Everyone is talking about Enterprise JavaBeans (EJBs). Perhaps your business is planning to implement an EJB application, or perhaps you want to discover more about this technology for personal enrichment. Whatever your reason, you're about to discover that there's a lot more to EJB programming than writing code.

To be a successful EJB programmer, you must first understand the conceptual model of the Enterprise JavaBean architecture. The*architecture* is the conceptual model that structures EJB applications and ensures that different parts of the application can work together. Understanding the EJB architecture is important because, as an EJB developer, you must adhere to certain development practices in order to ensure that the EJB application works properly. Your responsibilities are referred to as the *EJB developer contract*.

EJBs are software components. A *software component* is a program that runs inside a container. (What's a container? Read on!) The component provides some unique functionality that is specific to the application you develop, whether it's a shopping cart for an online retailer or an account management service for a bank. The *container* (See?) provides your software components with *system services*. System services are generic services that any type of application can benefit from, such as security and transaction services. Basically, that means you can benefit from many very powerful system features in your software components without writing any code to create those features.

The EJBs you develop in your EJB application should provide services that are unique and special to the business problems your software needs to address. If your EJB components don't address a unique problem, then you don't necessarily need to develop them yourself; you can probably buy existing components that do the job.

Now for the catch. (You never get somethin' for nothin'.) In the case of EJBs, in order to benefit from any container's services, you — as the EJB developer — must adhere to a*contract* with the container. The container agrees to provide certain features to your EJBs according to a specified set of rules. In exchange, you must develop your EJBs to conform to a specified structure that the EJB container can understand.

Think of this component concept in the same way that you might think of your stereo system. If you're an audiophile, you have the ability to choose between a variety of brands for the different components for your stereo system. You may get one brand of receiver, another brand of amplifier, and yet another brand of speakers. You can plug them all together because each component adheres to a convention that requires consistent interfaces for plugging into a stereo system. Likewise, Enterprise JavaBeans can be added and removed from any EJB container because the EJB specification requires consistent interfaces between the container and the EJB component.

Figure 1 illustrates a simple view of the component model for Enterprise JavaBeans.

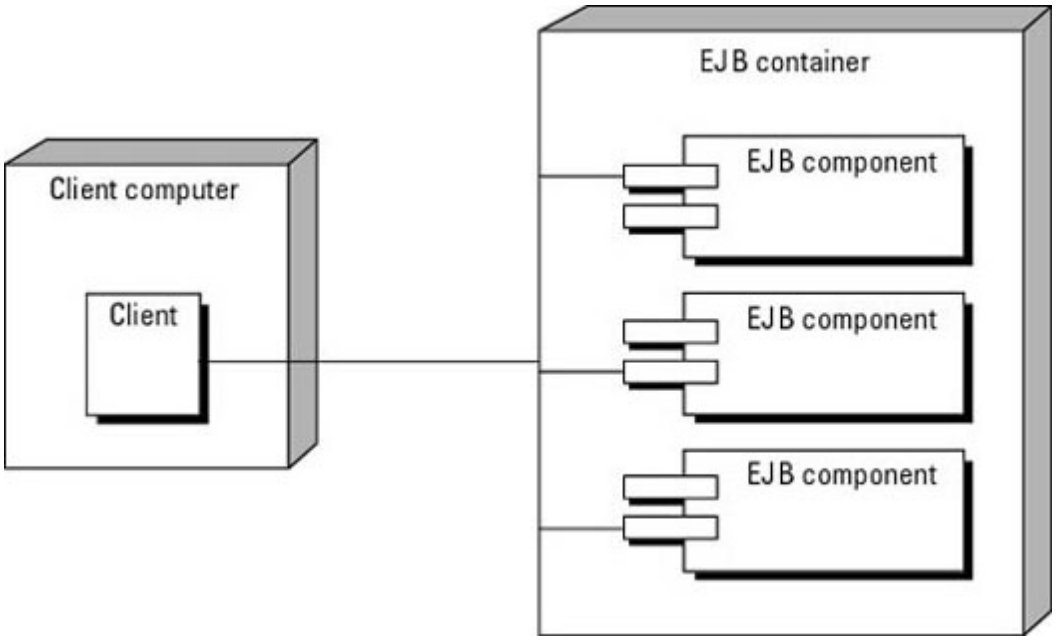

Figure 1: The component view of an EJB application.

The figure shows the following three key players in an EJB application:

- **The *client* is a software application that makes use of EJB components.** The client can reside on the same computer as the EJB component, or it can reside on a remote computer. The client can also be virtually any type of application. You may have a JavaServer Pages (JSP)

application as a client, or a desktop application residing on a user's computer. The client may also be another Enterprise JavaBean.

- **The container is the host for EJB components.** It provides a variety of system services to the EJB component so you don't have to develop them yourself. When a client application — such as a JSP application — invokes a method on an EJB component, the call is passed through the EJB container first. The container performs these extra services and then passes the client's call to the EJB component. Ultimately, the EJB component performs the operations requested by the client. This entire process is completely transparent to the client application; as far as the client is concerned, it thinks it's talking directly to an EJB component.

- **The EJB component is a provider of business services or business data.** *Business services* and *business data* are processes andinformation that*you* define and which are specific to the needs of your business. As an EJB component developer, your development responsibilities are twofold:

• **Your EJB components must implement the methods required by the EJB component architecture.** These methods are collectively referred to as the *application programming interface* (API). The methods defined in the API allow the EJB container to provide system services to your EJB components. They also allow you to make requests to the container to perform certain actions, such as getting the identity of a user.

• **You must implement the business methods required for the application you're developing.** That allows the client to receive business services and business data from your EJB component. For example, if you're developing a shopping cart application for your business, you'll need to define methods to add items to the cart and remove items from the cart.

Implementing business services in an EJB application is little different from implementing them in any other Java application. There's no mystery or magic to it.