Unit 7 Advanced Topics in Java

Overview of ORM
Object-Relational Mapping or ORM is a technique for converting data between Java objects and relational databases. ORM converts data between two incompatible type systems (Java and MySQL), such that each model class becomes a table in our database and each instance a row of the table.
It is a piece of software designed to translate between the data representations used by databases and those used in object-oriented programming.ORM allows you to work with database-backed data using the same object-oriented structures and mechanisms you'd use for any type of internal data.

Why do we need ORM?
Loading and storing objects in a relational database exposes us to the following five mismatch problems (impedance matching):

1. Granularity: Sometimes you will have an object model, which has more classes than the number of corresponding tables in the database.
2. Inheritance : RDBMSs do not define anything similar to Inheritance, which is a natural paradigm in object-oriented programming languages.
3. Identity: An RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).
4. Associations: Object-oriented languages represent associations using object references whereas an RDBMS represents an association as a foreign key column.
5. Navigation: The ways you access objects in Java and in RDBMS are fundamentally different.

Advantages of ORM

- The connection between ORM and the database is not dependent on what the database is, making it easier to switch to another database.
- ORM automation reduces the likelihood of errors in most database operations.
- Since ORM supports many programming languages, you can use the database you want in the programming language you want.
- ORM supports custom queries, so you can use your own queries with ORM.
- Resolves object code and relational mismatch
- Using ORM, the development process is quite simplified as it automates object to table and table to object conversion which results in lower development and maintenance cost
- The code is less as compared to embedded SQL
- Gives an optimized solution that results in faster application and easier maintenance.

Disadvantages of ORM

- If we delete the entity we created while using ORM, the table in the database will not be deleted. Therefore, both sides must be operated in case of any discrepancies between the code and the database.
- Similarly, if the name of the existing entity changes, a new table will be created for that name. Since the old table will not be overwritten, data is stored in two different places, causing inconsistency.
- Due to the limited types that ORM accepts, you may have to store data in a more limited way.
- ORM architecture may cause performance loss.

JDBC vs ORM

ORM	**JDBC**
Little slower than JDBC	It is faster compared to ORM
SQL queries requirement is comparatively quite less however this doesn't mean that you have to do less work using ORM	SQL queries are required here
ORM technology makes it easy to store objects/data to database automatically without writing manual code	We have to write code manually to store objects/ data in the database
There are not many restrictions while dealing with data. Even a single database cell can be retrieved, changed, and saved.	JDBC comes with a lot of restrictions on extracting the result-set, processing it, and then committing it back to the database.

Some popular Java ORM are Hibernate, TopLink, MyBatis etc.

Hibernate

Hibernate is a Java persistence framework that simplifies the development of Java application to interact with the database. It is an open source, widely used, lightweight, ORM tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

Advantages of Hibernate

- It is an open source, lightweight, flexible, and powerful ORM tool.

- It supports database independent query (HQL).
- Its performance is fast because cache is internally used.
- It simplifies the complex join.
- If there is any modification in table or database only need to change in XML file properties.

What is the Object-Relational Impedance Mismatch?
The object-relational impedance mismatch is the mismatch between the way data is represented and manipulated in an object-oriented programming language, and the way it is represented and manipulated in a relational database. In an object-oriented programming language, data is represented as objects, which have properties and methods. These objects can be related to one another through inheritance, aggregation, and other relationships. In a relational database, data is represented as tables, with rows and columns. These tables are related to one another through foreign keys and other relationships.
The mismatch arises because the two models have different ways of representing and manipulating data. OOP is based on the principle of encapsulation, where the data and the methods that operate on that data are combined into a single unit, whereas relational databases are based on the principle of normalization, where the data is divided into separate tables in order to minimize data redundancy and improve the performance of queries.Furthermore, OOP is focused on the relationships of objects, whereas relational databases are focused on the relationships between data. This makes it difficult to map between the two models, as the concepts used in OOP such as inheritance, polymorphism, and encapsulation are not directly supported in relational databases.
Hibernate solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions.

Web Framework Introduction
Frameworks are tools with pre-written code, that act as a template or skeleton, which can be reused to create an application by simply filling with your code as needed which enables developers to program their application with no overhead of creating each line of code again and again from scratch. It To simplify\ies and streamlines the tedious task of building applications. Frameworks come with pre-built libraries, tools, and APIs that address common programming challenges. They provide standardized approaches and conventions for tasks such as web development and database interactions.

Difference

Libraries	**APIs**	**Frameworks**
Libraries are collections of pre-written code modules or functions that developers can use to perform specific tasks. They provide ready-made functionality that can be integrated into an application.	APIs define a set of rules and protocols that allow different software applications to interact with each other. They specify the methods, parameters, and data formats that applications can use to communicate and exchange information.	Frameworks are more comprehensive than libraries and provide a structured and reusable architecture for building applications. A framework dictates the overall design and flow of an application, often following a specific architectural pattern.

Some popular Java based web frameworks are Spring, Struts, Grails etc.

Spring Boot
Java Spring Framework (Spring Framework) is a popular, open source, enterprise-level framework for creating stand-alone, production-grade applications that run on the Java virtual machine (JVM).
ava Spring Boot (Spring Boot) is a tool that makes developing web applications and microservices with Spring Framework faster and easier through three core capabilities:

- Autoconfiguration
- An opinionated approach to configuration
- The ability to create stand-alone applications

These features work together to provide you with a tool for setting up a Spring-based application with minimal configuration and setup. Spring Boot applications can also be optimized and run with the Open Liberty runtime.

Features of Spring Boot
- Model-view-controller (MVC) framework
  It has a model-view-controller (MVC) framework that is specifically created to handle HTTP requests. In Spring: Once a request is received from a client, it gets sent to URL Handler Mapping (aka controller). The controller then passes this request to the model, which then sends it back to the controller. The controller then responds to the request through view.
  The benefit of using MVC is that it makes the entire process easy — you can easily debug issues (if you come across any), and the code is easy to navigate.

- **Aspect Oriented Programming (AOP)**

Aspect Oriented Programming (AOP) refers to the breaking down of program logic into distinct parts (aka modularization) to reduce crosscutting concerns (crosscutting concerns refer to the concern, that, if not centralized, will create issues with the whole application, e.g., concerns like security, transaction management, authorization, etc.).
Aside from that, it also allows you to maintain the code from one place. For example, if you've written code in five different places, you don't need to edit code from those five places any time a client asks for a revision; you can simply edit the code from a centralized place. However, if you need to make edits in only one place, you can do that too.

- **Dependency Injection (DI)**

Dependency Injection (DI) refers to the design pattern in Spring's Inversion of Control (IoC). This function was specifically created to remove dependency in code. There are two ways to insert DI in Spring, and those are the setter method and the constructor method. This function is particularly useful when you have to connect multiple objects to make a task run, but don't want the objects to be dependent upon each other.

Pros

- It is an extremely lightweight framework that can manage both small and enterprise-level projects.
- Using Spring Initializer, you can run any project in seconds.
- It has a thriving, active community, always ready to help out.

Cons

- Requires users to go through a steep learning curve. (Difficult to learn at first)
- Since there are no strict guidelines and many features, this may get overwhelming for developers.

Design Patterns

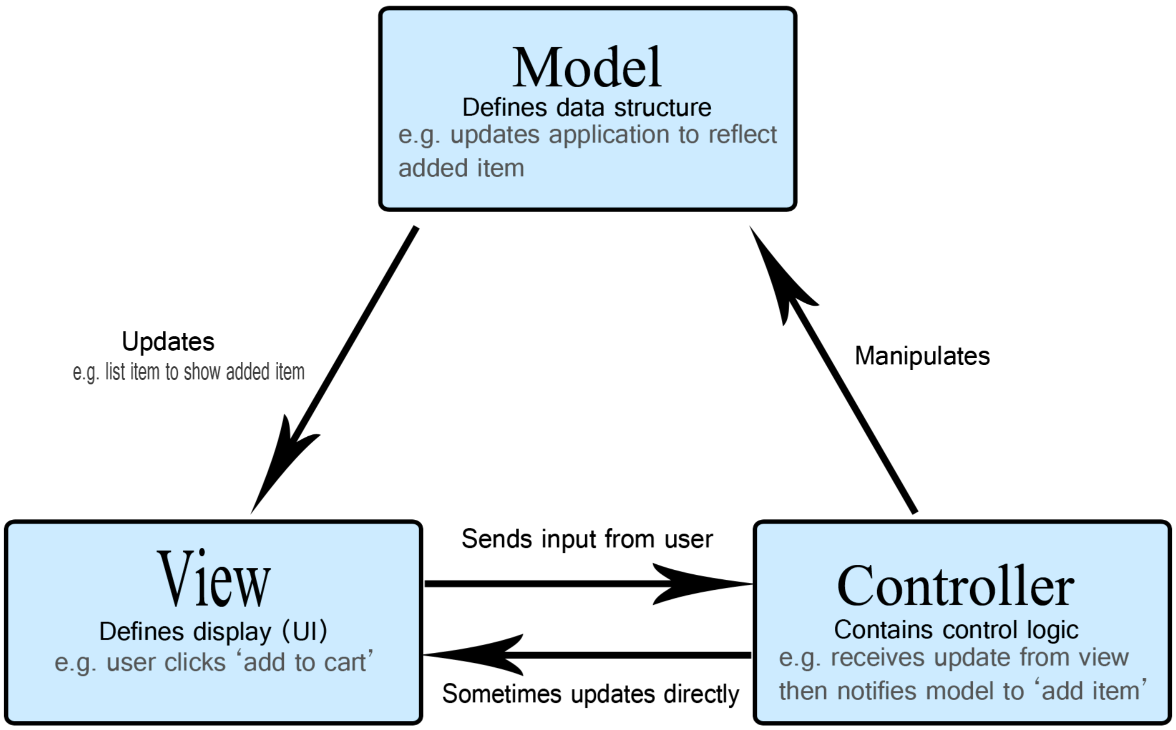
In software engineering, a software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that can be transformed directly into source or machine code.
There are mainly three types of design pattern: Creational, Structural and Behavioral.

MVC Design Pattern

The MVC design pattern is a software architecture pattern that separates an application into three main components: Model, View, and Controller, making it easier to manage and maintain the codebase. It also allows for the reusability of components and promotes a more modular approach to software development.
The Model View Controller (MVC) design pattern specifies that an application consists of a data model, presentation information, and control information. The pattern requires that each of these be separated into different objects.

- The MVC pattern separates the concerns of an application into three distinct components, each responsible for a specific aspect of the application's functionality.
- This separation of concerns makes the application easier to maintain and extend, as changes to one component do not require changes to the other components.

Components of MVC


Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

1. Model
The Model component in the MVC (Model-View-Controller) design pattern represents the data and business logic of an application. It is responsible for managing the application's data, processing business rules, and responding to requests for information from other components, such as the View and the Controller.

2. View
Displays the data from the Model to the user and sends user inputs to the Controller. It is passive and does not directly interact with the Model. Instead, it receives data from the Model and sends user inputs to the Controller for processing.

3. Controller
Controller acts as an intermediary between the Model and the View. It handles user input and updates the Model accordingly and updates the View to reflect changes in the Model. It contains application logic, such as input validation and data transformation.

Singleton

- Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the Java Virtual Machine.
- The singleton class must provide a global access point to get the instance of the class.
- Singleton pattern is used for logging, drivers objects, caching, and thread pool.
- Singleton design pattern is also used in other design patterns like Abstract Factory, Builder, Prototype, Facade, etc.
- Singleton design pattern is used in core Java classes also (for example, java.lang.Runtime, java.awt.Desktop).

Java Singleton Pattern Implementation
To implement a singleton pattern, we have different approaches, but all of them have the following common concepts.

- Private constructor to restrict instantiation of the class from other classes.
- Private static variable of the same class that is the only instance of the class.
- Public static method that returns the instance of the class, this is the global access point for the outer world to get the instance of the singleton class.

Sample Program (Singleton)

```
class Single{
    private static final Single obj = new Single();
    private Single(){}

    public static Single getInstance(){
        return obj;
    }
}

public class Singleton{
    public static void main(String [] args){
        Single one = Single.getInstance();
        Single two = Single.getInstance();
        System.out.println(one.hashCode());
        System.out.println(two.hashCode());
        if(one.hashCode() == two.hashCode()){
           System.out.println("Same object");
        }
    }
}
```

Factory Design Pattern
Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.
In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.

Advantages of Factory Design Pattern

- Factory Method Pattern allows the sub-classes to choose the type of objects to create.
- It promotes the loose-coupling by eliminating the need to bind application-specific classes into the code. That means the code interacts solely with the resultant interface or abstract class, so that it will work with any classes that implement that interface or that extends that abstract class.

Usage of Factory Design Pattern

- When a class doesn't know what sub-classes will be required to create
- When a class wants that its sub-classes specify the objects to be created.
- When the parent classes choose the creation of objects to its sub-classes.

Sample Program (Factory Pattern)
Step 1: Create an interface

```
public interface Shape {
   void draw();
}
```

Step 2: Create concrete classes implementing the same interface.

<u>Rectangle.java</u>

```
public class Rectangle implements Shape {

   @Override
```

```
  public void draw() {
    System.out.println("Inside Rectangle::draw() method.");
  }
}
```

Square.java

```
public class Square implements Shape {

  @Override
  public void draw() {
    System.out.println("Inside Square::draw() method.");
  }
}
```

Step 3: Create a Factory to generate object of concrete class based on given information.
ShapeFactory.java

```
public class ShapeFactory {

  //use getShape method to get object of type shape
  public Shape getShape(String shapeType){
    if(shapeType == null){
      return null;
    }
    if(shapeType.equalsIgnoreCase("RECTANGLE")){
      return new Rectangle();

    } else if(shapeType.equalsIgnoreCase("SQUARE")){
      return new Square();
    }

    return null;
  }
}
```

Step 4: Use the Factory to get object of concrete class by passing an information such as type.

```
public class FactoryPatternDemo {
  public static void main(String[] args) {
    ShapeFactory shapeFactory = new ShapeFactory();

    //get an object of Rectangle and call its draw method.
    Shape shape2 = shapeFactory.getShape("RECTANGLE");

    //call draw method of Rectangle
    shape2.draw();

    //get an object of Square and call its draw method.
    Shape shape3 = shapeFactory.getShape("SQUARE");

    //call draw method of square
    shape3.draw();
  }
}
```