Java Bean

A *Java Bean* is a software component that has been designed to be reusable in a variety of different environments. There is no restriction on the capability of a Bean. It may perform a simple function, such as obtaining an inventory value, or a complex function, such as forecasting the performance of a stock portfolio. A Bean may be visible to an end user. One example of this is a button on a graphical user interface. A Bean may also be invisible to a user. Software to decode a stream of multimedia information in real time is an example of this type of building block. Finally, a Bean may be designed to work autonomously on a user’s workstation or to work in cooperation with a set of other distributed components.

**Advantages of Java Beans**

The following list enumerates some of the benefits that Java Bean technology provides for a

component developer:

• A Bean obtains all the benefits of Java’s “write-once, run-anywhere” paradigm.

• The properties, events, and methods of a Bean that are exposed to another

application can be controlled.

• Auxiliary software can be provided to help configure a Bean. This software is only

needed when the design-time parameters for that component are being set. It does

not need to be included in the run-time environment.

• The configuration settings of a Bean can be saved in persistent storage and restored

at a later time.

• A Bean may register to receive events from other objects and can generate events

that are sent to other objects.

A Java Bean is a java class that should follow following conventions:

* It should have a no-arg constructor.
* It should be Serializable.
* It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use Java Bean?

|  |
| --- |
| According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object, so we can access this object from multiple places. Moreover, it provides the easy maintenance. |

Simple example of java bean class

1. //Employee.java
3. **package** mypack;
4. **public** **class** Employee **implements** java.io.Serializable{
5. **private** **int** id;
6. **private** String name;
8. **public** Employee(){}
10. **public** **void** setId(**int** id){**this**.id=id;}
12. **public** **int** getId(){**return** id;}
14. **public** **void** setName(String name){**this**.name=name;}
16. **public** String getName(){**return** name;}
18. }

How to access the java bean class?

|  |
| --- |
| To access the java bean class, we should use getter and setter methods. |

1. **package** mypack;
2. **public** **class** Test{
3. **public** **static** **void** main(String args[]){
5. Employee e=**new** Employee();//object is created
7. e.setName("Arjun");//setting value to the object
9. System.out.println(e.getName());
11. }}

### What Is an Enterprise Bean?

Written in the Java programming language, an **enterprise bean** is a server-side component that encapsulates the business logic of an application. The business logic is the code that fulfills the purpose of the application. In an inventory control application, for example, the enterprise beans might implement the business logic in methods called checkInventoryLevel andorderProduct. By invoking these methods, clients can access the inventory services provided by the application.

#### Benefits of Enterprise Beans

For several reasons, enterprise beans simplify the development of large, distributed applications. First, because the EJB container provides system-level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container, rather than the bean developer, is responsible for system-level services such as transaction management and security authorization.

Second, because the beans rather than the clients contain the application’s business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant Java EE server provided that they use the standard APIs.

#### When to Use Enterprise Beans

You should consider using enterprise beans if your application has any of the following requirements:

* The application must be scalable. To accommodate a growing number of users, you may need to distribute an application’s components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain transparent to the clients.
* Transactions must ensure data integrity. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.
* The application will have a variety of clients. With only a few lines of code, remote clients can easily locate enterprise beans. These clients can be thin, various, and numerous.

**JavaBeans vs EJB**

|  |  |
| --- | --- |
| **JavaBeans** | **EJB** |
| JavaBeans always behave as Local Object. Java Bean as is a plain java class with member variables and getter setter methods. | Enterprise Java Beans can behave as a Remote Object. EJBs are delpoyed on the EJB container and executes in the EJB container. EJB is not that simple, it is used for building distributed applications. |
| As it can be used as client as well server side programming, it is GUI/Non-GUI based. | As it is middle-tier component it can be used only as Non-GUI based. |

**Types of EJB:**

#### 1) Session Bean

A Session Bean is an Enterprise Bean that is generated for each session from the client and expires when the client exits. The lifecycle of the Session Bean does not exceed the range from the beginning until the end of the usage of the system by the user. Session Beans are classified into Stateless Session Beans, Stateful Session Beans, and Singleton Session Beans.

##### (a) Stateless Session Bean

This is a model in which the state of the session is not managed. Each session from the client needs to be concluded in one invocation of the Bean business method.

##### (b) Stateful Session Bean

This is a model in which the state of the session is managed. The EJB container manages the state of the session. Even when one session from the client invokes multiple EJB business methods, the state of each session is saved in between the invocation of the business methods.

##### (c) Singleton Session Bean

This is a model in which the state of the session is shared among multiple clients. One instance which is shared among all the sessions is created for an application. You must determine the Bean lifecycle according to the application.

For details on Singleton Session Beans that you can use with Application Server, see [*2.19 Using Singleton Session Beans*](http://itdoc.hitachi.co.jp/manuals/3020/30203Y0610e/EY060110.HTM#ID00321).

#### (2) Entity Bean

An Entity Bean expresses the entity, and as a prerequisite, must be stored (persisted) in the database. As a result, even when the client exits, the state of the Entity Bean continues to exist in the database. The lifecycle of this Enterprise Bean is longer as compared to that of a Session Bean. The following two management models are defined in the EJB specifications:

##### (a) BMP (Bean Managed Persistence)

This is a model for managing the data persistence of Enterprise Bean business methods. The developer of the Enterprise Bean must implement processes such as connecting to the database, assembling and executing SQL statements.

##### (b) CMP (Container-Managed Persistence)

This is a model in which the EJB container manages the data persistence. The EJB container executes processes, such as connecting to the database and storing data, therefore, these processes need not be executed by the business methods of the Enterprise Bean. Use the method provided by the EJB container to define the mapping of the Enterprise Bean data and the tables and columns of the database in which the data is to be stored. At the same time, define the connection information, such as the host name and port number of the database to which you will connect, in a resource adapter or a data source. The EJB container references this definition information, assembles the SQL statements, and references and stores the data in the tables of the database to which you will connect.

Note that since EJB QL has been implemented in CMP 2.0 added in EJB 2.0, the process of searching the database can be coded in the DD with syntax such as SQL, independent of the database to be used. In addition, CMR (Container-Managed Relationship) has been implemented to establish a relationship between the Entity Beans, therefore, the relationship between the Entity Beans can be specified in the DD and managed by the EJB container.

For details on the Java data type of the Entity Bean of the CMP functionality in the EJB container provided by Application Server and the SQL data type of the database, see [*2.4.2 Mapping the CMP field and database*](http://itdoc.hitachi.co.jp/manuals/3020/30203Y0610e/EY060026.HTM#ID00082).

#### (3) Message-driven Bean

A Message-driven Bean is a bean that integrates with JMS. The EJB container invokes a Bean, when a JMS message is received from the JMS Destination. Unlike a Session Bean or an Entity Bean, since the Message-driven Bean does not have a home interface and a component interface, it cannot be invoked directly from the client.

For a Message-driven Bean, the interfaces used for implementation in the EJB version 2.0 are different from the interfaces used for implementation in the EJB version 2.1 or later versions.

* In EJB 2.0, the following interfaces are implemented:
  + javax.ejb.MessageDrivenBean interface
  + javax.jms.MessageListener interface
* In EJB 2.1 or later versions, the following interfaces are implemented:
  + javax.ejb.MessageDrivenBean interface
  + Interface of any message listener provided by EIS