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 –

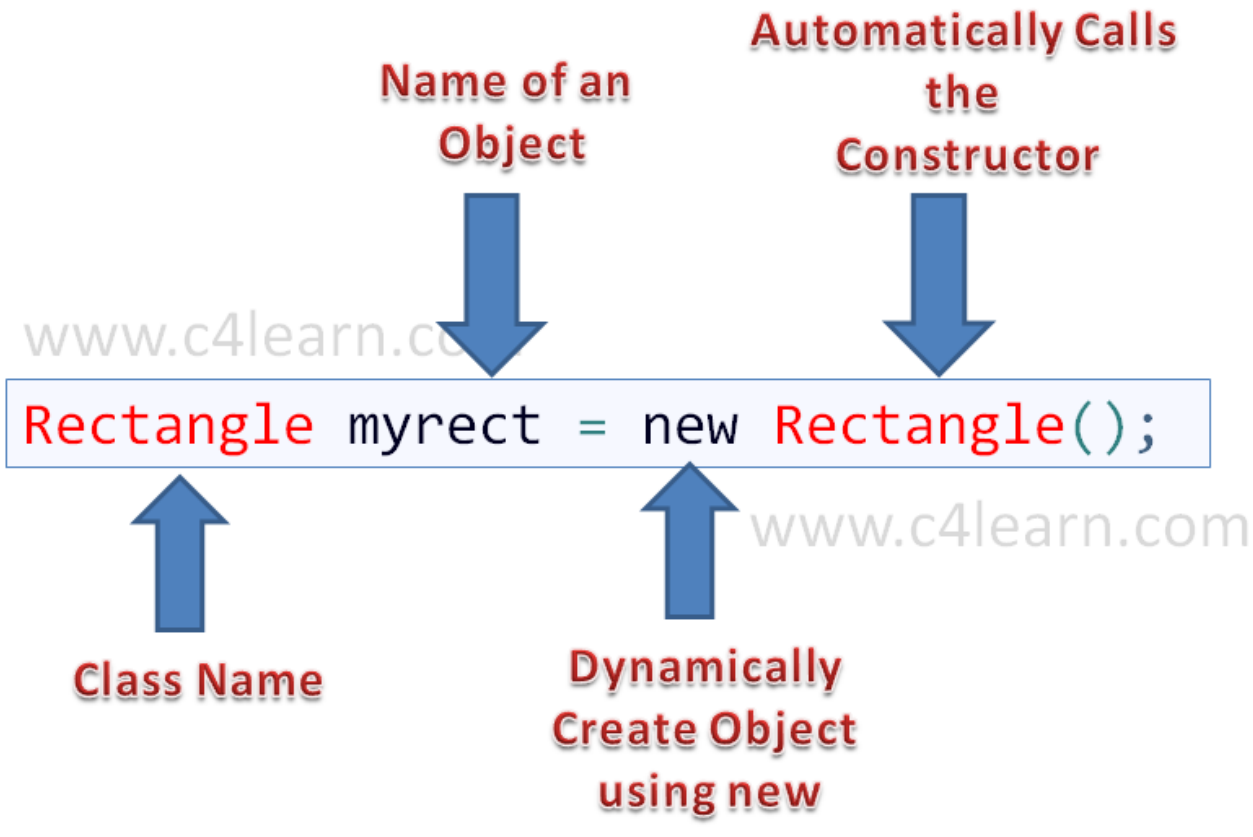
```
class Rectangle {
  double length;
  double breadth;
}
```

- **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).

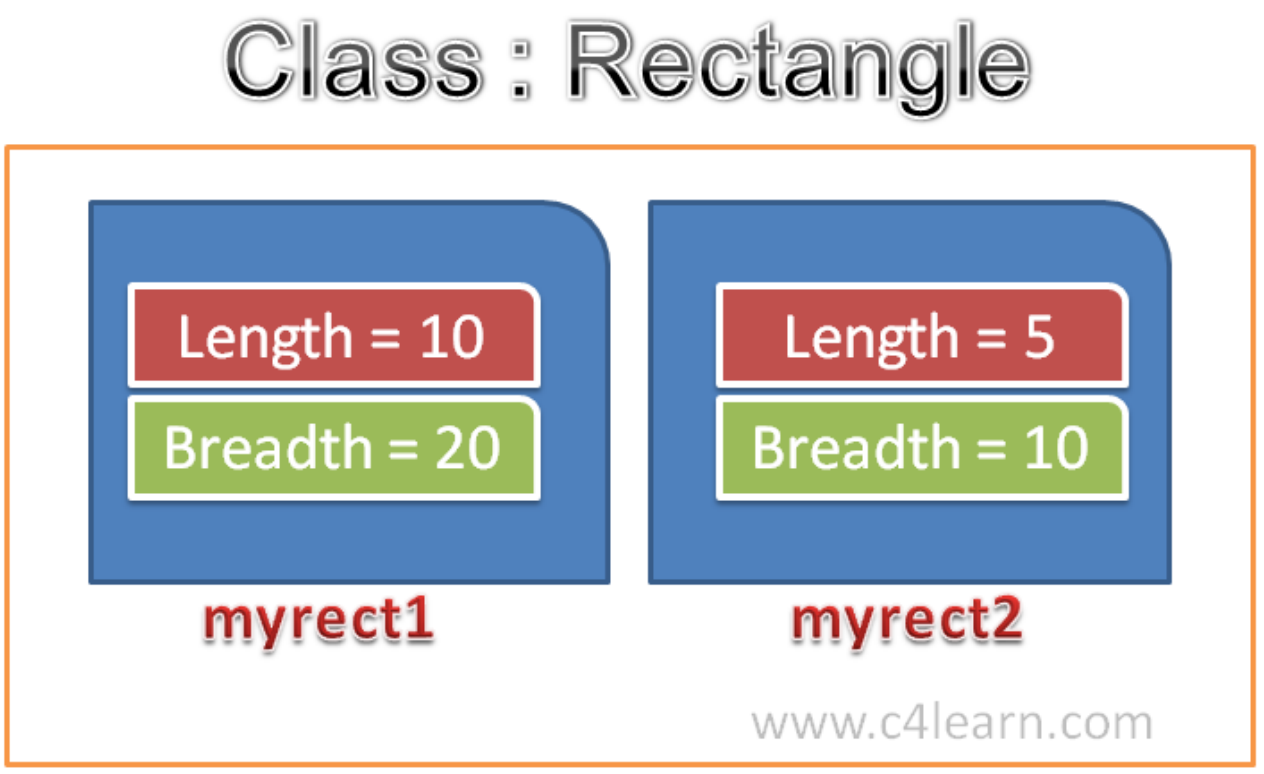
[468×60]

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-

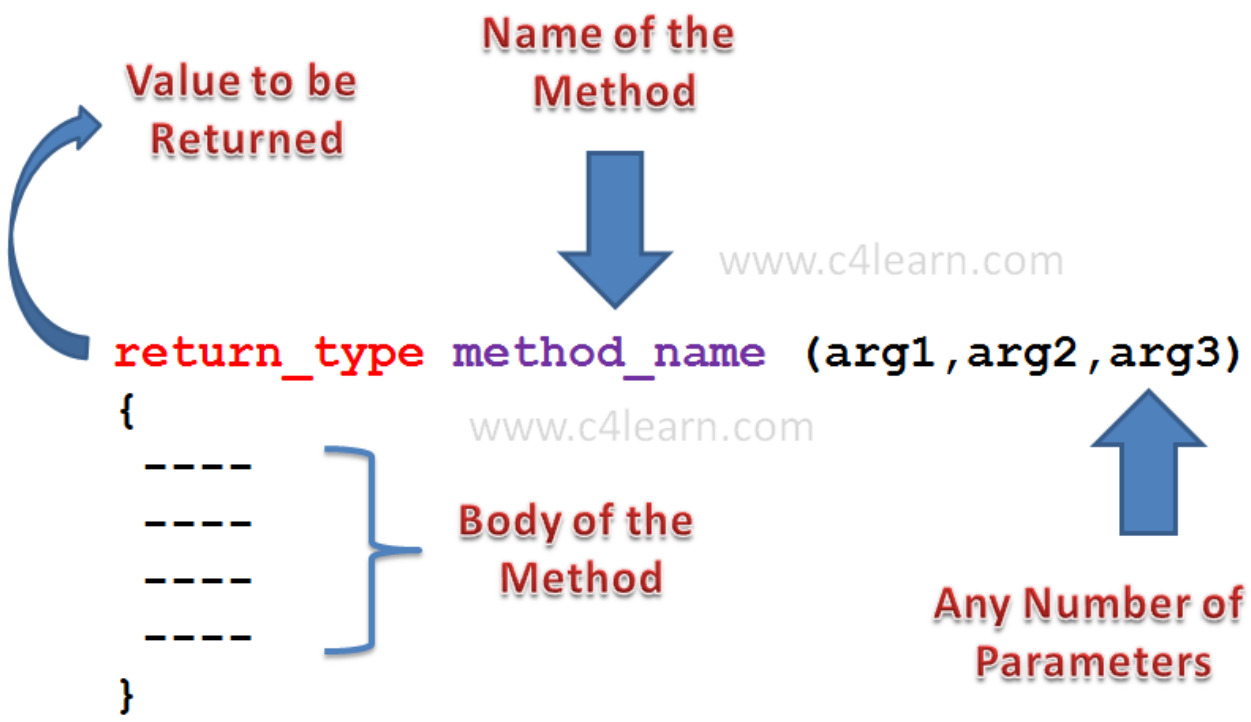
```
Rectangle()   //This is Constructor
  {
  length  = 20;
  breadth = 10;
  }
```

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 :

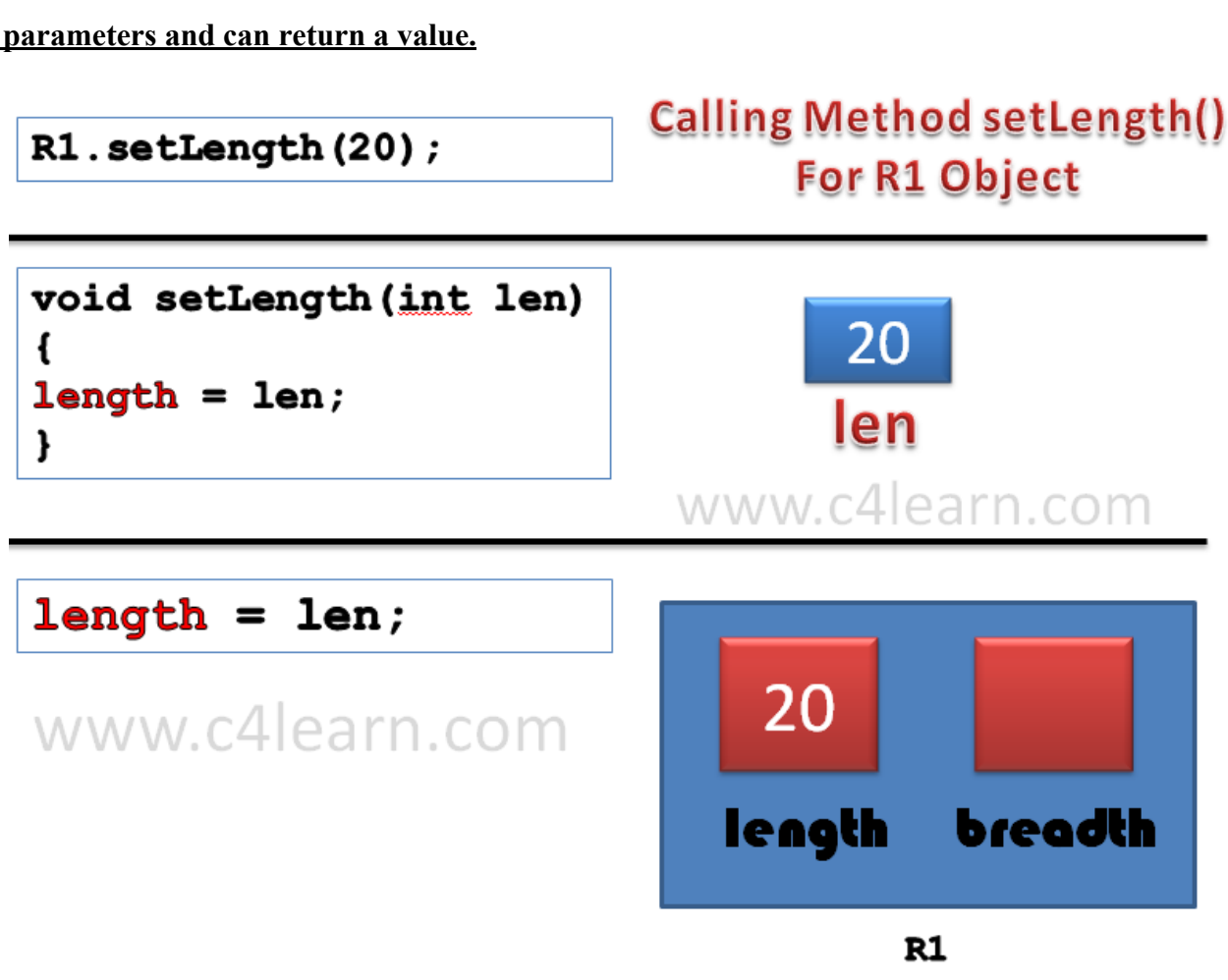
1. "**r1**" is an Object of Type Rectangle.
2. We are calling method "**setLength()**" by writing –

```
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 :

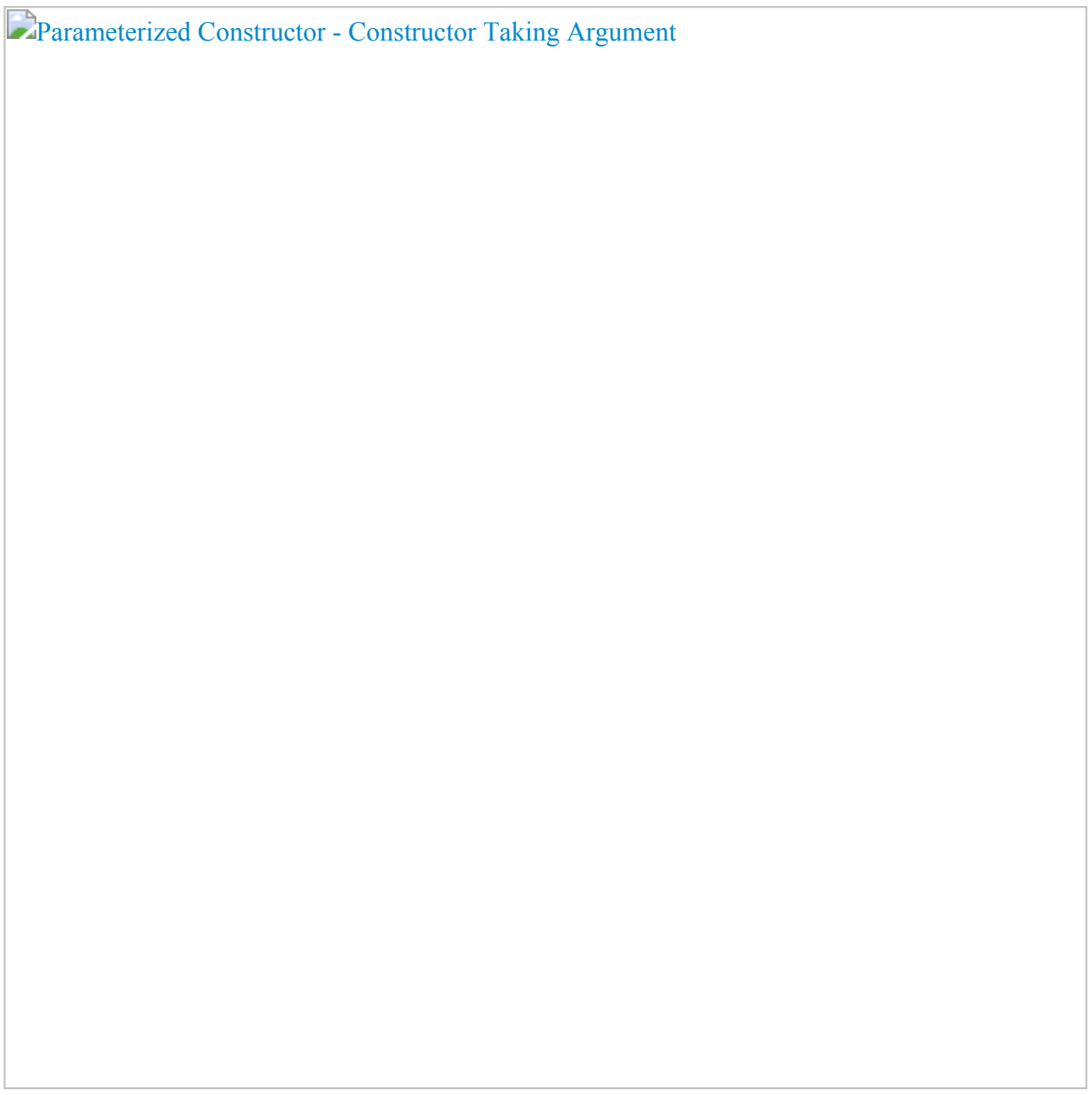
```
Length of Rectangle  : 20
Breadth of Rectangle : 10
```

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 :

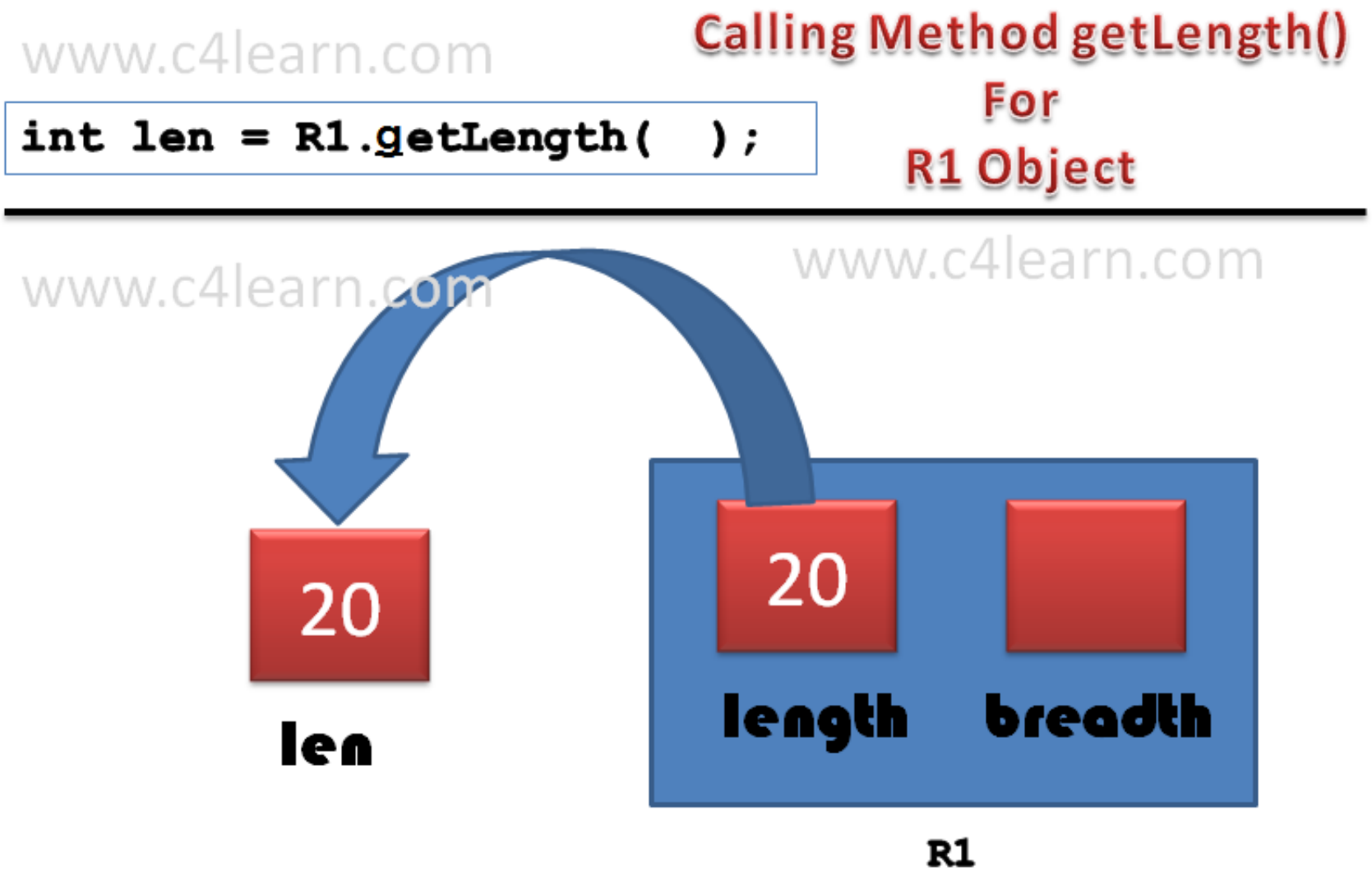
```
ob1.length : 40
ob1.breadth: 50
ob2.length : 10
ob2.breadth: 20
```

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 –

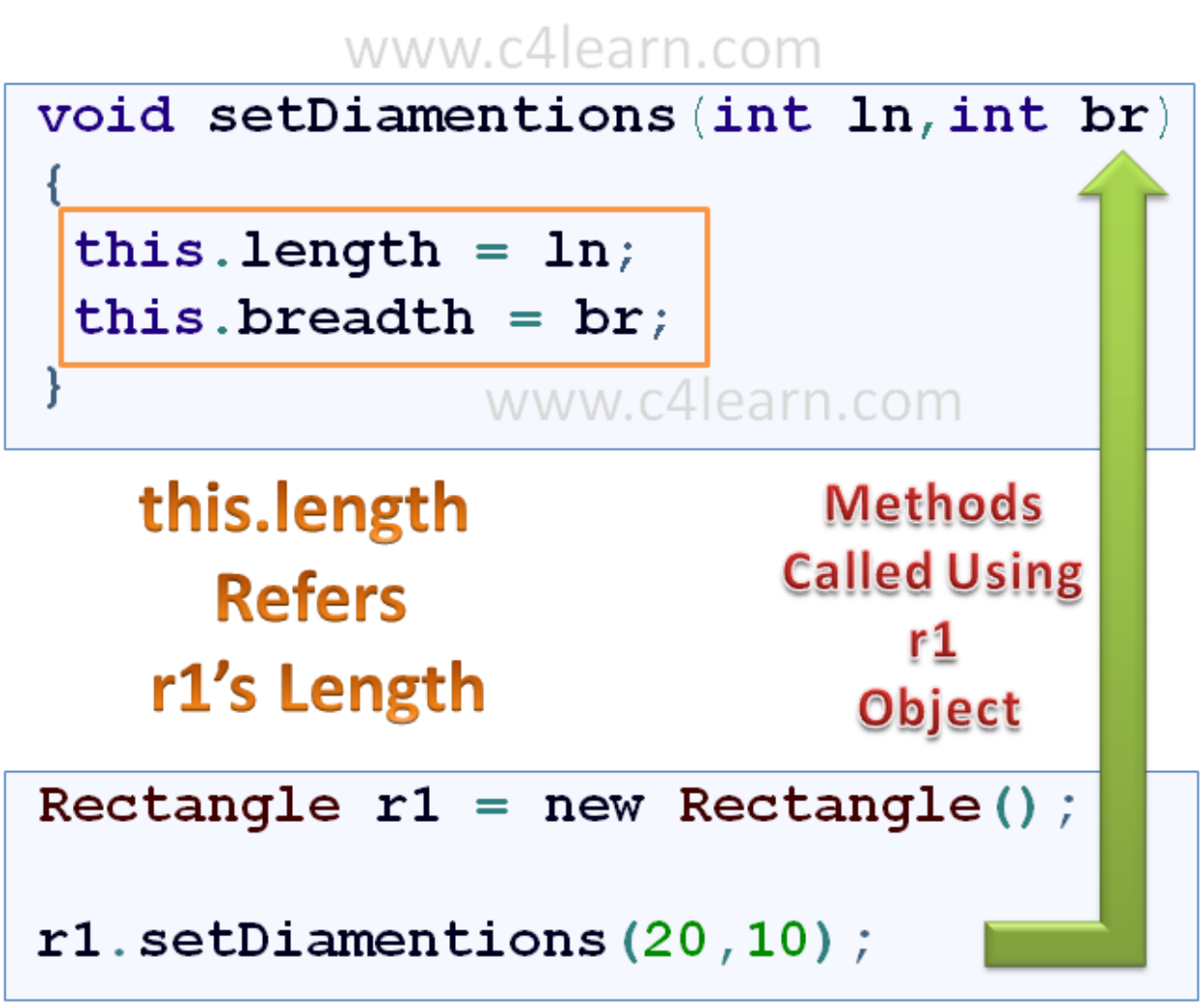
```
void setParameters(int length,String str)
  {
  -----
  -----
  -----
  }

 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**.