Parul® University

Advance Java

Prof. Gaurav Kumar, Assistant Professor
IT and Computer Science

Parul®
University

DIGITAL LEARNING CONTENT

Unit -5

Introduction to Spring and Hibernate

Spring Framework

- The **Spring Framework** is an open-source, powerful, lightweight, and widely used Java framework for building enterprise applications.
- It simplifies enterprise application development by using techniques like Aspect-Oriented Programming (AOP), Plain Old Java Objects (POJO), and Dependency Injection (DI).
- It provides alternatives to traditional Java APIs like JDBC, JSP, and Servlets.

Features of Spring Framework

- **Dependency Injection:** Dependency Injection is a design pattern where the Spring container automatically provides the required dependencies to a class, instead of the class creating them itself. This promotes loose coupling, easier testing, and better maintainability by decoupling the object creation and usage.
- **Aspect-Oriented Programming (AOP):** AOP allows developers to separate cross-cutting concerns (such as logging, security, and transaction management) from the business logic.
- **Spring MVC:** It is a powerful framework for building

Aspect-Oriented Programming (AOP)

Example:-

```
@Aspect
@Component
public class LoggingAspect {
    @Before("execution(* com.example.service.*.*(..))")
    public void logBefore() {
        System.out.println("Executing method...");
    }
}
```

- This logs a message before executing any method in com.example.service.

Inversion of Control (IOC) Container

- Inversion of Control (IoC) is a design principle used in object-oriented programming where the control of object creation and dependency management is transferred from the application code to an external framework or container.
- This reduces the complexity of managing dependencies manually and allows for more modular and flexible code.
- In Spring framework there are mainly two

Inversion of Control (IOC) Container

1. BeanFactory: BeanFactory is the simplest container and is used to create and manage beans. It is a basic container that initializes beans lazily (i. e., only when they are needed). It is typically used for lightweight applications where the overhead of ApplicationContext is not required.

Example :-

<bean id=”car” class=”com.example.Car”/>

This will create a Car bean inside the IoC container, which will be initialized when requested.

Inversion of Control (IOC) Container

2. Application Context: ApplicationContext is an advanced container that extends BeanFactory and provides additional features like internationalization support, event propagation, and AOP (Aspect-Oriented Programming) support. The ApplicationContext is preferred in most Spring applications because of its enhanced features.

Example:-

<context:component-scan base-package=”com.example”/>

This will scan the specified package for annotated components beans like **@Component, @Service**

Dependency Injection

Dependency Injection is a design pattern used in software development to implement Inversion of Control.

It allows a class to receive its dependencies from an external source rather than creating them within the class. This reduces the dependency between classes and makes the system more maintainable.

Types of Dependency Injection

1. Constructor Injection
2. Setter Injection
3. Field Injection

Constructor Injection

In constructor injection, the dependent object is provided to the class via its constructor. The dependencies are passed when an instance of the class is created.

Example:-

```
public class Car {
    private Engine engine;
    public Car(Engine engine) {
    this.engine = engine;
}}
```

Setter Injection

In setter injection, the dependent object is provided to the class via a setter method after the class is instantiated. This allows you to change the dependencies dynamically.

Example:-

```
public class Car {
    private Engine engine;
    public void setEngine(Engine engine) {
    this.engine = engine;
}}
```

Field Injection

In field injection, the dependent object is directly injected into the class through its fields. It is done using framework like Spring(via annotations). The Dependency Injection automatically injects the dependency without requiring explicit constructor or setter methods.

Example:-

```
public class Car {
@Autowired
private Engine engine;
```

Spring Annotation

@Component: Marks a class as a Spring bean, allowing Spring to automatically detect and manage it during classpath scanning.

@Autowired: Automatically injects dependencies into a class. It can be used on fields, constructors, or methods, allowing Spring to resolve and inject the required beans.

@Bean: Defines a Spring bean explicitly within a configuration class. This is used to create and configure beans that are not automatically detected by classpath scanning.

@Configuration: Indicates that a class contains bean definitions and acts as a source of bean configuration. It is used to mark a class as a configuration class that contains

Spring MVC(Model-View-Controller)

- The **Spring MVC Framework** follows the **Model-View-Controller** architectural design pattern, which works around the **Front Controller,** i.e., the **Dispatcher Servlet**. The Dispatcher Servlet handles and dispatches all incoming HTTP requests to the appropriate controller.
- It uses @Controller and @RequestMapping as default request handlers.
  The **@Controller** annotation defines that a particular class is a controller.
  The **@RequestMapping** annotation maps web

Spring MVC(Model-View-Controller)

The terms model, view, and controller are defined as follows:

Model: The Model encapsulates the application data.

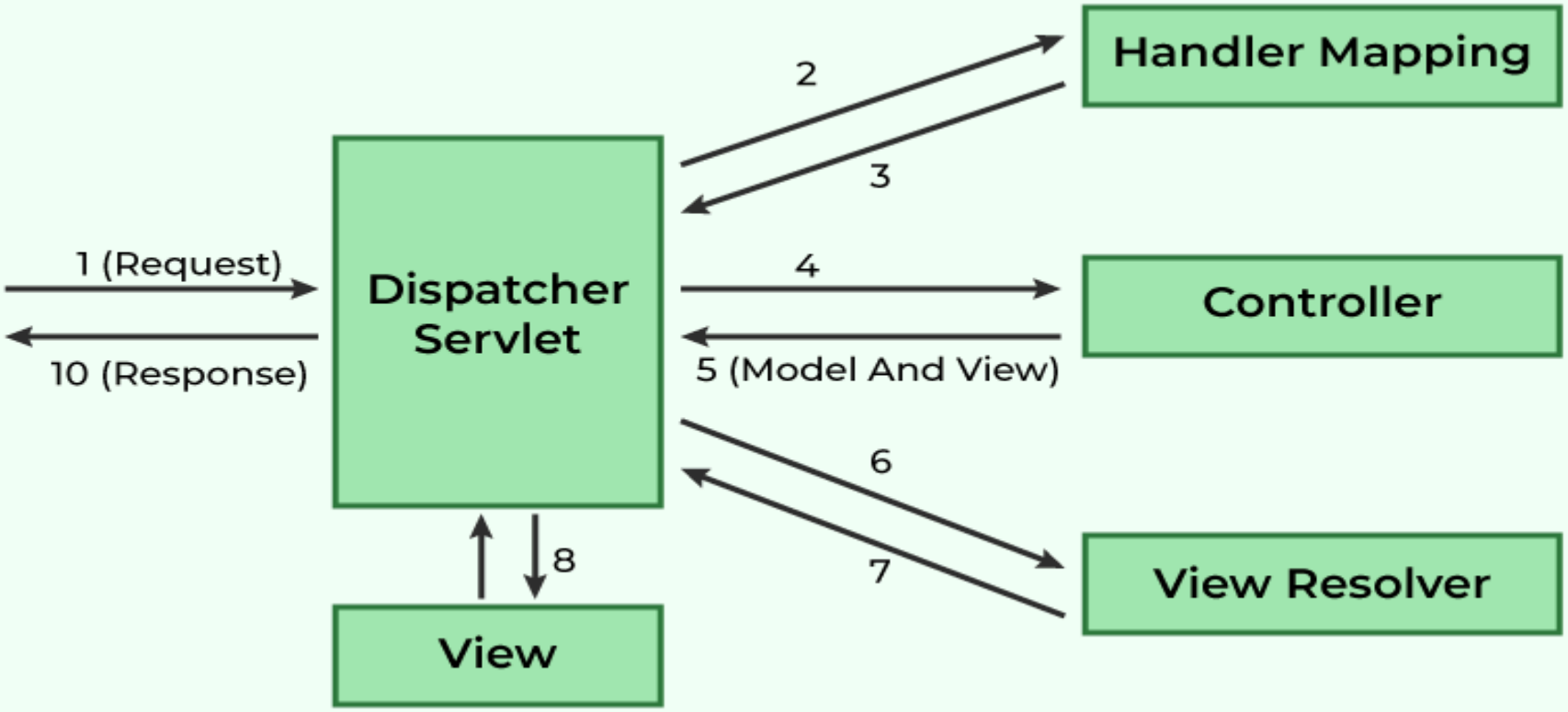
View: The View renders the model data and generates HTML output that the client's browser can interpret.

Controller: The Controller processes the user requests and passes them to the view

Advantages of Spring MVC

- **Separation of Concerns** – Clean division of Model, View, and Controller.
- **Loose Coupling** – Uses Dependency Injection to reduce tight coupling.
- **Flexible View Technology** – Works with JSP, Thymeleaf, PDFs, Excel, etc.
- **REST Support** – Easily create RESTful web services.
- **Built-in Exception Handling** – Provides default error handling.

Spring MVC Control Flow Diagram



Spring MVC Work Flow

- All incoming requests are intercepted by the DispatcherServlet, which works as the front controller.
- The DispatcherServlet retrieves an entry of handler mapping from the configuration file and forwards the request to the controller.
- The controller processes the request and returns an object of ModelAndView.
- The DispatcherServlet checks the entry of the view resolver in the configuration file and

Dispatcher Servlet

The Dispatcher Servlet is the front controller that manages the entire HTTP request and response handling process. Now, the question is: **What is a Front Controller?** It is quite simple, as the name suggests:

When any web request is made, it first goes to the Front Controller, which is the Dispatcher Servlet. The Front Controller stands first, which is why it is named as such. After the request reaches it, the Dispatcher Servlet determines the appropriate controller to handle the request.

Handler Mapping

- Handler Mapping is responsible for mapping a URL request to a controller.

Example:-

```
@Controller
public class HomeController {
    @RequestMapping("/home")
    public String home() {
        return "homepage";
}}
```

- DispatcherServlet checks HandlerMapping and finds home() method.

Http Handler Mapping

- **Http Get Request**

```
@GetMapping("/users")
public String fetchUsers() {
    return "userList";
}
```

- **Http Post Request**

```
@PostMapping("/register")
public String createUser() {
    return "success";
}
```

Passing Parameters in Request Mapping

1. Path Variables (@PathVariable)

- Used to capture values from the URL path

```
@GetMapping("/user/{id}")
public String getUserById(@PathVariable("id") int userId,
Model model) {
    model.addAttribute("userId", userId);
    return "userDetails";}
```

2.Query Parameters (@RequestParam)

- Used to capture values from URL query strings (?key=value).

```
@GetMapping("/search")
public String search(@RequestParam("q") String query,
```

View in Spring MVC

- A View is the part of an MVC application that displays data to users.
- It can be a JSP page, Thymeleaf template, PDF, or Excel document.
- The Controller passes data to the view, which renders it for the user.

Example:

- A user visits http://localhost:8080/welcome.
- The controller processes the request and returns a view (welcome.jsp)..

View Resolver

- Spring MVC uses View Resolvers to determine which view to render.
- View resolvers map logical view names (e.g., "home") to actual files (e.g., /WEB-INF/views/home.jsp).
- Example :-
- Controller returns a logical view name: return "home";.
- View Resolver maps "home" to /WEB-INF/views/home.jsp.

Passing Data to Views using Model

- Model is an interface used to pass attributes to the view.
- It is typically used in a controller method with @RequestMapping.
- You add data using model.addAttribute("key", value);.
- The view name is returned separately as a String.

```
@Controller
public class MyController {
    @GetMapping("/example")
    public String example(Model model) {
        model.addAttribute("message", "Hello from
```

Passing Data to Views using Model and View

- ModelAndView (class) combines both the model data and the view name into a single object.
- You set the view name using setViewName("viewName").
- You add data using addObject("key", value).

```
@Controller
public class MyController {
    @GetMapping("/example")
    public ModelAndView example() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("viewName");
        mav.addObject("message", "Hello from
```

Introduction to Hibernate

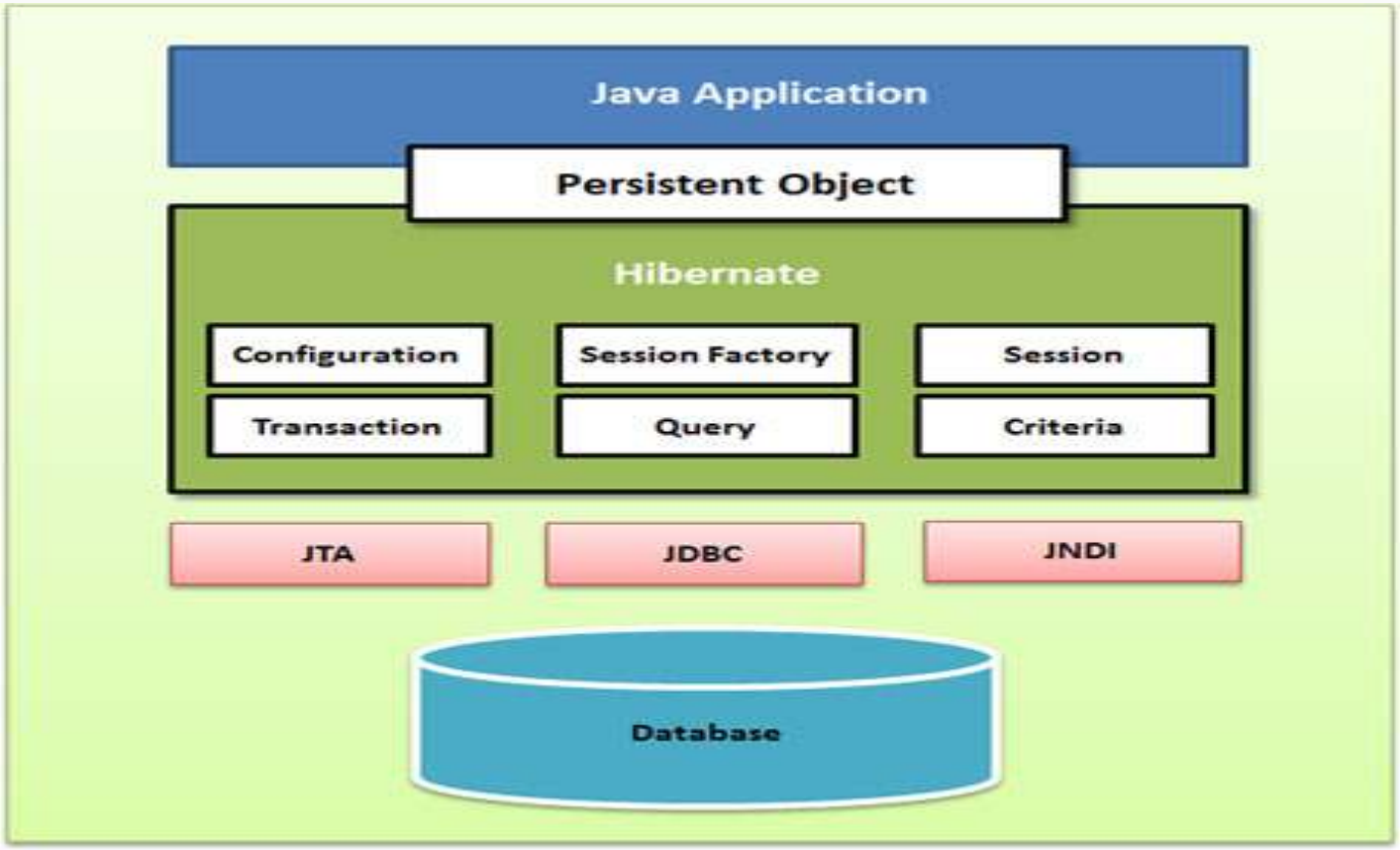
- Hibernate is an **O**bject-**R**elational **M**apping (ORM) solution for JAVA.
- It is an open source, powerful framework for Java Application.
- Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks.
- Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/

Introduction to Hibernate

Advantages of Hibernate

- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in the database or in any table, then you need to change the XML file properties only.
- Abstracts away the unfamiliar SQL types and provides a way to work around familiar Java Objects.

Architecture of Hibernate



Hibernate Configuration

- Hibernate requires a set of configuration settings related to database and other related parameters.
- All such information is usually supplied as a standard Java properties file called **hibernate.properties**, or as an XML file named **hibernate.cfg.xml**.

Hibernate Configuration

```
<hibernate-configuration>
 <session-factory>
 <property name="hbm2ddl.auto">update</property>
<propertyname="dialect">org.hibernate.dialect.
MySQL8Dialect <propertyname="connection.url">jdbc:
mysql://localhost/</property> <property
name="connection.username">root</property>
 <property name="connection.password">guest123</
property> <property name="connection.
driver_class">com.mysql.cj.jdbc.Driver
<mapping resource="employee.hbm.xml"/>
</session-factory>
```

Session

- A Session is used to get a physical connection with a database.
- The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database.
- Persistent objects are saved and retrieved through a Session object.
- The main function of the Session is to offer, create, read, and delete operations for instances of mapped entity classes.

Transaction

- A **Transaction** object in Hibernate (or JDBC) represents a **unit of work** that you want to execute **atomically** — meaning **either all operations succeed, or none do**.
- A Transaction object is used to begin, commit, or rollback a database transaction.
- begin(): Starts the transaction.
- commit(): Saves changes permanently.
- rollback(): Cancels changes if something goes wrong.

Transaction

Example:-

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();

try {
    // perform DB operations
    session.save(new Student(...));
    tx.commit();  // All good, save to DB
} catch (Exception e) {
    tx.rollback();  // Something went wrong, cancel it
}
```

Hibernate O/R Mapping

- O/R Mapping is the process of linking Java objects to database tables.
- It helps in persisting Java class instances into the database.
- Hibernate automates this mapping using annotations or XML.

```
@Entity                 // Maps the class to a table
@Table(name = "students")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(name = "student_name", nullable = false)
```

Hibernate Query Language

- Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties.
- HQL queries are translated by Hibernate into conventional SQL queries, which in turns perform action on database.
- **From Clause:-**

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
```

Hibernate Query Language

- **Select Clause**

```
String hql = "SELECT E.firstName FROM
Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

- **Where clause**

```
String hql = "FROM Employee E WHERE E.id =
10";
Query query = session.createQuery(hql);
List results = query.list();
```

Hibernate Query Language

- **Update Clause**

```
String hql = "UPDATE Employee set salary = :
salary " + "WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("salary", 1000); query.
setParameter("employee_id", 10);
int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

Hibernate Query Language

- **Delete Clause**

```
String hql = "DELETE FROM Employee " +
"WHERE id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id", 10);
int result = query.executeUpdate(); System.
out.println("Rows affected: " + result);
```

Hibernate Query Language

- **Insert Clause**

```
String hql = "INSERT INTO
Employee(firstName, lastName, salary)" +
"SELECT firstName, lastName, salary FROM
old_employee";
Query query = session.createQuery(hql);
int result = query.executeUpdate(); System.
out.println("Rows affected: " + result);
```

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in