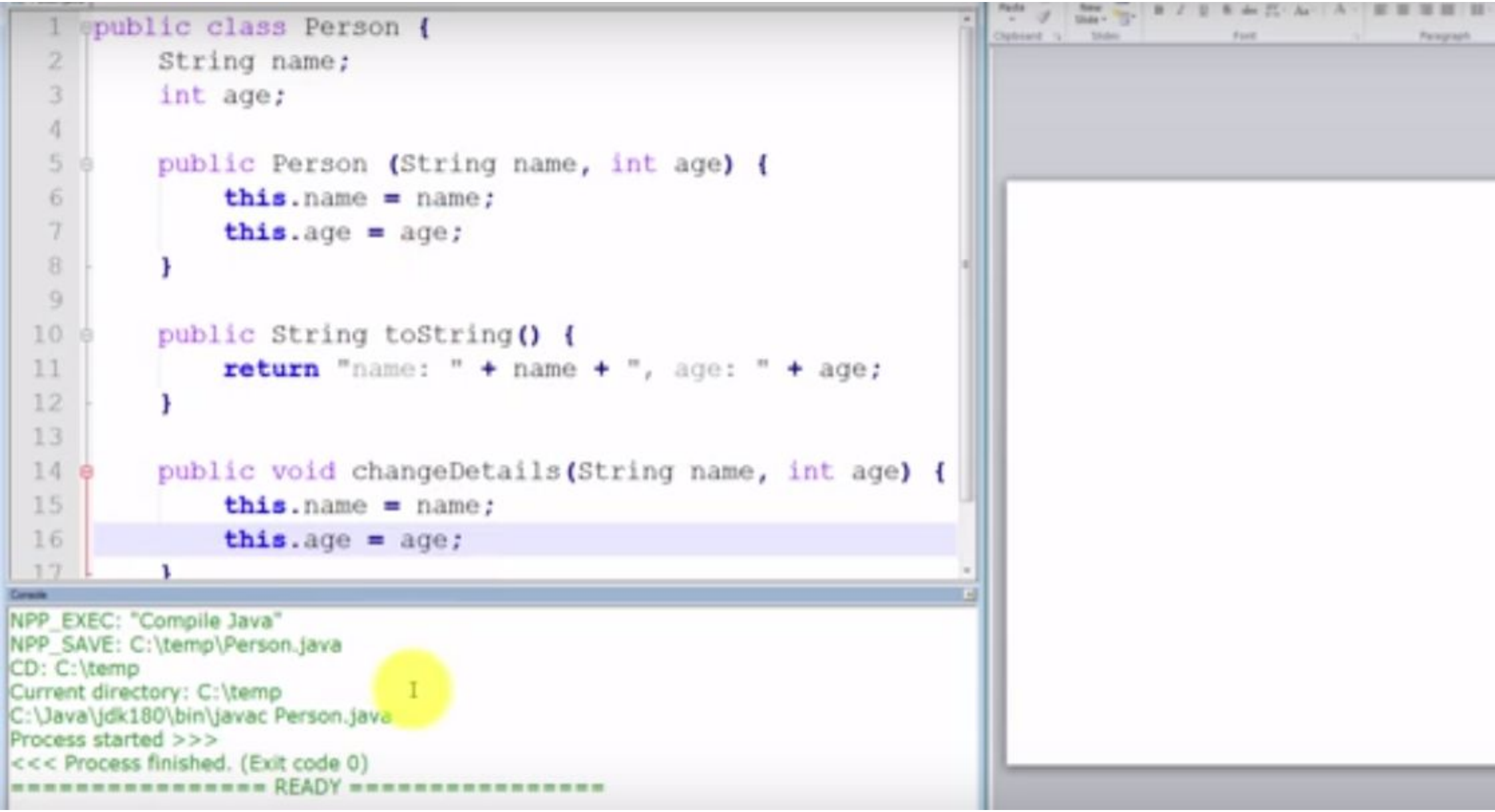
Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator −

For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3 * 2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

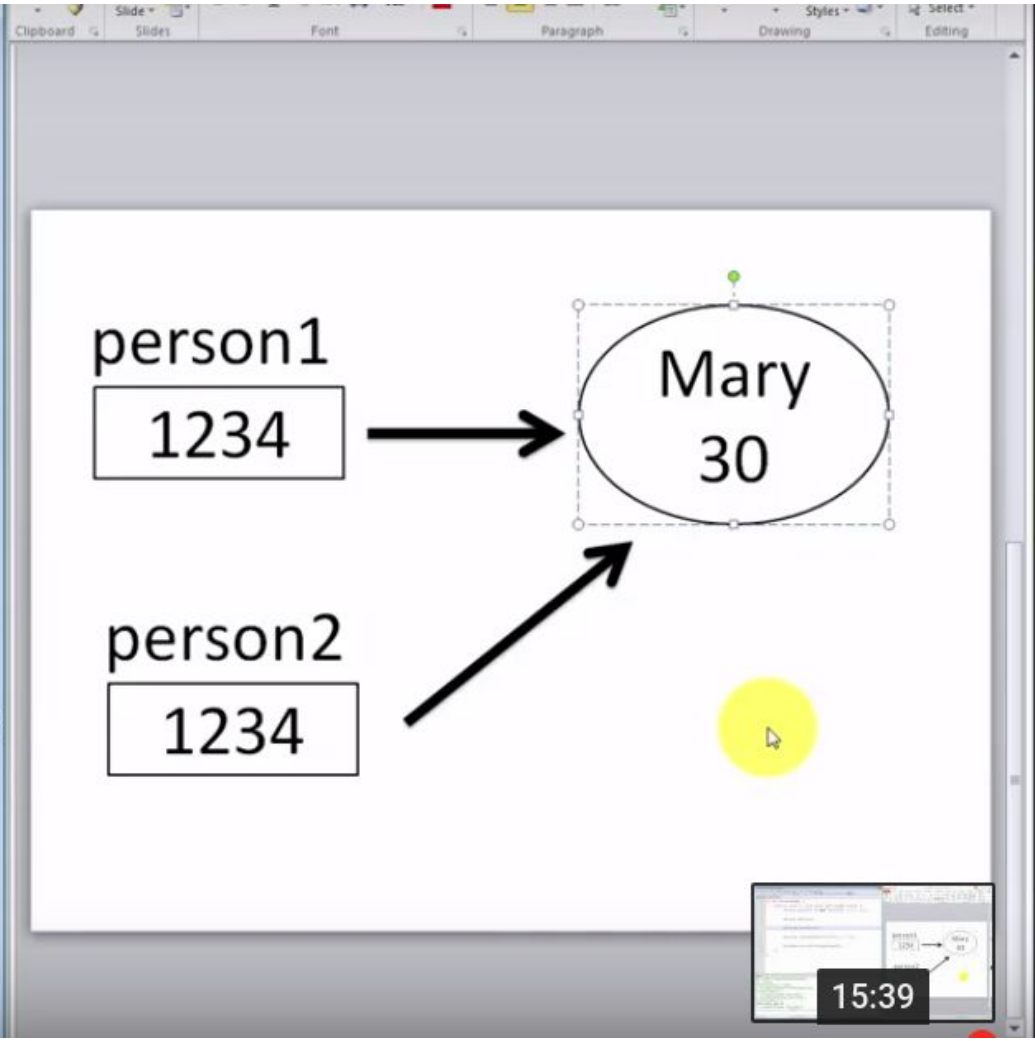
Category	Operator	Associativity

Postfix	>() [] . (dot operator)	Left toright
Unary	>++ - - ! ~	Right to left
Multiplicative	>* /	Left to right
Additive	>+ -	Left to right
Shift	>>> >>> <<	Left to right
Relational	>> >= < <=	Left to right
Equality	>== !=	Left to right
Bitwise AND	>&	Left to right
Bitwise XOR	>^	Left to right
Bitwise OR	>|	Left to right
Logical AND	>&&	Left to right
Logical OR	>||	Left to right
Conditional	?:	Right to left
Assignment	>= += -= *= /= %= >>= <<= &= ^= |=	Right to left


1 public class Person {
2 String name;
3 int age;
4
5 public Person (String name, int age) {
6 this.name = name;
7 this.age = age;
8 }
9
10 public String toString() {
11 return "name: " + name + ", age: " + age;
12 }
13
14 public void changeDetails(String name, int age) {
15 this.name = name;
16 this.age = age;
17 }
NPP_EXEC: "Compile Java"
NPP_SAVE: C:\temp\Person.java
CD: C:\temp
Current directory: C:\temp
C:\Java\jdk180\bin\javac Person.java
Process started >>>
<<< Process finished. (Exit code 0)
================ READY ================

```java
class PersonDemo {
    public static void main (String[] args) {
        Person person1 = new Person("John", 21);

        Person person2;

        person2 = person1;

        person2.changeDetails("Mary", 30);

        System.out.println(person1);
    }
}
```

```
Console
NPP_SAVE: C:\temp\PersonDemo.java
CD: C:\temp
Current directory: C:\temp
C:\Java\jdk180\bin\javac PersonDemo.java
Process started >>>
<<< Process finished. (Exit code 0)
C:\Java\jdk180\bin\java PersonDemo
Process started >>>
name: Mary, age: 30
<<< Process finished. (Exit code 0)
```



Defining a method with an object as a parameter

- When defining a method with a parameter specify the *data type* of each parameter.
- In this case, the *type* of the parameter is a Rectangle

```
public static void displayRectangle (Rectangle rect)
{
        System.out.println(rect.getLength());
        System.out.println(rect.getWidth());
}
```

- **rect** can call all of the methods defined in the Rectangle class.

Calling a method and passing an object

- When calling a method put the name of the object within the parenthesis

```
public static void main(String [] args)
{
    //Create a Rectangle object named r1
    Rectangle r1 = new Rectangle (5,7);

    //Call the displayRectangle method
    displayRectangle(r1);
}
```

Main Idea

- When an object is passed to a method, the memory address is being passed, not the data stored in it.

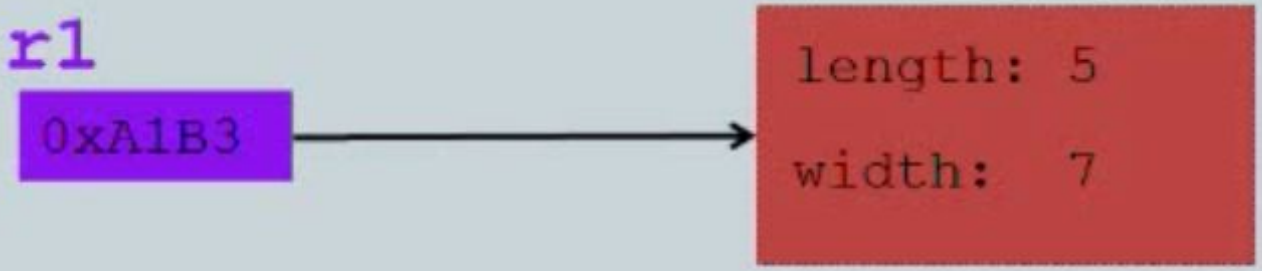
Passing Objects as Arguments to Methods - Blanton

Declaring an Object

- Create an object of a Rectangle class

```
Rectangle r1 = new Rectangle (5,7);
```

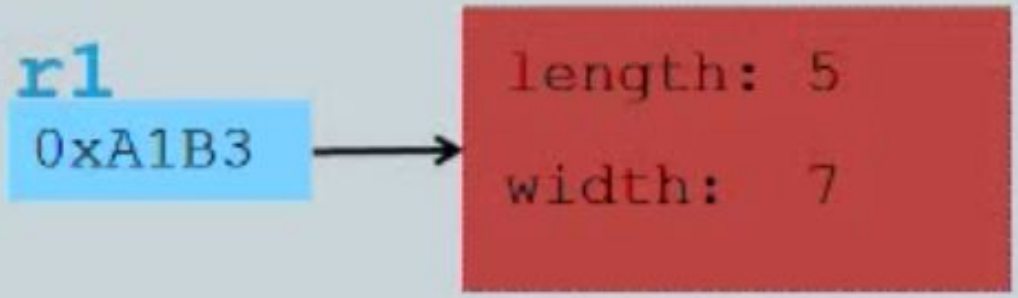
- **r1** holds the memory address of the object



Passing an object as a parameter

- The *address* stored in the argument r1 is sent to the parameter, rect

displayRectangle(r1);



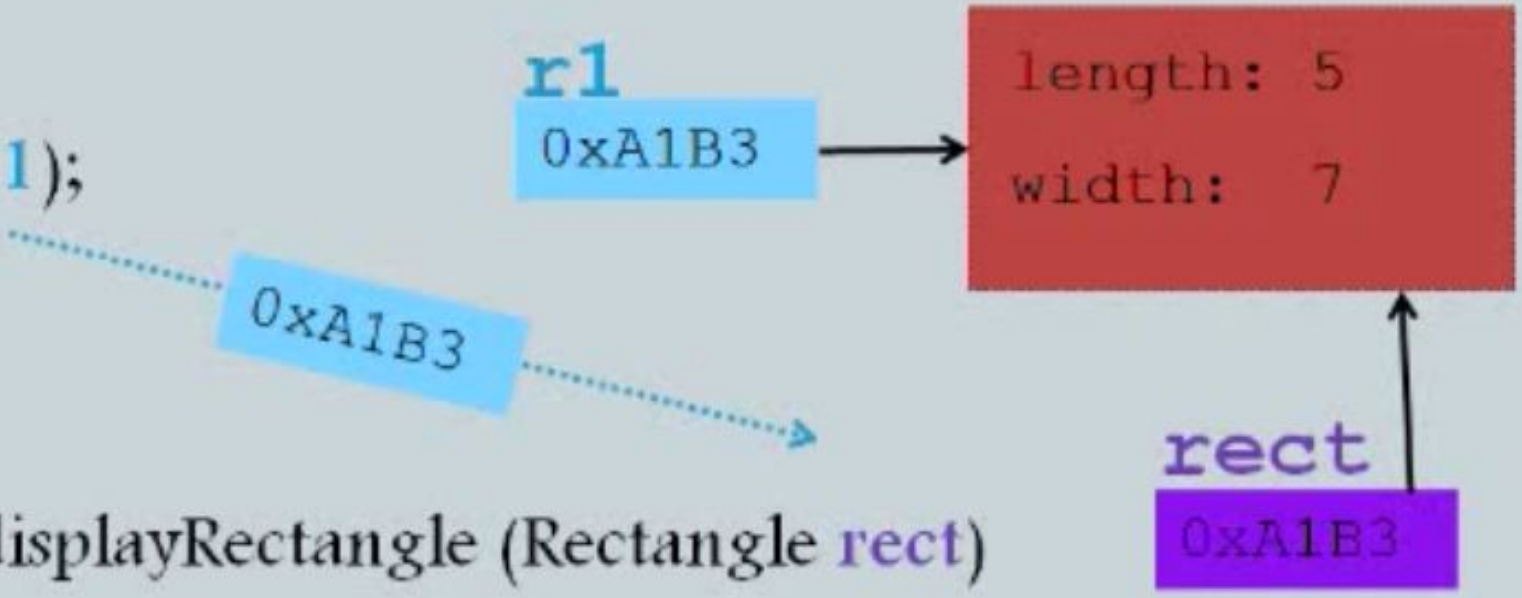

```
public static void displayRectangle (Rectangle rect)
```

- Both r1 and rect refer to the same Rectangle object

Passing an object as a parameter

- The *addresses*stored in the argument r1 is sent to the parameter, rect

displayRectangle(r1);



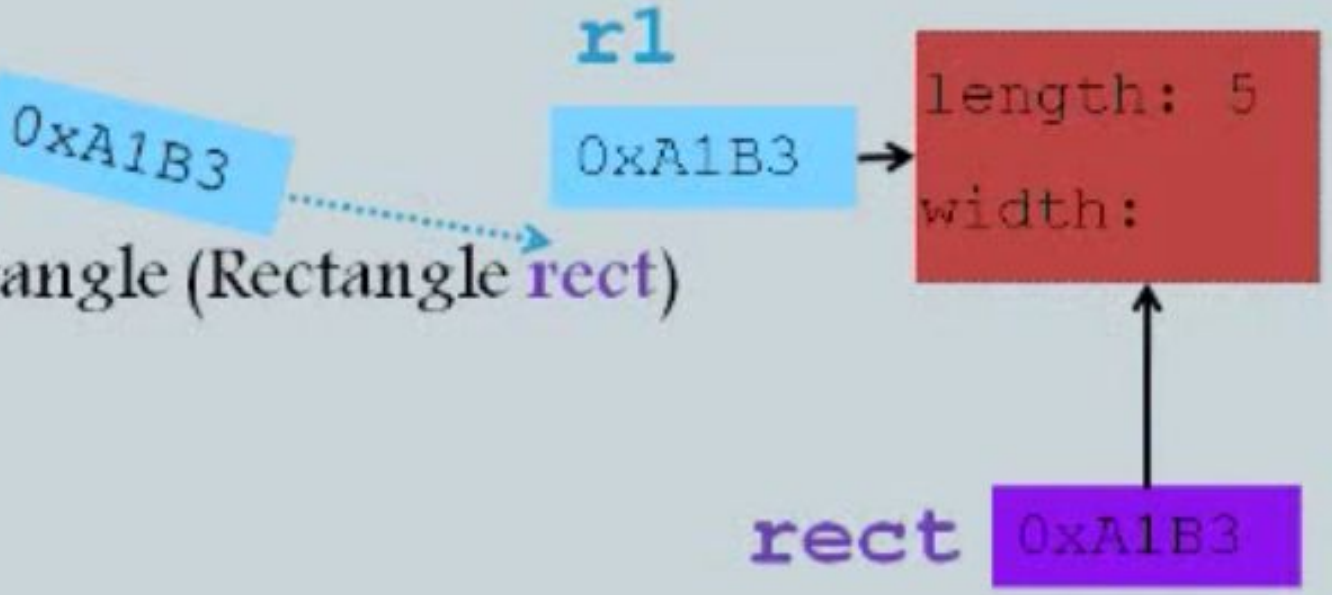
public static void displayRectangle (Rectangle rect)

- Both r1 and rect refer to the same Rectangle object

Passing an object as a parameter

- When a method receives an object reference as an argument, it may modify the contents of the object

```
setRectangle(r1);

public static void setRectangle (Rectangle rect)
{
    rect.setLength(6);
    rect.setWidth(3);
}
```

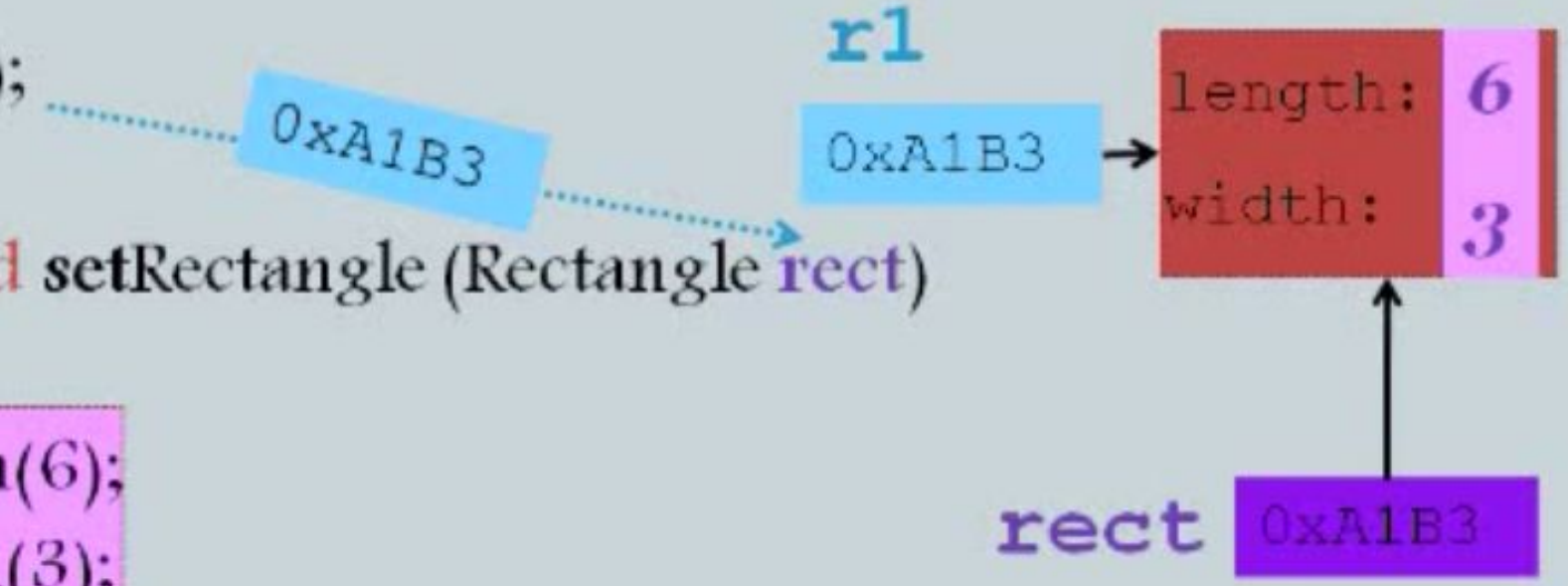


- Changes made to the object referenced by rect will also update r1

Passing an object as a parameter

- When a method receives an object reference as an argument, it may modify the contents of the object

```
setRectangle(r1);

public static void setRectangle (Rectangle rect)
{
    rect.setLength(6);
    rect.setWidth(3);
}
```

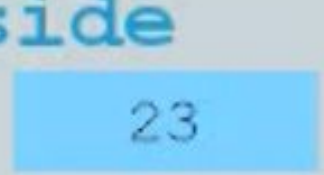


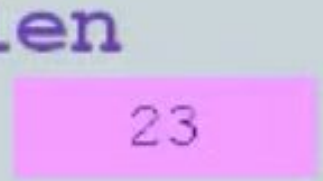
- Changes made to the object referenced by rect will also update r1

Passing Primitive Data Types

- Changing a parameter of a primitive data type will *not* change the argument.

```
public static void main (String [] args)
{
   double side= 23;
   showSquare(side) ;
}

public static void showSquare   (double len)
{
   len = len*len;
   System.out.println(len + "sq ft");
}
```

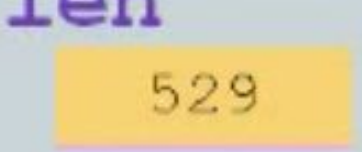




Passing Primitive Data Types

- Changing a parameter of a primitive data type will *not* change the argument.

```
public static void main (String [] args)
{
   double side= 23;
   showSquare(side) ;
}

public static void showSquare   (double len)
{
   len = len*len;
   System.out.println(len + "sq ft");
}
```



Key Points

- It is useful to pass an object as an argument to a method because when doing so you pass the memory address to the method instead a copy of a value. This means that you can update the object in another location and it will apply to the method.
- When writing a method that receives an object as an argument, do not accidentally write code that modifies the object.