PRATIK KUMAR

- **Assignment: 17 JUNE 2024**

1. Frontend Development

- **HTML**: Structure of web pages.
- **CSS**: Styling of web pages.
- **JavaScript**: Interactivity of web pages.
- **AngularJS** or **ReactJS**: Frontend frameworks for building dynamic user interfaces.

2. Middleware Development

- **Java**: Core programming language.
- **Frameworks**:
  - **JDBC**: Java Database Connectivity.
  - **Servlets and JSP**: Java-based web technologies.
  - **Spring Framework**: Comprehensive framework for building Java applications.
  - **Spring Boot**: Simplified Spring framework for creating standalone applications.
  - **Microservices**: Architectural style for developing small, independent services.

3. Backend Development (Database)

- **Relational Databases (SQL)**:
  - **Oracle**
  - **MySQL**
  - **DB2**
  - **H2**
- **Non-Relational Databases (NoSQL)**:
  - **MongoDB**
  - **DynamoDB**

Spring Framework Architecture

Components and Flow

1. **Frontend (User Interface)**
   - Users interact with the frontend, submitting forms or requests.
2. **Middleware (Server Side)**
   - **Dispatch Servlet**: Entry point for all incoming requests. It routes the requests to the appropriate handler.

- **Handler**: Processes the request. This is where the business logic is executed.
- **Handler Mapping**: Contains mappings of requests to their respective handlers.
- **Database Access**: Handlers may connect to the database using frameworks like Hibernate, JPA, or JDBC.
- **Model and View**: After processing, the handler returns a Model and View object.

3. **View Resolver**
   - **View Resolver**: Converts the Model and View object into a format understandable by the end-user (usually HTML).
4. **Response to User**
   - **Dispatch Servlet**: Sends the final view back to the frontend for the user to see.

Example: User Registration Flow

1. **Frontend Form**: User fills in details like first name, last name, mobile number, and clicks submit.
2. **Dispatch Servlet**: Receives the request and forwards it to the appropriate handler based on handler mappings.
3. **Handler**: Processes the request (e.g., validates input, saves data to the database).
4. **Database Interaction**: Uses JDBC, Hibernate, or JPA to interact with the database.
5. **Model and View**: Handler returns a Model and View object.
6. **View Resolver**: Converts the Model and View object to a proper HTML view.
7. **Response**: The Dispatch Servlet sends the HTML view back to the user.

- **<u>Assignment: 18 JUNE 2024</u>**

  Spring Framework Overview

The Spring Framework is a powerful, feature-rich framework for building Java-based enterprise applications. It simplifies the development process by providing comprehensive infrastructure support for developing Java applications. Here are some key modules within the Spring Framework:

1. **Spring IOC (Inversion of Control)**: Manages the creation and life cycle of beans (objects) and dependencies.
2. **Spring MVC (Model-View-Controller)**: Facilitates the development of web applications by separating concerns (model, view, and controller).
3. **Spring AOP (Aspect-Oriented Programming)**: Allows separation of cross-cutting concerns, such as logging and transaction management, from business logic.
4. **Spring DAO (Data Access Object)**: Provides a consistent way to access databases and manage transactions.

Why Use the Spring Framework?

Without using the Spring Framework, developers would have to manually manage object creation, dependency injection, and other repetitive tasks, which can lead to tightly coupled code and difficulties in managing dependencies. The Spring Framework offers several benefits:

- **Dependency Injection (DI)**: Reduces the need for boilerplate code to manage object dependencies.
- **Loose Coupling**: Promotes better design by decoupling components, making the code more modular and easier to maintain.
- **Improved Testability**: Facilitates easier unit testing by allowing mock implementations of dependencies.
- **Configuration Flexibility**: Supports XML-based configuration as well as Java-based annotations and configurations.
- **Comprehensive Infrastructure Support**: Includes transaction management, data access, messaging, and more.

Spring IOC (Inversion of Control)

Spring IOC is a core module of the Spring Framework. It is responsible for managing the lifecycle of beans and their dependencies. Here are some key concepts related to Spring IOC:

1. **Beans**: In Spring, objects are called beans. These are created, managed, and destroyed by the Spring container.
2. **Spring Containers**: There are two main containers in Spring that manage beans:
    - **BeanFactory**: The simplest container providing basic DI capabilities.
    - **ApplicationContext**: A more advanced container providing additional features such as event propagation, declarative mechanisms to create a bean, and more.
3. **XML and Annotation-Based Configuration**: Spring allows configuring beans and their dependencies using XML files or annotations.

Java Interfaces

1. **Interfaces**: A collection of abstract methods (methods without a body) that can be implemented by classes.
    - All methods in an interface are implicitly public and abstract.
    - Implementation of these methods is done in the classes that implement the interface using the implements keyword.
    - Interfaces can have default methods (with a body) and static methods (since Java 8).

Serializable Interface

1. **Serializable Interface**: A marker interface in Java.
    - It has no methods or fields.
    - It is used to indicate to the JVM that an object can be serialized (converted into a byte stream).
    - Serialization is the process of converting an object into a byte stream to save it to a file, send it over a network, etc.
2. **Marker Interface**: An interface with no methods or fields.
    - Used to convey some metadata to the JVM or other parts of the Java framework.
    - Examples include Serializable, Cloneable, and Remote.

Spring Framework and Dependency Injection

1. **Spring Framework**: A popular Java framework for building enterprise applications.

- **Inversion of Control (IoC)**: Spring's core concept, which means that the control of object creation and management is transferred from the application code to the Spring container.
- **Beans**: Objects that are managed by the Spring container.
- **Annotations**: Used to provide metadata to the Spring container. Examples include @Controller, @Service, @Repository, and @Component.

2. **Dependency Injection (DI)**: A technique where an object (dependency) is injected into another object rather than the dependent object creating the dependency itself.
    - **Types of Dependency Injection**:
        - **Constructor Injection**: Dependencies are provided through a class constructor.
        - **Setter Injection**: Dependencies are provided through setter methods.
3. **Bean Life Cycle**: The process through which a bean goes from creation to destruction.
    - **Bean Creation**: The bean is instantiated by the Spring container.
    - **Bean Initialization**: Dependencies are injected, and any initialization methods are called.
    - **Bean Usage**: The bean is used within the application.
    - **Bean Destruction**: The bean is destroyed, and any cleanup methods are called.
4. **Bean Scopes**: Defines the scope of a bean's lifecycle.
    - **Singleton**: A single instance per Spring container (default scope).
    - **Prototype**: A new instance is created each time it is requested.
    - **Request**: A single instance per HTTP request (used in web applications).
    - **Session**: A single instance per HTTP session (used in web applications).
    - **Global Session**: A single instance per global HTTP session (used in portlet-based web applications).

- **<u>Assignment: 19 JUNE 2024</u>**

1. Constructor-based Dependency Injection

In constructor-based DI, the dependencies are provided through the constructor of the class. This is a preferred method when the dependencies are required and the class cannot function without them.

2. Setter-based Dependency Injection

In setter-based DI, the dependencies are provided through setter methods. This approach allows the dependencies to be optional and can be changed after the bean is created.

3. Field-based (Property-based) Dependency Injection

In field-based DI, the dependencies are injected directly into the fields of the class using annotations. This method is the simplest to implement but is generally considered less desirable because it makes the code harder to test and violates the principle of immutability.

Annotations Overview

In the provided example, several annotations are used to manage the bean creation and injection:

- **@Component:** This annotation indicates that a class is a Spring component, and Spring should manage it as a bean.
- **@Autowired:** This annotation is used for automatic dependency injection. Spring's container will look for a bean of the required type and inject it where it is needed.
- **@Controller, @Service, @Repository:** These are specialized forms of @Component, indicating the role of the bean in the application:
    - **@Controller:** Used for defining a controller in the Spring MVC architecture.
    - **@Service:** Used for defining a service class (typically containing business logic).
    - **@Repository:** Used for defining a repository class (typically dealing with database operations).

ApplicationContext and BeanFactory

- **BeanFactory:** The simplest container providing basic DI support.
- **ApplicationContext:** A more advanced container, providing additional functionalities like event propagation, declarative mechanisms to create a bean, and various means to look up.

Meta Information

Annotations provide meta information to the Spring container to manage the beans. For example, @Component tells the container to create a bean for the annotated class. @Controller, @Service, and @Repository provide more specific meta information, but functionally they are similar to @Component.

Purpose of Constructor

In Java, a constructor is used to initialize an object. When a constructor is used for dependency injection, it ensures that the object cannot be created without its dependencies being provided.

@Autowired Annotation

- The @Autowired annotation is used in Spring Framework to enable automatic dependency injection.
- It can be used at the constructor, method, or field level to automatically inject the required bean.

Spring Framework Annotations

- @Component: Generic stereotype for any Spring-managed component.
- @Service: Specialization of @Component for service-layer classes.
- @Controller: Specialization of @Component for presentation layer (e.g., web controllers).
- @Repository: Specialization of @Component for persistence layer (e.g., DAO classes).

Dependency Injection Types

1. **Constructor-Based Injection**: Dependencies are provided through a class constructor.
2. **Setter Method-Based Injection**: Dependencies are provided through setter methods.
3. **Field-Based Injection**: Dependencies are provided directly on fields using annotations.

Spring Bean Scopes

- **Singleton**: One instance per Spring IoC container.
- **Prototype**: A new instance each time the bean is requested.
- **Request**: One instance per HTTP request (web-aware Spring ApplicationContext required).
- **Session**: One instance per HTTP session (web-aware Spring ApplicationContext required).
- **Application**: One instance per ServletContext (web-aware Spring ApplicationContext required).

Spring MVC Request-Response Flow

1. **Request**: The client sends a request to the server.
2. **DispatcherServlet**: Front controller receives the request and consults HandlerMapping.
3. **HandlerMapping**: Maps the request to the appropriate handler (controller).
4. **Controller**: Processes the request, interacts with the service layer, and returns a ModelAndView object.
5. **ViewResolver**: Resolves the logical view name to an actual view.
6. **DispatcherServlet**: Renders the view and sends the response back to the client.

Spring Boot Features:

- **Auto-Configuration:** Automatically configures Spring application based on the jar dependencies.
- **Starter Dependencies:** Simplifies dependency management (e.g., spring-boot-starter-web).
- **Embedded Server:** Built-in server (Tomcat, Jetty, etc.), no need for external server deployment.
- **DevTools:** Enables hot-swapping of code changes without restarting the server.
- **Actuator:** Provides production-ready features to monitor and manage application (e.g., health checks, metrics).
- **Profiles:** Facilitates different configurations for different environments (dev, test, prod).
- **Convention over Configuration:** Simplifies configurations by following default conventions.

Key Annotations in Spring Boot

1. @SpringBootApplication:

- This core annotation designates a configuration class that:

  - Declares one or more @Bean methods.

  - Triggers auto-configuration, component scanning, and additional configurations.

- Combines:

  - **@Configuration**: Enables class usage as a source of bean definitions by the Spring IoC container.

  - **@EnableAutoConfiguration:** Initiates bean addition based on classpath settings, beans, and property setups.

  - **@ComponentScan**: Facilitates component scanning for automatic discovery and registration of web controllers and other components in Spring's Application Context.

2. @RestController:

- A convenience annotation merging **@Controller** and **@ResponseBody .**

- Specifies that method return values directly populate the response body, rather than rendering a template.

3. @Controller:

- Identifies a class as a controller, typically used in web applications.

4. @ResponseBody:

- Directs a method's return value to the HTTP response body, often as JSON or XML.

5. @Repository:

- Labels a class as a Data Access Object (DAO) for interacting with the database.

- Provides benefits like automatic translation of database exceptions.

6. @Service:

- Denotes a class as a service provider for business logic.

7. @Autowired:

- Enables automatic dependency injection in constructors, methods, and fields.

8. @Component:

- A stereotype annotation indicating a Spring-managed component.

- Automatically detected through classpath scanning.

9. @Bean:

- Flags a method as a producer of a bean managed by the Spring container.

10. @Configuration:

- Identifies a class with @Bean definitions, processed by Spring to generate bean definitions and service requests at runtime.

11. @EnableAutoConfiguration:

- Activates Spring Boot’s auto-configuration mechanism.

12. @Conditional:

- Conditionally includes a bean based on specified conditions.

13. @Profile:

- Registers a component when specified profiles are active.

14. @ConditionalOnProperty:

- Conditionally enables or disables a bean based on the presence of a specific property.

15. @ExceptionHandler:

- Manages specific exceptions and sends customized responses to clients.

Additional Annotations

- @Entity:

- Marks a class as an entity, representing a database table.

Hibernate and Database Operations

1. Primary Key Generation:

- Hibernate offers mechanisms like @GeneratedValue to automatically generate primary keys for entities using strategies like IDENTITY, SEQUENCE, TABLE, etc.

2. Composite Key:

- Uses @EmbeddedId to map multiple columns as a composite primary key in an entity class.

Spring Framework Annotations

1. Associations and Relationships:

- Annotations like @OneToOne, @OneToMany, @ManyToOne, and @ManyToMany define relationships between entities.

2. Transactions:

- @Transactional ensures atomicity and consistency in database operations by demarcating transaction boundaries.

3. Security:

- @EnableWebSecurity enables security features such as authentication and authorization.

4. Scheduling:

- @EnableScheduling facilitates task execution at specified times or intervals.

5. Exception Handling:

- @ControllerAdvice and @ExceptionHandler centrally handle exceptions in Spring MVC applications.

6. HTTP Methods and Mapping:

- Annotations like @GetMapping , @PostMapping , @PutMapping , @DeleteMapping , and @PatchMapping map HTTP methods to controller methods.

7. Request and Response Handling:

- @RequestBody binds HTTP request data to method parameters, while @ResponseBody converts method return values to HTTP responses.

8. Component and Bean Annotations:

- @Component marks Spring-managed components, and @Bean explicitly declares beans in Java-based Spring configuration.

9. Profile Management:

- @Profile activates beans only in specific environments or profiles.Web Services and HTTP Protocol

1. **Web Services**: You explained the concept of web services, which facilitate communication between different systems or applications over the internet, typically using XML or JSON formats.
2. **RESTful Services**: You differentiated between SOAP-based (using WSDL) and RESTful web services, emphasizing the latter's use of HTTP methods (GET, POST, PUT, DELETE, PATCH) for CRUD operations.
3. **HTTP Protocol**: Briefly explained HTTP (Hypertext Transfer Protocol) and HTTPS (secure HTTP), highlighting their roles in client-server communication and data transfer over networks.

Miscellaneous Annotations

1. **Validation**: Annotations like @Validated and @NotNull (or @Required) were mentioned briefly for validating input parameters or fields in Spring applications.
2. **Configuration**: @Configuration and @ConfigurationProperties were mentioned for defining configuration classes and binding external configuration properties to Java objects in Spring.