CIE-306T

Advanced Java Programming

JAVA BEANS

B.Tech (CSE)
January, 2025

Nihar Ranjan Roy
Associate Professor,
VIPS School of Engineering & Technlogy
nihar.roy@vips.edu

VIPS
योगः कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

Java Beans

- JavaBeans are reusable software components for Java that follow specific conventions. They are commonly used in Java EE applications for encapsulating data and following the **getter-setter** principle.

Steps to Create Java Beans

A Java Bean must follow these conventions:

1. It should be a public class.
2. It should have a no-argument constructor (default constructor).
3. It should have private properties (fields).
4. It should provide public getter and setter methods for accessing and modifying the properties.
5. It should be serializable (optional but recommended).

Why Use Java Beans?

1. Encapsulation: Keeps fields private and provides controlled access.
2. Reusability: Java Beans can be easily reused in different applications.
3. Interoperability: Java Beans work well with frameworks like Spring, Hibernate, and Java EE.
4. Persistence: Since they are serializable, they can be saved to files, databases, or transferred over a network.

Properties of Java Beans

Below are the key properties of Java Beans:

- Encapsulation
- No-Argument Constructor
- Serializable
- Properties(Attributes)
- Support Event Handling (Optional)
- Reusability and Manageability
- Configurability
- Platform Independence

Encapsulation

- All fields (variables) must be private.
- Public getter and setter methods must be used to access and modify properties.

```
private String name;  // Private field
public String getName() { return name; }  // Getter
public void setName(String name) { this.name = name; }  // Setter
```

No-Argument Constructor

- Java Beans must have a public no-argument (default) constructor.
- This allows easy instantiation and management by frameworks.

```
public class Student {
    public Student() {}  // No-argument constructor
}
```

Serializable

- Java Beans should implement the Serializable interface to allow storage and transfer.
- This helps in saving bean objects to a file, database, or transferring over a network.

```
import java.io.Serializable;
public class Student implements Serializable {
    private static final long serialVersionUID = 1L;
}
```

Properties (Attributes)

- Java Beans have properties, which are accessed through getter and setter methods.
- Properties can be:
  - Simple Property: A single variable (e.g., name, age).
  - Indexed Property: An array or list (e.g., String[] subjects).
  - Bound Property: Notifies listeners when its value changes.

Supports Event Handling (Optional)

- Java Beans can register event listeners to notify other components when their state changes.
- Example: A graphical UI component updating when a property changes.

Reusability and Manageability

- Java Beans are modular and can be reused across different applications.
- Frameworks like Spring, Hibernate, and Java EE utilize Java Beans extensively.

Configurability

- Properties of Java Beans can be configured using external files (e.g., XML, property files).
- Many frameworks use reflection to dynamically set values at runtime.

Example

```
import java.io.Serializable;
public class Student implements Serializable {
// Implements Serializable for optional persistence
    private String name;
    private int age;
public Student() {} // No-argument constructor
    // Getter method for name
    public String getName() {
        return name;
    }
    // Setter method for name
    public void setName(String name) {
        this.name = name;
    }
    // Getter method for age
    public int getAge() {
        return age;
    }

    // Setter method for age
    public void setAge(int age) {
        this.age = age;
    }
}
```

Using the Java Bean

```
public class Main {
    public static void main(String[] args) {
        // Create an object of Student bean
        Student student = new Student();

        // Set values using setter methods
        student.setName("John Doe");
        student.setAge(22);

        // Get values using getter methods
        System.out.println("Student Name: " + student.getName());
        System.out.println("Student Age: " + student.getAge());
    }
}
```

Types of java beans

1. Simple Java Beans – Basic reusable components with encapsulated properties.
2. Entity Beans (EJB - Enterprise Java Beans) – Used for database persistence in Java EE.
3. Session Beans (EJB - Enterprise Java Beans)
   - Stateless – No client-specific state.
   - Stateful – Maintains client session data.
   - Singleton – Single instance shared across the application.
4. Message-Driven Beans (MDB - EJB Component) – Handles asynchronous messages (JMS).
5. Managed Beans (Java EE & Spring Beans) – Supports dependency injection in web applications.
6. Spring Beans – Specialized Java Beans managed by the Spring Framework.

Simple Java Beans

- Basic, reusable components that encapsulate properties with getter and setter methods.
- Used in applications to store and manage data.
- Example: A Student bean storing name and age.

```
import java.io.Serializable;

public class Student implements Serializable {
    private String name;
    private int age;

    public Student() {} // No-argument constructor

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
}
```

Entity Beans (EJB - Enterprise Java Beans)

- Part of Java EE (Jakarta EE now) used in enterprise applications.
- Manages persistence with databases (e.g., using JPA).
- Runs inside an EJB container, ensuring transaction management, security, and scalability.

```
import jakarta.persistence.*;

@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private double salary;

    // Getters and Setters
}
```

Session Beans (EJB - Enterprise Java Beans)

- Used in Java EE applications to handle business logic.
- Types:
  - Stateless Session Beans: Do not maintain client state.
  - Stateful Session Beans: Maintain client-specific session data.
  - Singleton Session Beans: A single instance shared across the application.

```
import jakarta.ejb.Stateless;

@Stateless
public class CalculatorBean {
    public int add(int a, int b) {
        return a + b;
    }
}
```

Message-Driven Beans (MDB - EJB Component)

- Handles asynchronous messages from a message queue (JMS).
- Used in event-driven applications like chat systems and banking notifications.

```
import jakarta.ejb.MessageDriven;
import jakarta.jms.Message;
import jakarta.jms.MessageListener;

@MessageDriven
public class NotificationBean implements MessageListener {
    public void onMessage(Message message) {
        System.out.println("Received Notification: " + message);
    }
}
```

Managed Beans (Java EE & Spring Beans)

- Used in Java EE (Jakarta EE) and Spring Framework.
- Supports dependency injection and lifecycle management.

```
import jakarta.enterprise.context.RequestScoped;
import jakarta.inject.Named;

@Named
@RequestScoped
public class UserBean {
    private String username;

    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
}
```

Used in **JSF (JavaServer Faces), Spring, and Jakarta EE applications**.

Spring Beans

- Specialized Java Beans managed by Spring Framework.
- Configured using annotations (@Component, @Service, @Repository) or XML files.
- Supports Dependency Injection (DI) and Inversion of Control (IoC).

```
import org.springframework.stereotype.Component;

@Component
public class Product {
    private String name;
    private double price;

    // Getters and Setters
}
```

Used in **Spring Boot and Spring MVC applications**.

Difference Between Stateless and Stateful Session Beans in EJB

Feature	**Stateless Session Bean**	**Stateful Session Bean**
State Management	Does **not** maintain client-specific state between method calls.	Maintains client-specific state between multiple method calls.
Client Association	Shared across multiple clients. A new instance may be used for each request.	Associated with a single client for the session duration.
Use Case	Best for operations that do not require session persistence (e.g., authentication, stateless computations).	Best for processes that require session persistence (e.g., shopping cart, user session tracking).
Performance	More scalable as instances can be shared among multiple clients.	Less scalable as each client gets its own instance, consuming more resources.
Lifecycle	Created and destroyed per request or managed by the EJB container.	Created when a client session starts and remains active until the session ends or explicitly removed.
Passivation	**Not passivated**, as no state is maintained.	Can be **passivated** (saved to disk) if inactive, and **activated** (restored) when needed to optimize resources.
Example	A bean handling login authentication, order processing, or sending notifications.	A bean maintaining a user's shopping cart or a banking transaction.

Conclusion

Type	Usage
Simple Java Beans	Data management in standalone applications
Entity Beans (JPA/EJB)	Database persistence
Session Beans (EJB)	Business logic in enterprise apps
Message-Driven Beans (MDB)	Event-driven messaging
Managed Beans (Jakarta EE, JSF, Spring)	Web & UI frameworks
Spring Beans	Spring applications with DI