Unit 7: Study of Frameworks

Unit/Module	Unit 7
Person	Austin Makasare
Status	complete

Introduction

- The main objective of a framework is to provide a common structure and set of tools for building software applications. This structure helps developers to write code more organized, maintainable, and reusable.
- A framework is a set of guidelines, libraries, and tools that provide a Structure for building software applications. It is a reusable set of code that developers can use to save time and effort when building new applications.
- Frameworks provide a standard way of doing things such as handling input and output, connecting to a database, or implementing security features. This helps developers to focus on the unique features of their application, rather than on the level details of how to perform common tasks.

Web Application Frameworks

A web application framework is a set of tools that helps developers to build web applications. Some core features of a web application framework include:

- **Routing**: Maps URLs to specific actions or resources within the application.
- **Templates**: Generates dynamic HTML or other markup bâsed on data passed to the application.
- **Database Integration**: Allows interaction with datqbases, like querying, inserting, updating and deleting records.
- **Form Handling**: Handles and validates form data submission and error handling
- **APT**: Allows the application to interact with other systems or applications through API calls.
- **Security**: Built-in support for securing the application and controlling access to resources or functionality.
- **Input Validation**: Validate user inputs and sanitize the data.
- **Session Management**: Manages user sessions.
- **Caching**: Improves performance by caching data or pages:
- **Error Handling**: Mechanisms for handling and reporting errors that occur within the application.

Types of Frameworks:

1. **Software Framework**: A software framework is a set of libraries, tools, and conventions that provide a structure for developing software applications. It serves as q foundation on which software developers can build, making it easier to develop, test, and maintain code.
2. **Web Application Frameworks:** A web application framework is a set of libraries and tools that are specifically designed to support the development of web applications. These frameworks provide a structure for building web applications, making it easier to handle common tasks such as routing, template rendering, and database integration.
3. **Mobile App Frameworks**: Mobile App Frameworks are a set of libraries and tools that are specifically designed to support the development of mobile applications for iOS or Android platforms. These frameworks provide a structure for building mobile applications, making it easier to handle common tasks such as UI design, device-specific functionality, and app deployment.

Frameworks as Reusable Chunks Of Architecture

Java Enterprise Edition (Java EE) is a set of technologies used to build enterprise-level applications in Java. It provides reusable components, frameworks, and services that help create multi-tier, scalable, and secure systems.

A key feature of Java EE is its **component-based architecture**, where applications are built using reusable parts such as:

- **JSF** (javaServer Faces) for user interfaces
- **EJBs** (Enterprise JavaBeans) for business logic
- **JPA** (java Persistence API) for database access

These components work together smoothly, making development easier.

Java EE also includes supporting technologies:

- **Servlet API** for handling HTTP requests
- **JSP** (javaServer Pages) for generating dynamic web pages
- **JTA** (Java Transaction API) for managing transactions

  It also offers built-in security features such as authentication, authorization, SSL, and role-based access control.

In summary, a framework acts as a reusable architectural block that simplifies building, testing, and maintaining applications. Java EE provides unified components and services that make enterprise application development easier and more secure.

The Framework Lifecycle, Development using Frameworks*

A **Spring bean** is an object managed by the Spring container. The container creates the bean, injects its dependencies, and manages its entire lifecycle. Beans are defined in the configuration file with properties like class name, ID, and dependencies. They can have different scopes, such as **singleton** or **prototype**. The Spring container ensures beans are created, initialized, and destroyed properly.

Bean Lifecycle

The **bean lifecycle** describes how Spring creates, configures, initializes, and eventually destroys a bean. The Spring container controls every step. The main stages are:

1. **Instantiation** : The Spring container creates an instance of the bean, by either using a constructor or a factory method.
2. **Dependency Injection:** Any required dependencies are injected into bean using constructor or setter injection.
3. **Configuration:** Additional settings are applied using methods like `setBeanName()`, `setBeanFactory()`, or `setApplicationContext()`.
4. **Initialization** – If the bean has an initialization method (e.g., `init-method` or `@PostConstruct`), it is executed.
5. **Ready to Use** – The bean is now fully initialized and ready to use.
6. **Destruction** – Before removal, Spring calls the bean's destroy method if defined (e.g., `destroy-method` or `@PreDestroy`).

Spring also supports lifecycle callbacks such as **InitializingBean** and **DisposableBean** for custom initialization and destruction logic.

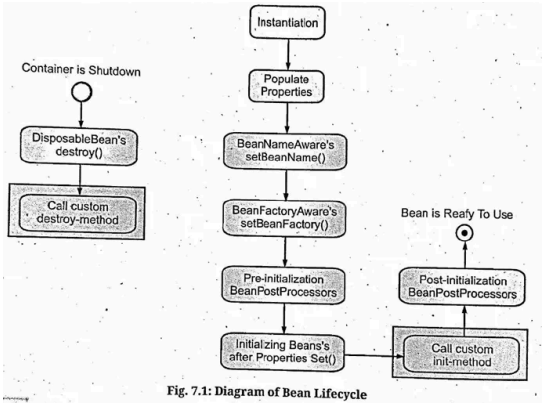


Fig. 7.1: Diagram of Bean Lifecycle

LifeCycle Callback Methods

Lifecycle callback methods allow a Spring bean to execute custom code **during initialization or destruction**. Spring provides special interfaces and annotations that let a bean respond to lifecycle events.

1. Initialization Callbacks: These methods run **after** the bean is created and dependencies are injected.

Common ways:

- `InitializingBean` **interface** → implements `afterPropertiesSet()`
- `init-method` attribute in XML configuration
- `@PostConstruct` annotation

Used for tasks like:

- opening resources
- setting up connections
- validating properties

2. Destruction Callbacks: These methods run **before** a bean is removed from the container (mainly for singleton scope).

Common ways:

- `DisposableBean` **interface** → implements `destroy()`
- `destroy-method` attribute in XML
- `@PreDestroy` annotation

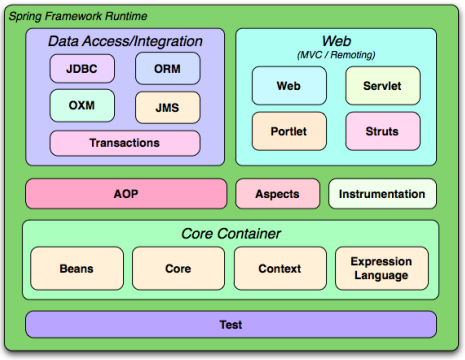
Used for tasks like:

- closing files
- releasing database connections
- cleaning up resources

Lifecycle callback methods let developers plug in **custom logic** during a bean's initialization or destruction. Spring supports this through interfaces, XML attributes, and annotations, making bean lifecycle management flexible and customizable.d

---

Spring Core Framework



Inversion of Control (IoC)

- Inversion of Control (IoC) Inversion of Control (IoC) is a design principle that aims to invert the flow of control in an application by removing the dependency of a component on its collaborators. It is also known as the "Hollywood principle" - "Don't call us, we'll call you
- In the **Spring Framework**, the **IoC container** is responsible for:
  - Creating objects (beans)
  - Injecting dependencies
  - Configuring beans
  - Managing their full lifecycle

This makes components **loosely coupled**, easier to test, maintain, and reuse because configuration is handled outside the objects.

Types of IoC

1. **Type 1**: Dependency Pull: In this type. the component asks -the container for an instance of the required dependency.
2. **Type 2**: Dependency Push: In this type, the container creates and injects the dependencies into the component.

Spring uses Type 2 (Dependency Push) through various Dependency Injection (DI) methods:

- Constructor Injection
- Setter Injection
- Field Injection

Dependency Injection

- Dependency Injection (DI) is a technique used in software development to manage the dependencies Of objects. In Spring Framework, it is implemented by a **BeanFactory** or **ApplicationContext**, which creates and manages the lifecycle of objects (beans) and injects their dependencies.
- There are three types of DI in Spring Framework: Constructor injection, Setter injection, and Field injection. This technique makes the code more flexible maintainable and easy to test.

Difference between IoC and DI:

- **Inversion of Control (IoC)** is a design principle that inverts the flow of control to remove direct dependencies between components.
- **DI** is the specific technique used to achieve IoC by passing dependencies to an object externally, rather than the object creating or looking them up itself

---

Spring AOP

- **Spring AOP (Aspect-Oriented Programming)** is a feature of the Spring Framework that helps modularize **cross-cutting concerns** such as logging, security, and transaction management into reusable units called **aspects**.
- It separates these concerns from the main business logic, making the code cleaner, more modular, and easier to maintain.
- Spring AOP is implemented using **AspectJ**, a powerful AOP framework that provides tools for defining aspects through **pointcuts**, **advice**, and **join points**.
- *Key Concepts*:
  - **Pointcut**: An expression that selects specific join points (places in the program) where an aspect should apply.
  - **Advice**: The action taken at a join point (e.g., before, after, or around a method call).
  - **Join Point**: A point in program execution, such as a method call or exception.
  - **Aspect**: A module that encapsulates a cross-cutting concern affecting multiple classes.

Spring DAO

- **Spring DAO (Data Access Object)** is a part of the Spring Framework that provides an easy, consistent, and abstract way to interact with various data sources like databases, LDAP, or file systems. It hides the low-level data handling code and offers a clean API for performing CRUD (Create, Read, Update, Delete) operations.
- Spring DAO simplifies database access, reduces boilerplate code, and ensures better exception handling.

1. **Template Classes**: Spring DAO provides template classes such as JdbcTemplate, HibernateTemplate and JpaTemplate to simplify the data access code and provide exception translation.
2. **DAO Support Classes**: Spring DAO provides support classes such as NamedParameterJdbcTemplate and SimpleJdbcTemplate to simplify the data access code.
3. **Exception Translation**: Spring DAO provides a convenient mechanism for translating checked exceptions thrown by the data •access code into runtime exceptions.
4. **Support for Transaction Management**: Spring DAO provides support for declarative transaction management and programmatic transaction management.
5. **Support for Stored Procedures:** Spring DAO provides support for calling stored procedures.

Spring ORM

Spring ORM (Object-Relational Mapping) is a feature of the Spring Framework that provides a consistent and abstract way of working with relational database using Object-Relational Mapping (ORM) frameworks such as Hibernate, JPA (Java Persistence API) and iBauS.

1. **Template Classes**: Spring ORM provides template classes such as HibernateTemplate and JpaTemplate to simplify thc ORM codc and provide exception translation.
2. **Support for Transaction Management**: Spring ORM provides support for declarative transaction management and programmatic transaction management.
3. **Support for Multiple ORM Frameworks**: Spring ORM provides support for multiple ORM frameworks such as Hibernate, JPA, and iBatis, allowing developers to choose the one that best fits their needs.
4. **Support for Data Access Exception Hierarchy:** Spring ORM provides a consistent exception hierarchy for data access exceptions, making it easy to handle and diagnose errors.
5. **Support for Lazy Loading:** Spring ORM provides support for lazy loading of associated objects, which' can improve the performance of an application by loading data only when it is needed.

JEE

JEE (Java Enterprise Edition) is a standard platform for building large-scale, enterprise-level Java applications. The **Spring Framework** is widely used alongside JEE to simplify development and provide additional features.

Spring offers several modules that support JEE development, such as:

- **Spring Web** → building web applications
- **Spring Data** → database access
- **Spring Security** → authentication and authorization
- **Spring Batch** → batch processing
- **Spring Integration** → connecting different parts of an application

These modules can work seamlessly with JEE technologies like **JSF**, **JSP**, **EJB**, and **JPA**, helping developers build powerful, scalable enterprise applications.

Web MVC

Spring Web MVC is a module of the Spring Framework used to build web applications based on the **Model-View-Controller (MVC)** pattern.

It includes components such as:

- **DispatcherServlet**
- **Controllers**
- **View Resolvers**
- **Validators**
- **Form Controllers**

These help handle requests, process data, validate inputs, and render views.

Spring Web MVC is built on the **Servlet API**, so it works smoothly with JEE web containers. It can also integrate with other Spring modules like **Spring Data** and **Spring Security** to develop complete enterprise-level web applications.

Spring Boot Framework

Spring Boot is a framework built on top of the Spring Framework designed to simplify the creation of stand-alone, production-ready applications.

- **Key Features:**
  - **Auto-configuration:** Automatically configures the application setup.
  - **Embedded Servers:** Includes built-in servers (like Tomcat), removing the need for external server deployment.
  - **CLI & Starter POMs:** Provides a Command Line Interface for quick project creation and "Starter POMs" for easily managing dependencies.
  - **Actuator:** A feature for monitoring application health and performance.

Benefits: It significantly reduces infrastructure configuration time, allowing developers to focus on business logic, which is particularly useful for building microservices

---

Spring and AOP

AOP complements Object-Oriented Programming (OOP) by focusing on "aspects" rather than classes to enhance modularity.

- **Core Concept:** AOP separates "cross-cutting concerns"—logic that affects the entire application, such as logging, security, and transaction management—into distinct modules.
- **Why Use AOP?**

    - **Dynamic Behavior:** It allows developers to add behavior (like notifications) dynamically before, after, or around core logic.
    - **Maintenance & Flexibility:** Instead of embedding code (e.g., logging) inside every method—which is hard to maintain—AOP defines it externally (e.g., in an XML configuration). This means changes, such as removing a notification, only require updating one file rather than modifying code across multiple methods.
- **Primary Use Cases:**
    - Declarative enterprise services, most notably **transaction management**.
    - Implementing custom aspects efficiently

DAO Support and JDBC Framework

DAO (Data Access Object) Support

The DAO pattern is a structural pattern that separates the data persistence logic into a distinct layer.

This pattern offers an abstract interface to various types of databases or other persistence mechanisms.

The primary benefit of using DAO is the separation of data access logic, which clearly distinguishes it from business logic;

Key Components:

1. DAO Interface: Defines the standard operations to be performed on model objects. Example methods may include save, update, delete, and find.
2. DAO Implementation: Offers the concrete implementation of the DAO interface. Includes the actual database operations using JDBC or another persistence mechanism.
3. Model Objects (Entities): Acts as a representation of the data being accessed and manipulated. Typically consists of JavaBeans or Plain Old Java Objects (POJOs).
4. Service Layer: Utilizes the DAO layer to execute business logic. This layer invokes DAO methods to interact with the database.

Benefits:

- Modularity: The DAO pattern encourages the separation of concerns by distinguishing data access logic from business logic.
- Maintainability. Simplifies the management and modification of database access logic without impacting other parts of the application.
- Testability: Eases unit testing of the data access layer through the use of mock objects.

JDBC (Java Database Connectivity) Framework

- JDBC is a Java technology designed for accessing databases.
- It offers an API for establishing connections to a database, executing SQL statements, and retrieving query results.

Key Components:

1. **DriverManager**: Oversees a collection Of database drivers. Establishes database connections by choosing the appropriate driver.
2. **Connection**: Represents a session with a particular database. Offers methods for creating Statement, PreparedStatement, and CallableStatement Objects.
3. **Statement**: Used to run Static SQL queries. Includes methods such as executeQuery for SELECT statements and executeUpdate for INSERT, UPDATE, DELETE, and DDL operations.

4. **PreparedStatcment**: An advanced version of Statement designed to execute precompiled SQL queries with parameters. Enhances performance and security, including protection against SQL injection•
5. **ResultSet**: Represents the result of a query. Provides methods for iterating over the returned data.
6. **SQLException**: An exception that offers details on database access errors or other related issues.

Step to use JDBC:

1. **Load the JDBC Driver**:
   Class.forName("com.mysql.cj.jdbc.Driver")
2. **Establish a Connection**:
   Connection connection = DriverManager.getConnection ("jdbc : mysql : / / localhost : 3306/dbname" ,"username", 'password");
3. **Create a Statement:**
   Statement = statement connection.createstatement();
4. **Execute a Query**:
   *ResultSet resultset = statement.executeQuery(" SELECT * FROM tablename");*
5. **Process the Result Set**:
   while (resultSet.next()) {
   System. out. println (resultSet. getString("columnname••)) ;
   }
6. **Close the Connection:**
   resultset. close() ;
   statement. close() ;
   connection. closet);

Benefits:

- *Database Flexibility*: JDBC API allows interaction with any database as long as it has a suitable driver:
- *Ease of Use:* It provides a simple interface for executing SQL commands and obtaining results.
- *Detailed Control*: It allows precise management of database transactions and operations

---

Spring and EJB

Spring Framework

The **Spring Framework** is a powerful and lightweight platform for building enterprise Java applications. It focuses on ease of use, flexibility, and modular design. Spring provides features such as **Dependency Injection (DI)**, **Aspect-Oriented Programming (AOP)**, and smooth integration with various technologies.

Key Features of Spring

- **Dependency Injection (DI):** Helps manage object dependencies and promotes loose coupling.
- **AOP:** Handles cross-cutting concerns like logging, security, and transactions without mixing them into business logic.
- **Modularity:** Offers multiple modules—Spring Core, Spring MVC, Spring Data, Spring Boot, etc.—for building modular applications.
- **Integration:** Works well with Hibernate, JPA, JMS, and many other tools.
- **Flexibility:** Independent of specific servers or environments; supports a wide variety of deployment models.
- **Spring Boot:** Accelerates development with auto-configuration, embedded servers, and production-ready features.

Common Use Cases of Spring

- Building **RESTful APIs**

- Developing **web applications** using MVC
- Managing **business logic and transactions**
- Integrating with **databases and external services**

Enterprise JavaBeans (EJB)

- EJB (Enterprise JavaBeans) is a server-side component framework within the Java EE (Enterprise Edition) ecosystem, designed to facilitate the development of distributed, transactional, and scalable enterprise applications.
- As part of the Java EE standard, EJB offers built-in features for managing transactions, ensuring security, and handling concurrency, making it suited for complex, large-scale applications.

Key Features:

- **Managed Beans**: EJBs are overseen by the EJB container, which manages their lifecycle, transactions, and security aspects.
- **Transaction Management**: EJBs come with integrated transaction management capabilities, supporting both container-managed transactions (CMT) and bean-managed transactions (BMT).
- **Security**: EJBs include declarative security features that enable developers to define security requirements at the method level.
- **Concurrency. EJBs handle** concurrent access to beans and address synchronization challenges.
- **Remote Access**: EJBS are designed to be accessible remotely, which is ideal for applications distributed across multiple locations.'

Common Use Cases of EJB

- Distributed and transactional business logic
- Large-scale, high-performance enterprise systems
- Applications needing strong transactional and security features

Comparison of Spring and EJB (Shortened)

1. Complexity

- **Spring:** More lightweight, flexible, and developer-friendly.
- **EJB:** More rigid and container-dependent, often more complex.

2. Configuration

- **Spring:** Uses annotations or XML; easier to configure.
- **EJB:** Relies on deployment descriptors and strict container rules.

3. Deployment

- **Spring:** Can run as stand-alone apps or in web containers with embedded servers (via Spring Boot).
- **EJB:** Must be deployed on a full Java EE application server.

4. Integration

- **Spring:** Integrates with many external frameworks and technologies.
- **EJB:** Mostly tied to Java EE technologies; may require extra configuration for other tools.

Advantages of Spring

1. **Loose Coupling:** Spring uses **Dependency Injection (DI)** and **Inversion of Control (IoC)** to reduce dependency between components, making applications easier to test, maintain, and modify.

2. **Modularity:** Spring is highly modular—developers can use only the required components instead of the entire framework.
3. **Portability:** Spring applications run on **Java SE**, **Java EE**, and any web/app server, making it platform-independent.
4. **Data Access Support:** Spring simplifies database operations with built-in support for **JDBC**, **Hibernate**, **JPA**, and other data access tools.
5. **Security:** Spring Security provides a powerful and flexible framework for securing applications and services.
6. **Web Services:** Spring includes built-in support for building and consuming both RESTful and SOAP-based web services.
7. **Testing Support:** Spring offers extensive testing tools like **Spring Test**, **JUnit**, and **TestNG**, making unit and integration testing easier.
8. **Scalability:** Spring is designed to handle large, complex, and high-performance applications efficiently.
9. **Community Support:** A large and active community provides continuous updates, documentation, libraries, and help.
10. **Flexibility:** Spring can be used for **small web apps**, **enterprise applications**, **microservices**, and more—making it suitable for any Java-based project.

Spring Boot Features

1. **Auto-Configuration:** Spring Boot automatically configures the application based on the dependencies added, reducing manual setup.
2. **Embedded Web Servers:** Comes with embedded servers like **Tomcat**, **Jetty**, or **Undertow**, allowing applications to run as **stand-alone** Java programs.
3. **Spring Boot CLI:** Provides a Command-Line Interface for rapidly creating, running, and testing Spring applications.
4. **Actuator:** Includes production-ready tools for **monitoring**, **metrics**, **auditing**, and **health checks**
5. **Starter POMs:** Offers pre-configured Maven/Gradle “starter” dependencies to simplify adding common features.
6. **Externalized Configuration:** Application settings can be stored in external files, making configuration easy to manage without modifying code.
7. **Testing Support:** Provides built-in support for **unit** and **integration testing**, including auto-configuration and mock object setup.
8. **Security:** Includes Spring Security integration for quickly securing applications with minimal configuration.
9. **Spring Data Integration:** Makes working with databases and data sources easier through seamless integration with Spring Data.
10. **Spring MVC Integration:** Allows easy development of **web applications** and **RESTful web services**.
