miniOrange First Week - Akhil Tomar

Maven is a build and dependency management tool. It simplifies project configuration and management by providing a standard structure and easing the process of downloading dependencies and building the project.

Key Concepts:

POM (Project Object Model):The pom.xml file is the heart of a Maven project. It defines project details, dependencies, plugins, and build configurations.

Dependency Management:Maven automatically downloads and manages project dependencies from repositories like Maven Central.

* Build Lifecycle: Maven follows a standard build lifecycle, with common phases like:
* clean: Cleans the project directory by removing previous build artifacts.
* compile: Compiles the source code.
* test: Runs unit tests.
* package: Packages the application (e.g., JAR or WAR file).
* install: Installs the package in the local repository.
* deploy: Deploys the application to a remote repository.

Spring Boot :Spring Boot is a framework built on top of Spring to simplify the development of Java-based applications. It reduces boilerplate configuration and promotes a "convention over configuration" approach.

Spring Initializer:A tool to quickly bootstrap a new Spring Boot project with the desired dependencies.

Annotation-Based Configuration:Annotations like @SpringBootApplication, @RestController, and @Service simplify bean definitions and configurations.

A bean in Spring is an object that is managed by the Spring IoC (Inversion of Control) container.A bean definition is the configuration that tells the Spring container how to create and configure an instance of a particular class.Beans are defined using annotations, XML configuration, or Java-based configuration.The Spring container reads the bean definitions and instantiates the required beans during the application context initialization.

Thymeleaf: Thymeleaf is a modern server-side Java template engine for creating dynamic web pages. It integrates seamlessly with Spring Boot for rendering views.

Git - Git is a distributed version control system that tracks changes in source code during development. It allows multiple developers to work on a project simultaneously, manage code versions, and collaborate efficiently.Git does not store full copies of files for each commit.Only the changes (deltas) between them are stored, making it highly efficient in space usage.

Conflict Detection:A conflict occurs when two changes overlap in the same file, in the same location, from different branches or commits.Git cannot automatically decide which change to keep, and it flags the file as conflicted.

Major Concepts in Git

1. Repository (Repo): A directory where Git tracks changes.Can be local (on your machine) or remote (hosted on a platform like GitHub, GitLab).
2. Commit: A snapshot of changes in the repository.Represents a specific point in the project’s history.
3. Branch: A separate line of development within a repository. The default branch is typically main or master.
4. Merge: Combines changes from one branch into another. Used to integrate feature branches into the main branch.
5. Clone: Creates a local copy of a remote repository.
6. Pull: Fetches changes from a remote repository and integrates them into the local repository.
7. Push: Sends local changes to a remote repository.
8. Staging Area: A temporary area where changes are prepared (staged) before committing.
9. Conflict: Occurs when changes in different branches overlap, requiring manual resolution.

OOP Session:

Encapsulation: Bundling data and methods that operate on the data within a single unit (class) while restricting direct access to some components.

Abstraction: Hiding implementation details and exposing only essential features to simplify complexity.

Inheritance: Reusing and extending existing class functionalities by creating hierarchical relationships.

Polymorphism: Allowing a single interface or method to behave differently based on the object or data it interacts with.

Classes and Objects: Classes are blueprints for creating objects, and objects are instances of classes representing real-world entities.

Dynamic Binding: Resolving method calls at runtime, enabling method overriding and polymorphism.

Locks: A synchronization technique used to manage access to shared resources. Prevents multiple threads or processes from accessing a resource simultaneously.

Types:Reentrant Lock: A thread can re-acquire the lock it already holds.

Mutexes (Mutual Exclusions): A type of lock specifically designed for mutual exclusion.Ensures only one process can access a resource at a time.Threads trying to acquire a locked mutex are blocked until it is released.

Semaphores: A signaling mechanism and synchronization tool that manages access to resources.Types:Counting Semaphore: Tracks multiple resources (can have a count > 1).Binary Semaphore: Similar to a mutex but with simpler behavior (count = 0 or 1).

Concurrency - The ability of a system to handle multiple tasks or threads that make progress independently of each other. Tasks are not necessarily executed at the same time but are managed to make efficient use of system resources.Example: A single CPU core running multiple threads using time-slicing.

Parallelism- The simultaneous execution of multiple tasks or threads on multiple CPU cores or processors. Tasks are executed at the same time.A multi-core processor running multiple threads, each on a separate core.

Exception Handling A mechanism to handle runtime errors or exceptional conditions gracefully.Try Block: Code that may throw an exception.Catch Block: Handles specific exceptions.Finally Block (optional): Executes cleanup code regardless of whether an exception was thrown.

JUnit: Widely used Java testing framework for writing and running unit tests which in turn ensures code correctness by testing individual components (units) of the application.Unit Testing: Testing a single, isolated piece of code, typically a method or class.Assertions: Used to verify the expected outcome (e.g., assertEquals, assertTrue).

Major Annotations:

@Test: Marks a method as a test case.

@BeforeEach: Runs before each test method; useful for setup.

@AfterEach: Runs after each test method; used for cleanup.

@BeforeAll: Runs once before all tests in the class; for global setup.

@AfterAll: Runs once after all tests in the class; for global cleanup.

@Disabled: Skips a test.

Mockito: A Java framework used with JUnit to create mock objects for testing dependencies.Simulates the behavior of complex objects and helps isolate the unit being tested.Mocking: Creating a fake implementation of a class or interface for testing purposes.Stubbing: Defining behavior for mock methods when specific conditions are met.

Major Annotations:

@Mock: Creates a mock object

@InjectMocks: Injects mock objects into the class being tested.

Flyway is a database migration tool that simplifies version control and schema management, ensuring consistency across environments. It supports migrations written in plain SQL or Java and can work with various databases like PostgreSQL, MySQL, Oracle, and SQL Server. Features like validation, rollback support, and integration with tools like Maven, Gradle, and Spring Boot are included. Flyway enhances database reliability, simplifies collaboration, and ensures smooth updates across development, staging, and production environments.

Project - Library Management System

1. Overall Objective

* Admin can:
  + Manage books by creating, deleting book records.
  + Manage users by deleting them or viewing user information.
* Users can:
  + Register and log in to their accounts.
  + View a list of available books.
  + Borrow books.

2. Workflow Components

(a) Models

1. Book: Represents the book entity with attributes like book id and book name.This is stored in the database table for books.
2. User: Represents the user entity with attributes like user email, user name and password.This is stored in the database table for users.

(b) Repositories

1. BookRepository: Interface extending JpaRepository to perform CRUD operations on the Books table.
2. UserRepository: Interface for interacting with the User table. Methods to find a user by username, save user details, delete a user, etc.

(c) Services

1. BookService:Contains the business logic for handling book-related operations.
2. UserService:Contains the business logic for user-related operations.

(d) Controllers

1. AdminController: Handles admin-related functionality, such as managing books and users.
2. UserController: Handles user-related functionality, such as login, signup, and borrowing books.

(e) Frontend Templates

* Thymeleaf templates are used for rendering the UI for both admin and users:
  + login.html: Login form for users and admins.
  + signup.html: Registration form for new users.
  + home.html: Home page showing available books for borrowing.
  + admin/users.html: Admin dashboard to manage users.
  + admin/home.html: Admin dashboard to manage books.

3. Workflow

For Users

1. Signup: User accesses /signup and fills out the registration form.The UserController validates and saves the user via UserService.
2. Login: User submits credentials to /login. The UserController verifies credentials using UserService and starts a session.
3. View Books: Logged-in users access /home to view a list of available books using BookService.
4. Borrow Books: Users borrow books via a button, which updates the book id in the database.

For Admin

1. Login: Admin logs in through /login with more access.
2. Manage Books:Admin can view, add, or delete books using BookService.
3. Manage Users:Admin can view all users and delete specific users using UserService.

4. Tools and Database

* Database: PostgreSQL, where Books and Users tables store data.
* Spring Data JPA: Handles database interaction.
* Thymeleaf: Dynamically renders data for views.

This is the summarized flow for managing books and users in your library management system. Let me know if you need more details on any specific part!

@Entity: Marks this class as a JPA entity that will be mapped to a table in the database.

@Table(name = "books"): Specifies the name of the database table (books) that this entity is mapped to.

@Id: Denotes the primary key field of the entity.

@GeneratedValue(strategy = GenerationType.IDENTITY): Specifies that the primary key (book\_id) is automatically generated by the database (e.g., using an auto-increment column).

@Column : By default, all fields in the entity are mapped to columns in the table unless explicitly excluded.

@Repository: Marks this interface as a Spring Data repository, indicating that it will handle data access and persistence for the Book entity.

JpaRepository: A Spring Data interface that provides built-in CRUD (Create, Read, Update, Delete) operations and pagination support.

@Service: Indicates that this class is a Spring Service component, typically used to define business logic.

@Autowired: Enables dependency injection by automatically injecting an instance of the BookRepository into the bookRepository field.

@Controller: Indicates that this class is a Spring MVC Controller, responsible for handling HTTP requests and returning views.

@RequestMapping("/admin"): Maps all HTTP requests starting with /admin to the methods in this controller.

@GetMapping("/users"): Maps HTTP GET requests for /admin/users to the viewUsers method.

@PostMapping("/book/add"): Maps HTTP POST requests for adding a book to the addBook method.  
@ModelAttribute: Binds the Book model object to the form data in the addBook method.  
@PathVariable: Captures the value from the URL and binds it to the method parameter (e.g., {id} is bound to id).

Session Management: HttpSession is used to retrieve the username of the logged-in user to check their authorization.

Form Submission: Data from forms is captured using @ModelAttribute for adding books and @PathVariable for capturing IDs in delete operations.