Parul® University

Advance Java

Prof. Renuka Parmar, Assistant Professor
IT and Computer Science

Parul®
University

DIGITAL LEARNING CONTENT

Unit -5

Introduction to Spring and Hibernate

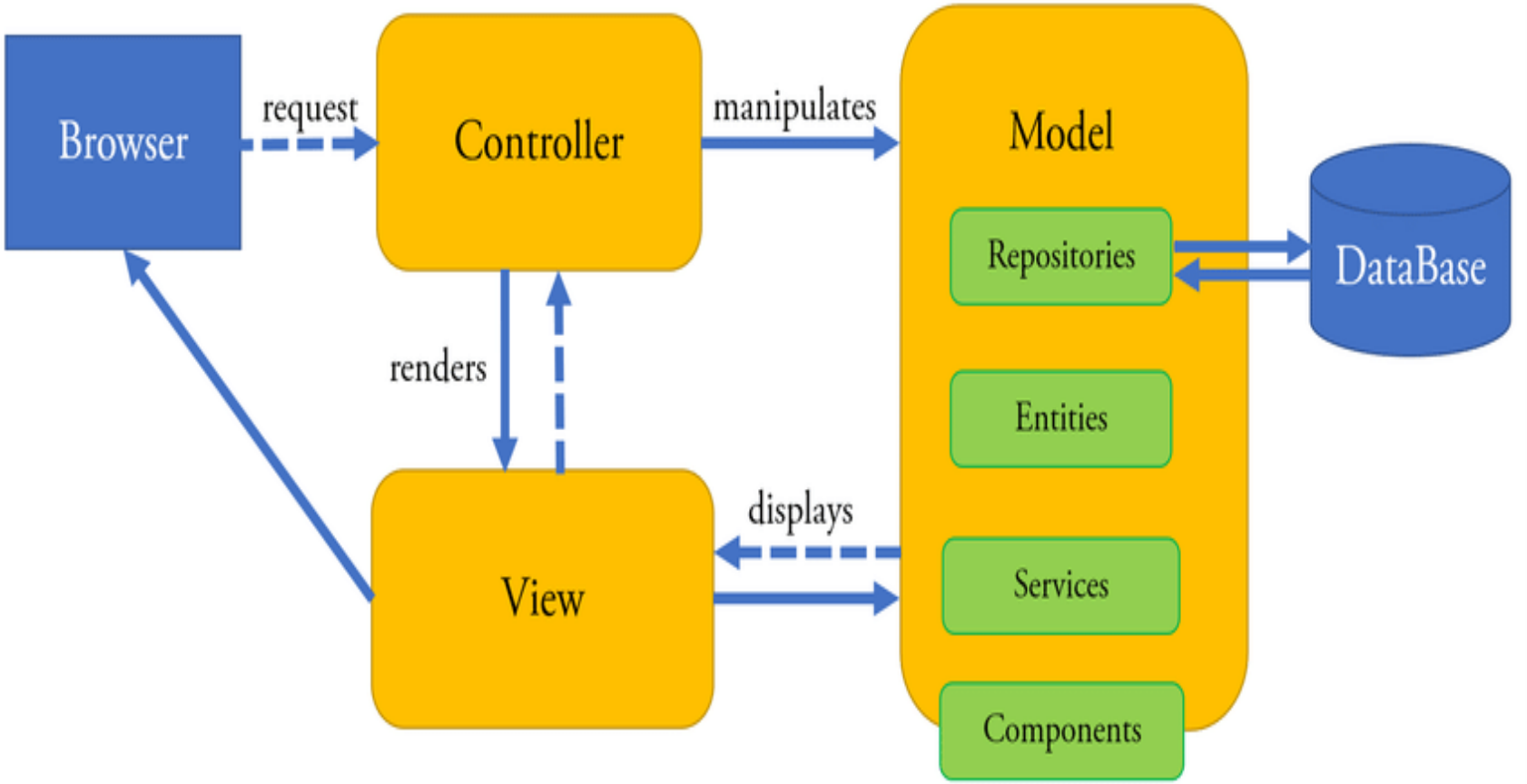
Introduction to Spring MVC Framework

Spring MVC (Model-View-Controller) is a module of the Spring Framework that helps in building web applications. It separates the application into three interconnected components:

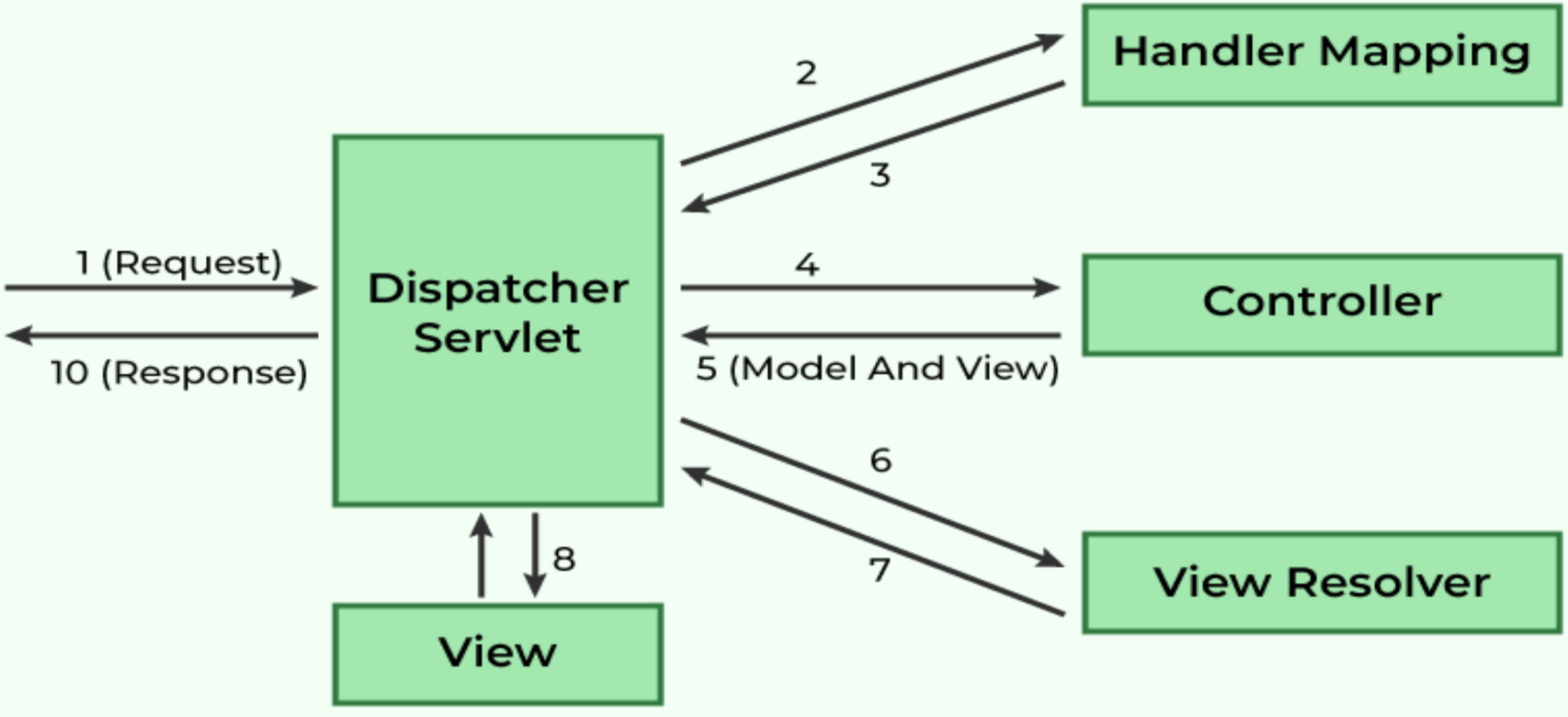
- **Model** – handles the business logic/data.
- **View** – represents the presentation layer (UI).
- **Controller** – handles the user's request and updates the Model and View accordingly.

Spring MVC uses the **Front Controller** design pattern, where the DispatcherServlet acts as the front controller to handle all incoming HTTP requests.

Spring MVC Architecture



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 invokes the appropriate view component.

Dispatcher Servlet

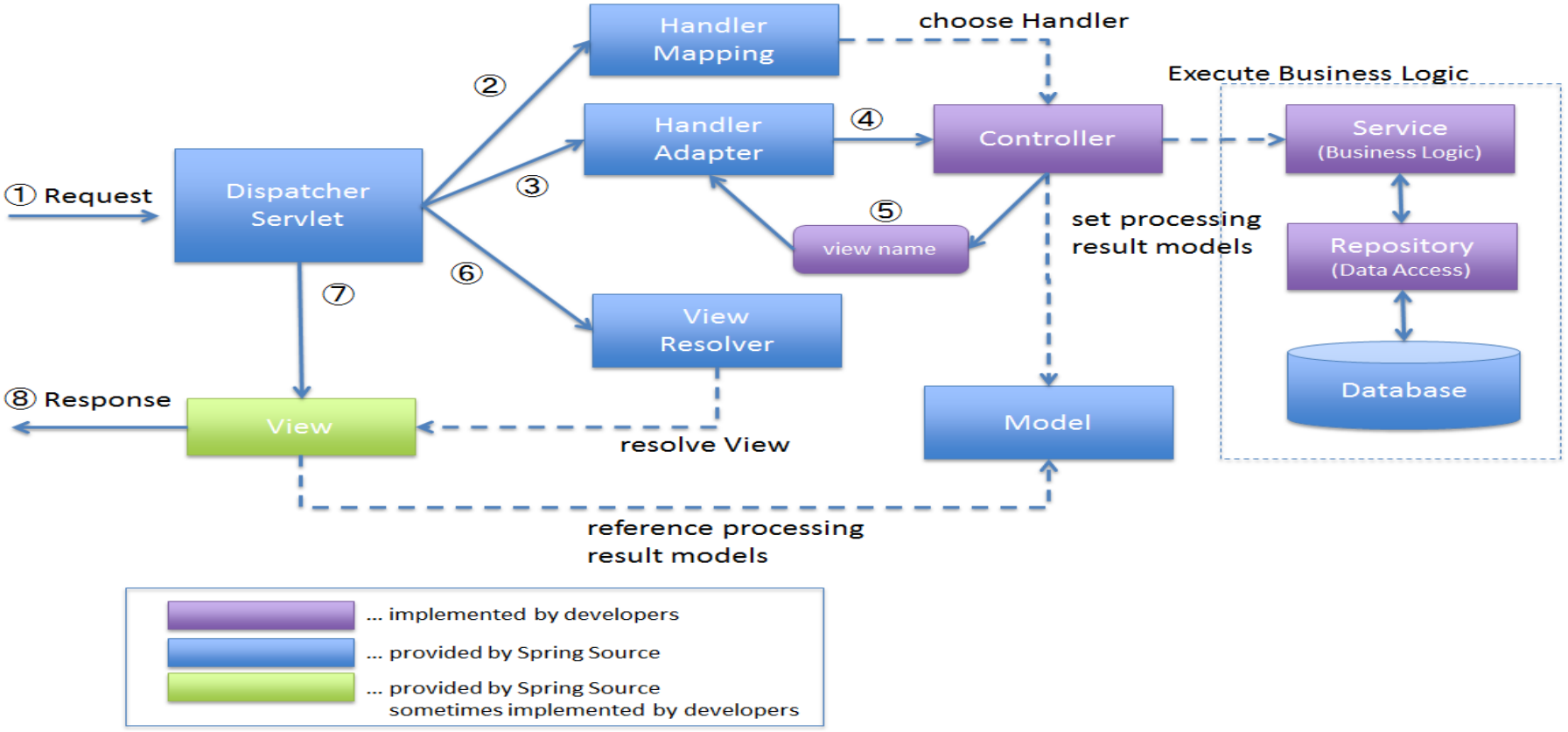
The Dispatcher Servlet is the front controller that manages the entire HTTP request and response handling process. Now, the question is: **What is a Front Controller?** It is quite simple, as the name suggests:

•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.
•Then, it dispatches the HTTP request to the specific controller.

Parul® University

DIGITAL LEARNING CONTENT

Spring MVC Architecture



Continue….

1.DispatcherServlet receives the request.
2.DispatcherServlet dispatches the task of selecting an appropriate controller to HandlerMapping. HandlerMapping selects the controller which is mapped to the incoming request URL and returns the (selected Handler) and Controller to DispatcherServlet.
3.DispatcherServlet dispatches the task of executing of business logic of Controller to HandlerAdapter.
4.HandlerAdapter calls the business logic process of Controller.

Continue….

5.Controller executes the business logic, sets the processing result in Model and returns the logical name of view to HandlerAdapter.
6.DispatcherServlet dispatches the task of resolving the View corresponding to the View name to ViewResolver. ViewResolver returns the View mapped to View name.
7.DispatcherServlet dispatches the rendering process to returned View.
8.View renders Model data and returns the response.

Spring MVC Main Components

Component	Description
DispatcherServlet	Central servlet that dispatches requests to controllers.
Controller	Java class with @Controller annotation to handle logic.
Model	Data holder that is sent to the view.
View	Frontend layer, usually JSP, Thymeleaf, etc.
ViewResolver	Resolves logical view names to actual views.
HandlerMapping	Maps requests to handler methods.

Spring MVC Architecture Flow

1. **Client Request**: Browser sends a request to the server.
2. **DispatcherServlet**: The front controller receives the request.
3. **HandlerMapping**: Finds the appropriate Controller to handle the request.
4. **Controller**: Processes the request and returns a ModelAndView object.
5. **ViewResolver**: Resolves the logical view name into an actual view (like JSP).
6. **View**: Generates the final HTML response sent back to the browser.

Advantages of Spring MVC

1. **Loose Coupling** – Clear separation of Model, View, and Controller.
2. **Easy Integration** – Can easily integrate with other frameworks like Hibernate, JPA, etc.
3. **Annotation-based Configuration** – Reduces boilerplate XML configuration.
4. **Built-in REST Support** – Supports building RESTful services easily.
5. **Testability** – Spring provides easy unit and integration testing capabilities.
6. **Flexible View Resolution** – You can use JSP, PDF, Excel, Thymeleaf, etc.
7. **Exception Handling** – Easy to manage exceptions using @ExceptionHandler.

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";
}
```

Spring MVC Annotations

Annotation	Purpose
@RestController	Combines @Controller and @ResponseBody. Used for REST APIs.
@RequestMapping	Maps HTTP requests to handler methods.
@GetMapping, @PostMapping, @PutMapping, @DeleteMapping	Shortcut annotations for specific HTTP methods.
@PathVariable	Binds URI template variable to method parameter.
@RequestParam	Binds request parameters to method parameters.
@RequestBody	Binds the HTTP request body to a Java object.
@ResponseBody	Sends Java object as the HTTP response body.
@ModelAttribute	Binds form data to model object.
@ExceptionHandler	Handles specific exceptions at controller level.
@ControllerAdvice	Global exception handler across multiple controllers.

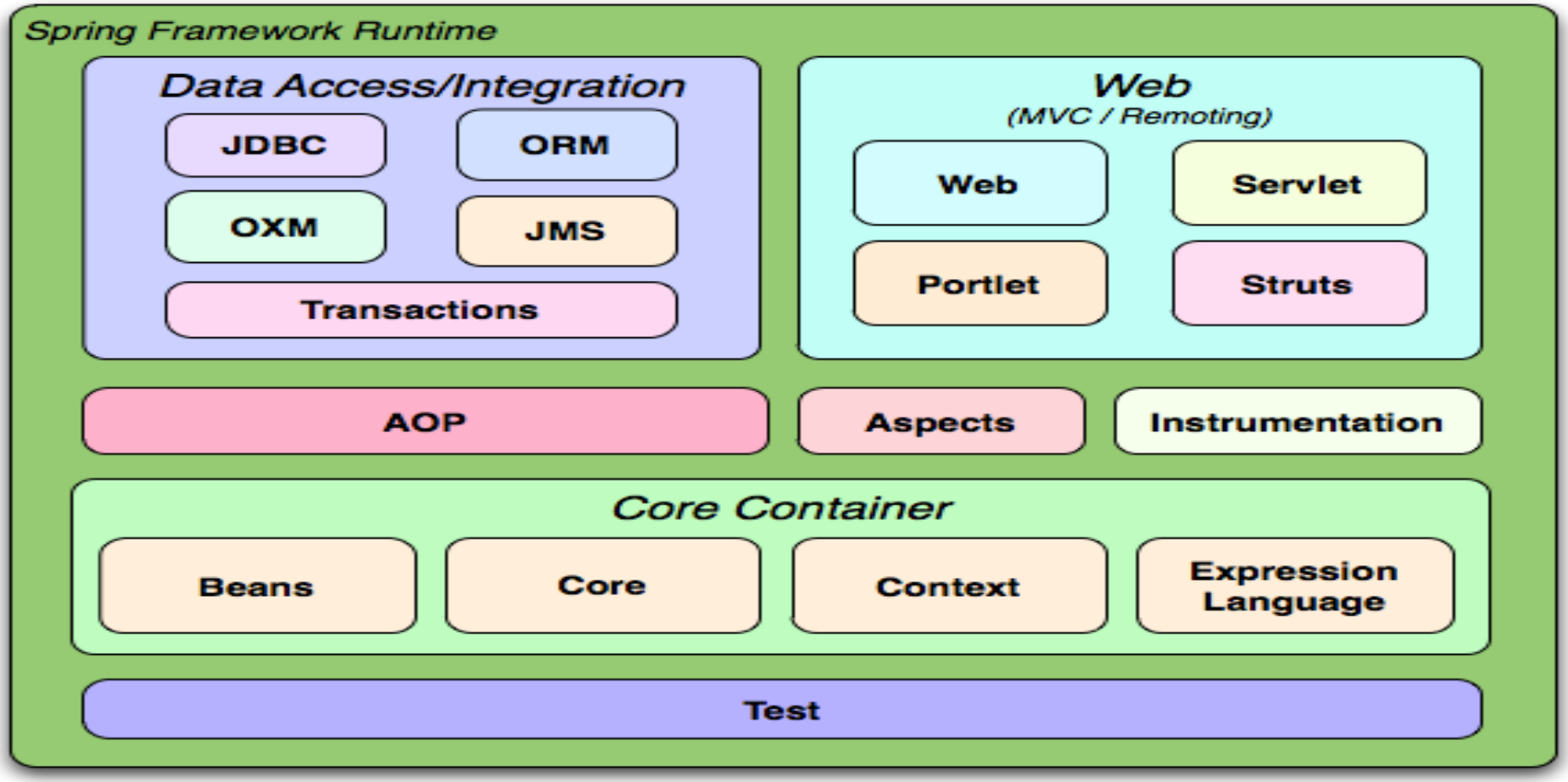
Spring Framework

- 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.

Feature of Spring Framework

Feature	Description
Lightweight	Minimal memory footprint, fast to start.
Inversion of Control (IoC)	Objects are created and managed by the Spring container.
Aspect-Oriented Programming	Separates cross-cutting concerns like logging and security.
Transaction Management	Provides consistent programming model across different transaction APIs.
MVC Web Framework	Builds robust and flexible web applications.
Integration with Other Frameworks	Easily integrates with Hibernate, JPA, JDBC, etc.
Security	Manages authentication and authorization via Spring Security.
Testability	Supports JUnit and TestNG for testing.

SPRING FRAMEWORK



Parul® University

DIGITAL LEARNING CONTENT

Modules Of Spring Framework

Module Name	Description
Core Container	Core, Beans, Context, and Expression Language (SpEL).
Spring AOP	Aspect-Oriented Programming integration.
Data Access/Integration	JDBC, ORM (Hibernate, JPA), JMS, Transactions.
Web Module	Web, Web-MVC, Web-Websocket.
Testing	JUnit/TestNG integration.
Spring Boot	Rapid development, auto-configuration.
Spring Security	Authentication and authorization.
Spring Cloud	Microservices and distributed systems support.

Core Container Modules

Module	Description
Core	Provides core functionalities including Inversion of Control (IoC) and Dependency Injection (DI).
Beans	Deals with bean creation, configuration, and management in the Spring container.
Context	Builds on Core and Beans modules to provide a way to access application objects using a consistent approach.
Expression Language (SpEL)	Allows querying and manipulating object graphs at runtime (used in annotations, XML, etc.).

Data Access/Integration Modules

Module	Description
JDBC	Simplifies the use of JDBC with JdbcTemplate and error handling.
ORM	Provides integration with ORM frameworks like Hibernate, JPA, iBatis.
JMS	Provides features for producing and consuming messages.
Transactions	Supports programmatic and declarative transaction management across different platforms.

Web Modules

Module	Description
Web	Basic web features like multipart file upload, initialization, and web context.
Web-MVC	Provides Model-View-Controller architecture for building web apps (like Spring MVC).
Web-Socket	Adds support for WebSocket for real-time two-way communication between client and server.

AOP and Intrumentation Module

Module	Description
AOP (Aspect-Oriented Programming)	Provides support for separating cross-cutting concerns like logging, security, and transactions.
Aspects	Integration with AspectJ (a popular AOP framework).
Instrumentation	Classloader and byte-code instrumentation support (mainly for app servers).

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.

Test Module

Module	Description
Test	Provides support for unit testing and integration testing using JUnit or TestNG. Supports mocking and loading Spring ApplicationContext for tests.

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 types of IOC Container which are listed below:

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, @Repository,** etc.

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 Core Annotation

Annotation	Purpose
@Component	Marks a class as a Spring-managed component (generic stereotype).
@Controller	Marks a web controller class (used in Spring MVC).
@Service	Marks a service class (used for business logic).
@Repository	Marks a DAO class (data access object), provides exception translation.
@Configuration	Marks a class that contains Spring bean definitions (Java-based config).
@Bean	Declares a bean manually inside a @Configuration class.
@Autowired	Automatically injects dependencies by type.
@Qualifier	Specifies the exact bean to inject when multiple candidates exist.
@Value	Injects values from properties file or constants.
@Primary	Marks a bean as primary to avoid ambiguity when multiple candidates exist.
@Lazy	Marks a bean to be lazily initialized.

Life Cycle & Scope Annotations

Annotation	Purpose
@PostConstruct	Executes a method after the bean initialization.
@PreDestroy	Executes a method before the bean is destroyed.
@Scope("prototype") / "singleton"	Defines the bean scope (default is singleton).

Spring Application

1. Set Up the Development Environment

- **Install JDK** (Java Development Kit) – preferably version 8 or above.
- **Install IDE** – like IntelliJ IDEA, Eclipse, or Spring Tool Suite (STS).
- **Install Maven or Gradle** – for dependency management.

2.Create a New Spring Project

You can do it in two ways:

(a) Using Spring Initializr

•Visit: https://start.spring.io

•Choose:

•Project: Maven or Gradle

•Language: Java

•Spring Boot version

•Project Metadata: Group, Artifact, Name, Description

•Dependencies: Spring Web, Spring Data JPA, Thymeleaf, etc.

•Click **Generate** to download the .zip file

Unzip and open it in your IDE

(b) Using IDE (like Eclipse or IntelliJ)

•File → New → Spring Starter Project

•Follow the wizard to select dependencies and finish setup.

3.Project Structure

```
src/
└── main/
    ├── java/
    │   └── com/example/demo/          → Your Java code
    │       └── DemoApplication.java   → Main class with
@SpringBootApplication
    └── resources/
        ├── application.properties     → Configuration
        ├── templates/                 → For Thymeleaf/HTML
        └── static/                    → For CSS, JS, images
```

4.Main Application Class

```
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

5. Create the Application Components

- **Model** – Define POJO classes.
- **Repository** – Interface for DB operations (e.g., extends JpaRepository).
- **Service** – Business logic.
- **Controller** – Handles web requests using @RestController or @Controller.

6. Configure the Application

In application.properties or application.yml, add things like:

```
server.port=8080
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
```

7. Run the Application

- Run DemoApplication.java as a Java Application.

- OR use terminal:

  mvn spring-boot:run

- Visit http://localhost:8080 in your browser.

8. Test and Debug

- Use Postman or browser to test endpoints.
- Check logs for any issues and debug.

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/R mechanisms and patterns.

Introduction to Hibernate

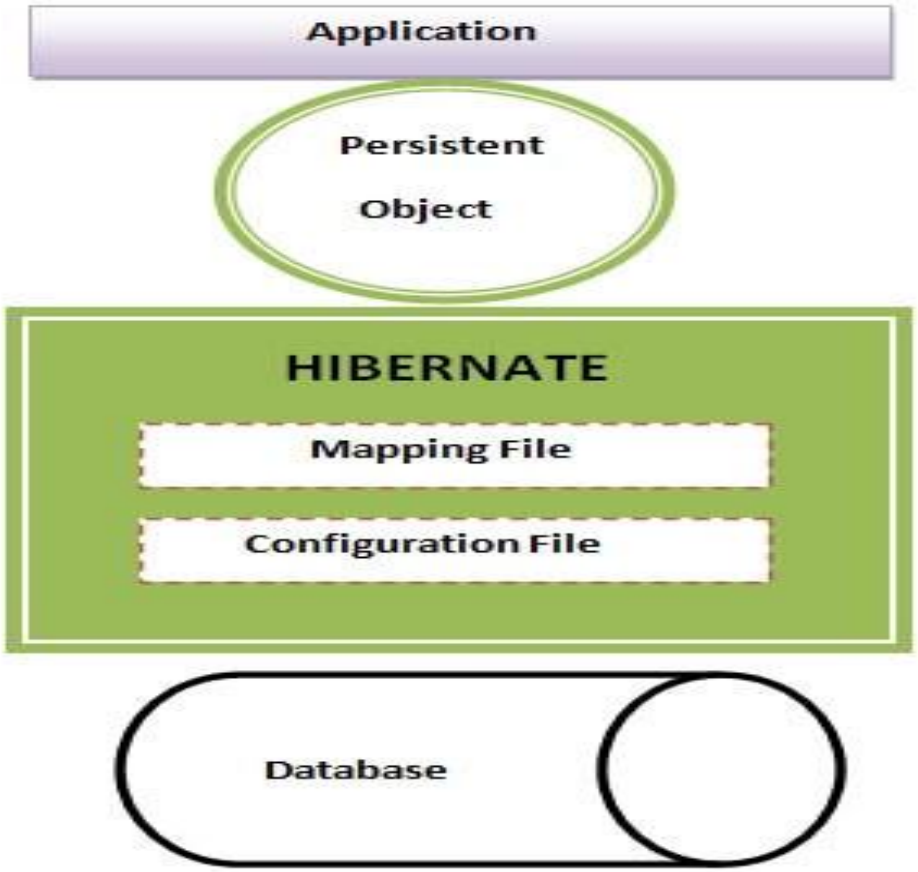
Features of Hibernate

- Lightweight and open-source.
- Automatically handles SQL generation and execution.
- Provides **HQL** (Hibernate Query Language) similar to SQL but for Java objects.
- Supports **caching**, **lazy loading**, and **transaction management**.

Advantages of Hibernate

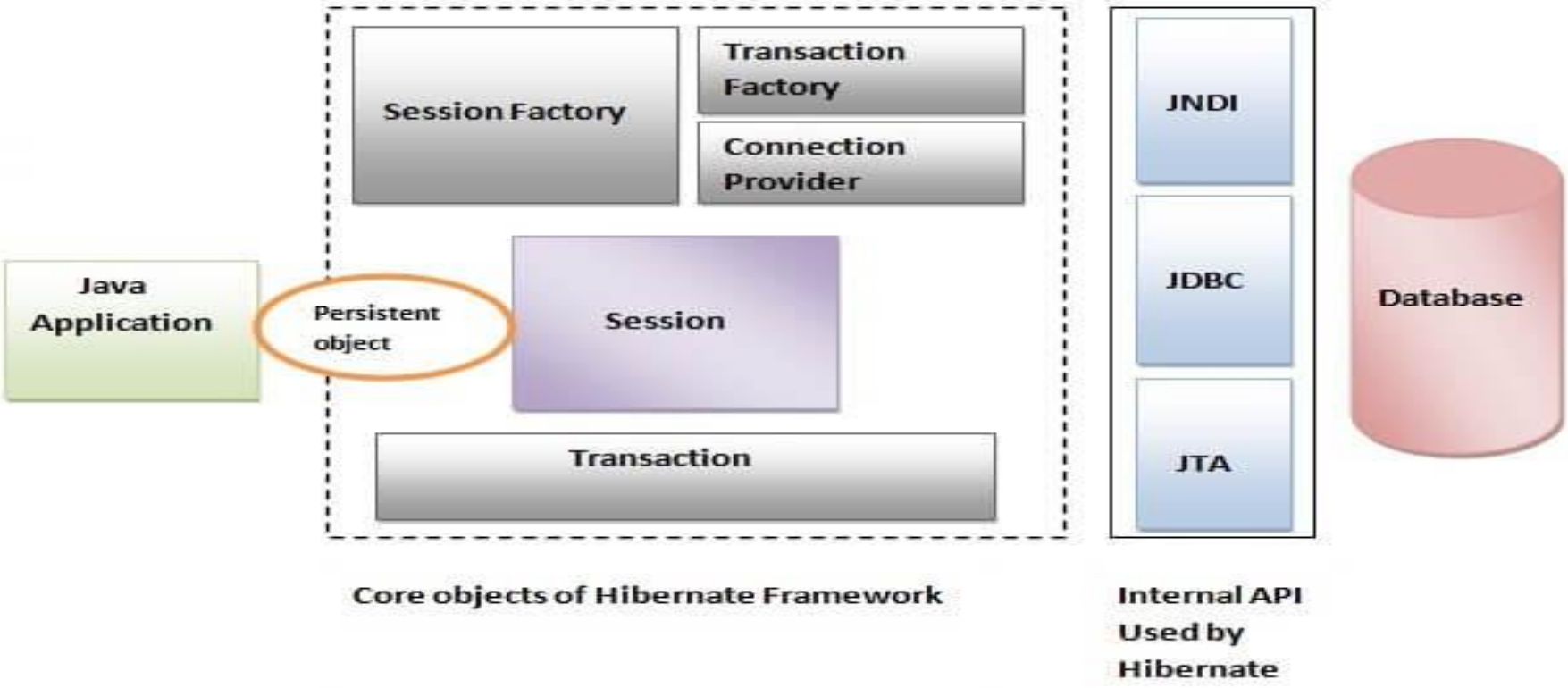
- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in the database or in any table, then you need to change the XML file properties only.
- Abstracts away the unfamiliar SQL types and provides a way to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Manipulates Complex associations of objects of your database.

Architecture of Hibernate



- Java application layer
- Hibernate framework layer
- Backhand api layer
- Database layer

Architecture of Hibernate



Elements of Hibernate Architecture

SessionFactory:

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session:

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Elements of Hibernate Architecture

- **Transaction**: The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.
- **ConnectionProvider**: It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.
- **TransactionFactory**: It is a factory of Transaction. It is optional.

Hibernate Object/Relational Mapping (O/R Mapping)

- O/R Mapping maps Java classes (Objects) to database tables (Relations).
- Each Java class maps to a table.
- Each field (property) maps to a column.
- Mapping can be done in:
- **XML Mapping File** (.hbm.xml)
- **Annotations** (Java 5+)

Elements of Hibernate Architecture

Annotation Example:

```
@Entity
@Table(name="students")
public class Student {
   @Id
   @GeneratedValue
   private int id;

   @Column(name="name")
   private String name;
}
```

Configuring Hibernate Development

- **Step 1: Create Java Project**

Use your IDE to create a new Java project named HibernateExample.

- **Step 2: Add Hibernate & JDBC JARs**

If using **Maven**, add the following to pom.xml:

```
<dependencies>
   <!-- Hibernate Core -->
      <dependency>
       <groupId>org.hibernate</groupId>
       <artifactId>hibernate-core</artifactId>
       <version>5.6.15.Final</version>
      </dependency>
   <!-- MySQL Connector --><dependency>
       <groupId>mysql</groupId>
       <artifactId>mysql-connector-java</artifactId>
       <version>8.0.33</version>  </dependency>
</dependencies>
```

Configuring Hibernate Development

Step 3: Create Hibernate Configuration File

hibernate.cfg.xml:

```
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/testdb</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">your_password</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property name="hibernate.show_sql">true</property>
        <mapping resource="Student.hbm.xml"/>
    </session-factory>
```

Implementing Hibernate Object/Relational Mapping (O/R Mapping)

- **Step 1: Create the POJO Class**

Student.java

```
public class Student {
    private int id;
    private String name;
    private int age;
    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    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; }
}
```

Implementing Hibernate Object/Relational Mapping (O/R Mapping)

- Step 2: Create XML Mapping File

Student.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="Student" table="students">
        <id name="id" column="id">
            <generator class="increment" />
        </id>
        <property name="name" column="name" />
        <property name="age" column="age" />
    </class>
</hibernate-mapping>
```

Implementing Hibernate Object/Relational Mapping (O/R Mapping)

Step 3: Write Main Class to Test [MainApp.java]

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class MainApp {
    public static void main(String[] args) {
        // Create session factory from hibernate.cfg.xml
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        // Create and save object
        Student s1 = new Student();
        s1.setName("Renuka Parmar");
        s1.setAge(25);
        session.beginTransaction();
        session.save(s1);
        session.getTransaction().commit();
        System.out.println("Student saved successfully!");
        session.close();
        factory.close(); }}
```

Introduction to Hibernate Query Language (HQL)

HQL stands for **Hibernate Query Language**.
It is an **object-oriented** query language used to perform operations like **SELECT, INSERT, UPDATE, DELETE** on Java **objects** (not directly on database table)

HQL is **similar to SQL**, but works with **class names** and **property names** instead of table and column names.

- It is **database-independent**.
- Supports **joins**, **aggregations**, **group by**, **order by**, etc.
- Can use **named parameters** (:paramName) or **positional parameters** (?1, ?2).

Introduction to Hibernate Query Language (HQL)

Query Interface

It is an object oriented representation of Hibernate Query. The object of Query can be obtained by calling the createQuery() method Session interface.

The query interface provides many methods. There is given commonly used methods:

1.public int executeUpdate() is used to execute the update or delete query.

2.public List list() returns the result of the ralation as a list.

3.public Query setFirstResult(int rowno) specifies the row number from where record will be retrieved.

4.public Query setMaxResult(int rowno) specifies the no. of records to be retrieved from the relation (table).

5.public Query setParameter(int position, Object value) it sets the value to the JDBC style query parameter.

6.public Query setParameter(String name, Object value) it sets the value to a named query parameter.

Introduction to Hibernate Query Language (HQL)

Student.java

```
@Entity
@Table(name = "students")
public class Student {
    @Id
    @GeneratedValue
    private int id;

    @Column(name = "name")
    private String name;

    @Column(name = "age")
    private int age;

    // Getters and setters...
}
```

Introduction to Hibernate Query Language (HQL)

1. Fetch All Records

```
Query query = session.createQuery("FROM Student");
List<Student> list = query.list();
```

2. Fetch with condition

```
Query query = session.createQuery("FROM Student WHERE age > :minAge");
query.setParameter("minAge", 20);
List<Student> list = query.list();
```

3. Select Specific Columns

```
Query query = session.createQuery("SELECT name FROM Student");
List<String> names = query.list();
```

Introduction to Hibernate Query Language (HQL)

4. Update Record

```
Query query = session.createQuery("UPDATE Student SET age =
:newAge WHERE name = :name");
query.setParameter("newAge", 30);
query.setParameter("name", "Renuka");
int result = query.executeUpdate();
```

5. Delete Record

```
Query query = session.createQuery("DELETE FROM Student
WHERE id = :id");
query.setParameter("id", 1);
query.executeUpdate();
```

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in