upGrad

#LifeKoKaroLift

Session 3 | Data Access Layer: Entities

upGrad

Course: Developing RESTful Web APIs Using Spring Boot

Lecture On: Data Access Layer: Entities

Instructor: Vishwa Mohan

Topics covered in the previous class...

1. Introduction to Spring Boot and its various features
2. Creating a Spring Boot application using the Spring Boot Initializr
3. Different components of a Spring Boot pom.xml file and Dependency Management
4. Embedded servers and containers
5. Spring Boot Auto-Configuration and @SpringBootApplication
6. Custom Configuration using application.properties file and Profiles
7. Developing a Spring Boot Application and deploying it as JAR and WAR
8. Using DevTools to improve the runtime efficiency

Poll 1 (15 seconds)

Which of the following is the most convenient way to provide custom configuration while dealing with hierarchical data?

A. application.properties file

B. Command-line arguments

C. Java System Properties

D. YAML files

Poll 1 (15 seconds)

Which of the following is the most convenient way to provide custom configuration while dealing with hierarchical data?

A. application.properties file

B. Command-line arguments

C. Java System Properties

D. YAML files

Homework Discussion

https://github.com/ishwar-soni/SpringBootHomework

Today's Agenda

- **Create Entities to Interact With the Database**
  - What is Layered Architecture and how it will be implemented in the Movie Booking Application
  - Understanding the issues in designing Data Access Layer using JDBC
  - Object-Relational Mapping (ORM) and its implementation
  - JPA (Java Persistence API), Hibernate and Spring ORM
  - Setting up the connection with the database
  - Entity creation and basic annotations

Layered Architecture

Layered Architecture

upGrad

- In the previous session, you got introduced to Spring Boot and how its auto-configuration helps in reducing development time and effort for Spring applications.
- In this session, we will start building the back end of our Movie Booking Application, which would be a web application.
- Generally, a web application is designed using layered architecture.
- Let’s first understand what a web application is and how layered architecture helps you design a web application.

Layered Architecture: Web Application

upGrad

What Is a Web Application?

- An application that is installed on a remote machine and hosted using web servers such as **Apache Tomcat** and **Jetty** is known as a web application.
- It is accessed via a public network called the **Internet** through **HTTP protocol**.
- End users need to use a **web browser** to access a web application.
- Example: Facebook is a web application that is installed on a remote machine and hosted using a server whose address is https://www.facebook.com/, and it can be accessed using a web browser such as Google Chrome.

Layered Architecture: Segments

upGrad

- As web applications are hosted on servers and accessed via web browsers, they consist of the following two segments:
  - **Back-end segment**: This is the code that runs on the server and interacts with the database.
  - **Front-end segment**: This is the code that controls how the web pages will be rendered in the browser.

Layered Architecture: Back-End Segment

upGrad

Back-End Segment

- It is the part of a web application that runs on the web servers and interacts with the database.
- It accepts requests from the browsers, processes those requests using business logic, stores and fetches data from the database, prepares a response and sends the response back to the front end.
- It is designed/developed by back-end developers using server-side technologies such as Java and PHP or frameworks such as Spring and Node.

Layered Architecture: Front-End Segment

upGrad

Front-End Segment

- It consists of the **Graphical User Interface (GUI)** and is responsible for all the UI-related functional work.
- It helps users to send a request to the back end using GUI components and render the response from the back end on the web browser.
- It is designed/developed by front-end developers using client-side technologies such as **HTML, CSS** and **JavaScript** or frameworks such as Angular and React.

Layered Architecture

upGrad

- We can build the back end and the front end of a web application separately, which will interact with each other using HTTP requests and responses.
- We also know that we should build applications (back end and front end) in a loosely coupled manner so that they are maintainable.

Layered Architecture

upGrad

- Layered Architecture is one such technique that helps you build loosely coupled applications by breaking down an application into layers and organising its different components (Do you remember ‘components’?) into different layers based on the roles and responsibilities carried out by those components.
- All these layers (basically, the components inside the layers) will be connected to each other in a loosely coupled manner using the concepts of IoC and DI.

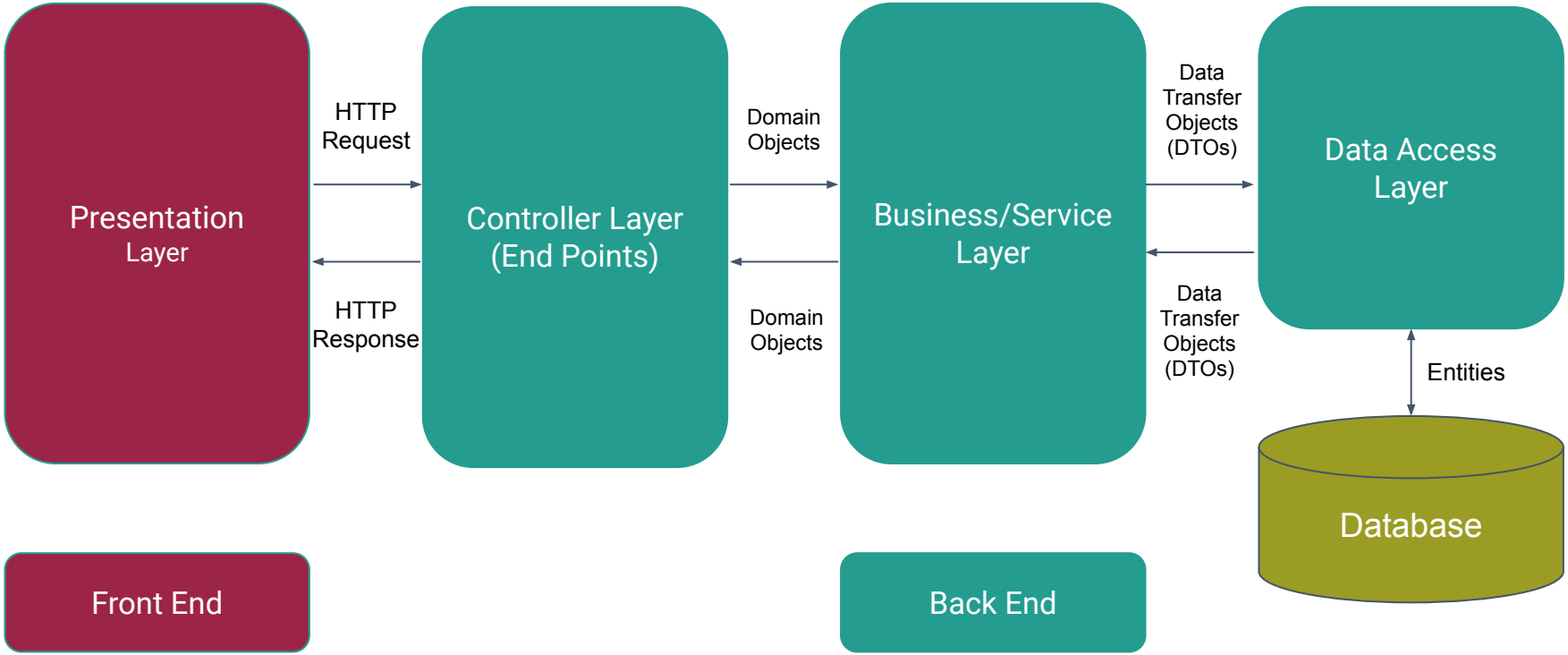
Layered Architecture

upGrad

- A web application is typically distributed into the following three layers:
  - Data access layer
  - Service/Business layer
  - Presentation/Controller layer
- Each of these layers has separate roles and contributes differently to the application.

Layered Architecture: Web Application

upGrad



Layered Architecture: Data Access Layer

upGrad

- This layer is part of the back-end segment.
- All the components (basically, beans) that are responsible for interacting with the database will be part of this layer.
- There will be a package inside our application that will contain all the classes whose objects will be used either for storing data into the database or for fetching data from the database.
- The components inside this layer will not be dependent on the components of the other two layers.

Layered Architecture: Service/Business Layer

upGrad

- This layer is also part of the back-end segment.
- It will consist of all those components that are responsible for processing the client request based on the business logic and preparing the response.
- There will be a separate package inside our application to contain all the classes whose objects will be used for performing the business logic.
- The components inside this layer use the components of the data access layer to process the client request and prepare the response. Thus, the components of the data access layer will be injected into the components of the service layer using the concepts of IoC and DI.

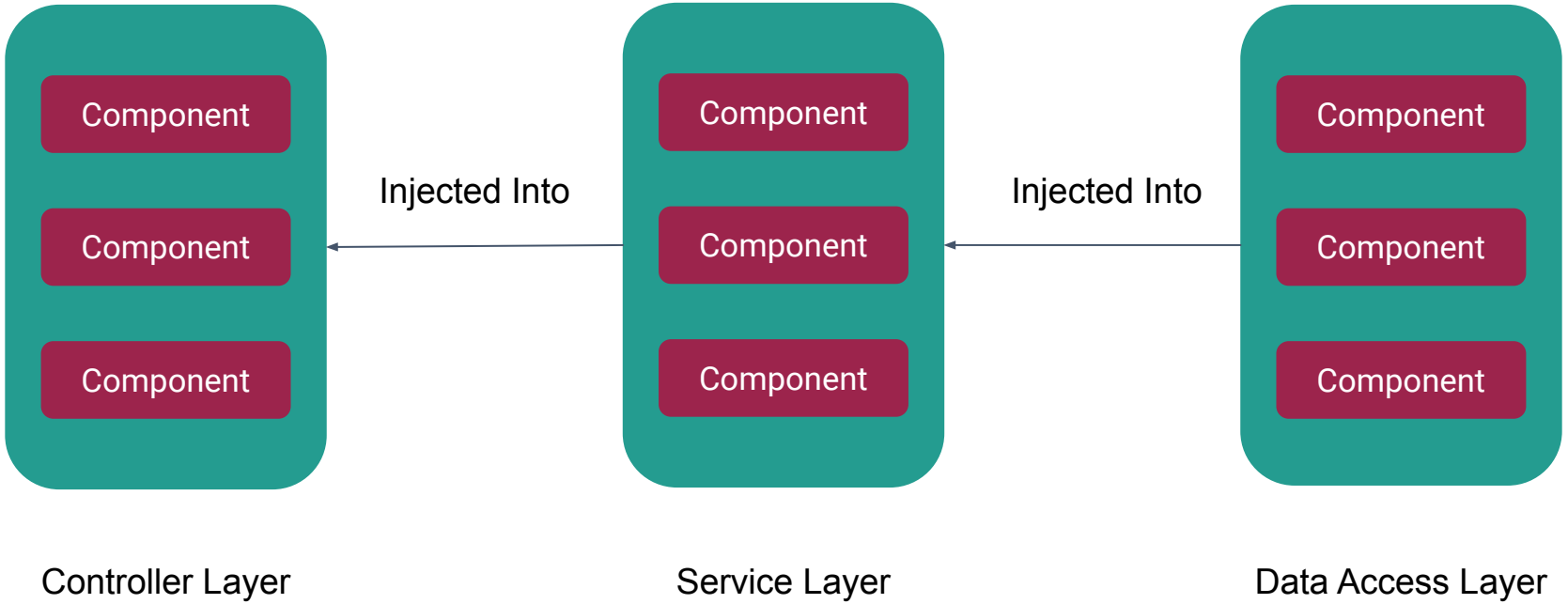
Layered Architecture: Presentation Layer

upGrad

- This layer is part of both the front end and the back end. In the back end, it consists of the controller layer or the end points, which are used by the front end to send the request to the back end and wait for the response.
- This layer will consist of all those components that expose the end points to the front end.
- There will be a separate package inside our application to contain all the classes whose objects will be used for exposing the end points.
- The components inside this layer use the components of the service layer to get the request processed and generate the response. Thus, the components of the service layer will be injected into the components of the controller layer using the concepts of IoC and DI.

Layered Architecture: Dependency Flow

upGrad



Layered Architecture

upGrad

Consider an online shopping website. The types of functions that are taken care of by the different layers are provided in the following table.

Front End	**Back End**	
Presentation Layer	Service/Business Layer	Data Access Layer
Displaying all product images using web page GUI Displaying all valid information about products on web page GUI	Selecting and adding products to the cart Calculating bills and discounts and generating purchase order invoice	Storing customer details such as billing and orders into the database Retrieving product information from the database Retrieving customer comments related to products from the database

Introduction to Movie Booking Application

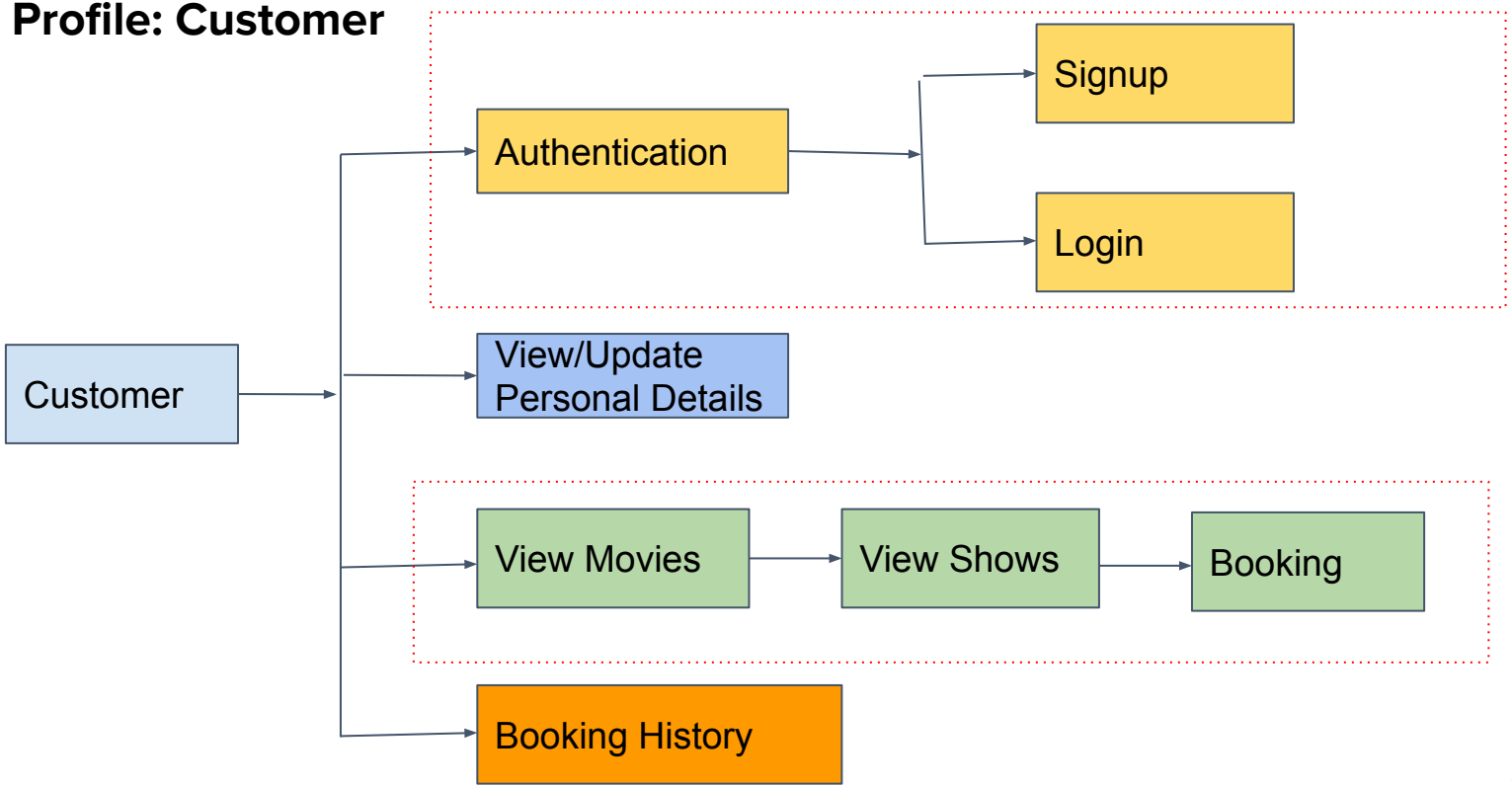
upGrad

- In this course, we will be developing the back end of the Movie Booking Application whose database was created in the previous course on ‘Foundations of Database’.
- This application will allow users to view movies and their respective showtimes and book tickets for the movies of their choice.
- The application will have the following two types of users:
  - Customers
  - Admins
- Let’s take a look at the various features of the application available to the different types of users.

Introduction to Movie Booking Application

upGrad

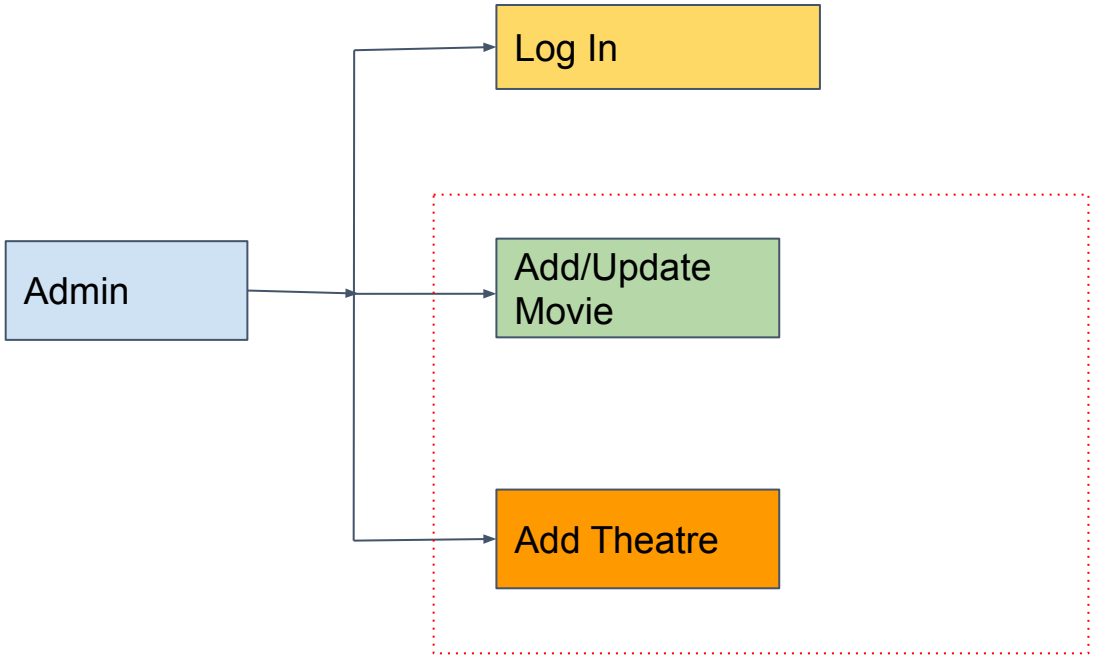
User Profile: Customer



Introduction to Movie Booking Application

upGrad

User Profile: Admin



Introduction to Movie Booking Application

upGrad

With reference to layered architecture, the part of the job that will be done by each layer of the Movie Booking Application is as follows:

- **Data access layer:** To store and fetch all the details related to the movies, users, bookings, etc., in the form of a database with different tables connected with each other
- **Service layer:** To perform the functions related to adding or updating movies and theatres, adding or updating users' personal details, booking seats for the movies, etc.
- **Presentation layer:** To perform authentication, view the details of movies and theatres, book the shows, display user details and booking history, etc.

Poll 2 (15 seconds)

Suppose you have to design a component that will be used to store or fetch user details from the database. Which of the following layers would this component be part of?

A. Data Access Layer

B. Service Layer

C. Controller Layer

D. Front-End Part of the Presentation Layer

Poll 2 (15 seconds)

Suppose you have to design a component that will be used to store or fetch user details from the database. Which of the following layers would this component be part of?

A. Data Access Layer

B. Service Layer

C. Controller Layer

D. Front-End Part of the Presentation Layer

Poll 3 (15 seconds)

There are three components A, B and C. A is part of the Data Access Layer, B is part of the Service Layer and C is part of the Controller Layer. Which of the following shows the correct direction of dependency injection? (X -> Y means X will be injected into Y.)

A. A <- B <- C

B. A -> B -> C

C. A -> B <- C

D. A <- B -> C

Poll 3 (15 seconds)

There are three components A, B and C. A is part of the Data Access Layer, B is part of the Service Layer and C is part of the Controller Layer. Which of the following shows the correct direction of dependency injection? (X -> Y means X will be injected into Y.)

A. A <- B <- C

B. A -> B -> C

C. A -> B <- C

D. A <- B -> C

Introduction to Object-Relational Mapping (ORM)

Introduction to ORM

upGrad

- Until now, you learnt that to design the back end, we have to design the three layers of the back end.
- So, with which layer we should start? Data Access Layer, Business Layer or Controller Layer?

Introduction to ORM

upGrad

- We will be starting with the Data Access Layer, as other layers are, directly or indirectly, dependent on this layer.
- Once we have designed the Data Access Layer, we will continue with the Service Layer, as the Controller Layer is dependent on the Service Layer.
- After the Service Layer is designed, we will design the Controller Layer to complete the back-end part of the movie booking application.

Introduction to ORM

upGrad

- Now that we have created the database for the Movie Booking Application, let's put it into use and connect it with the back end of our application.
- We can easily pass data between two Java applications, as all Java applications store data in the form of objects. For example, we can have an external JAR file attached to our Java application and call its method by passing objects and getting an object in return.
- However, the same is not true when a Java application is interacting with RDBMS. This is because, in Java, data is stored in the form of objects, whereas in RDBMS, data is stored in the form of rows inside tables; thus, there is a clear mismatch of data representation here.

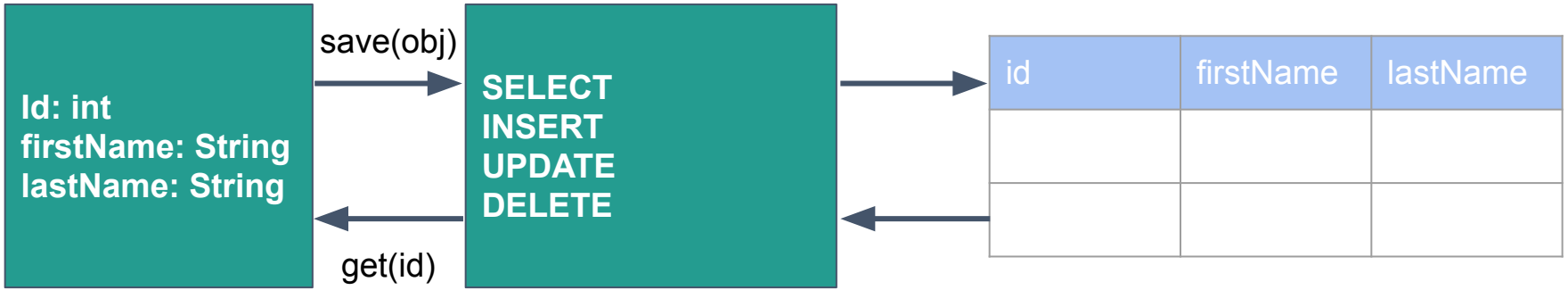
Introduction to ORM

upGrad

- You have to map Java objects to RDBMS table rows using the INSERT query and then map RDBMS table rows back to Java objects using the SELECT query with the help of JDBC API (Do you remember JDBC API?).
- For big applications, where a lot of tables exist, writing queries will decrease your productivity. It is also error-prone and has to be tested completely, which affects the productivity even more.
- Also, when you want to switch the database, you have to write all those queries for the new database.
- This is where ORM comes into play. It helps you by providing a well-tested code to map Java objects to RDBMS rows.

Introduction to ORM

upGrad



Java Object

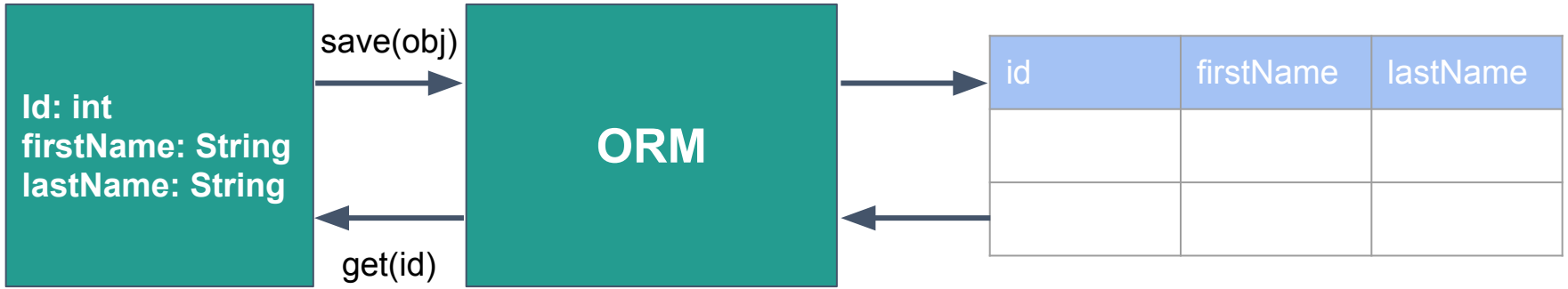
Manually Written Code

- Takes effort
- Error-prone
- Needs to be tested
- Needs to be rewritten when the database changes

RDBMS Table

Introduction to ORM

upGrad



Java Object

External Library (Jar)

- Takes no extra effort; simply add as dependency
- Extremely less chances of bugs
- Well-tested
- Just mention which database you are using; it will adapt accordingly

RDBMS Table

Introduction to ORM

upGrad

- What exactly is an ORM?
- According to Wikipedia, "*Object-relational mapping (ORM, O/RM, and O/R mapping tool) in computer science is a programming technique for converting data between* ***incompatible type systems*** *using object-oriented programming languages*".
- Here, the incompatible type systems would be Java and RDBMS.
- ORM allows easy transfer of data between RDBMS tables and the objects that are part of the application code.
- ORM not only helps you to convert data between Java objects and RDBMS table rows but also provides other features, such as table creation based on the entities (You will learn about this shortly.) present in the application.

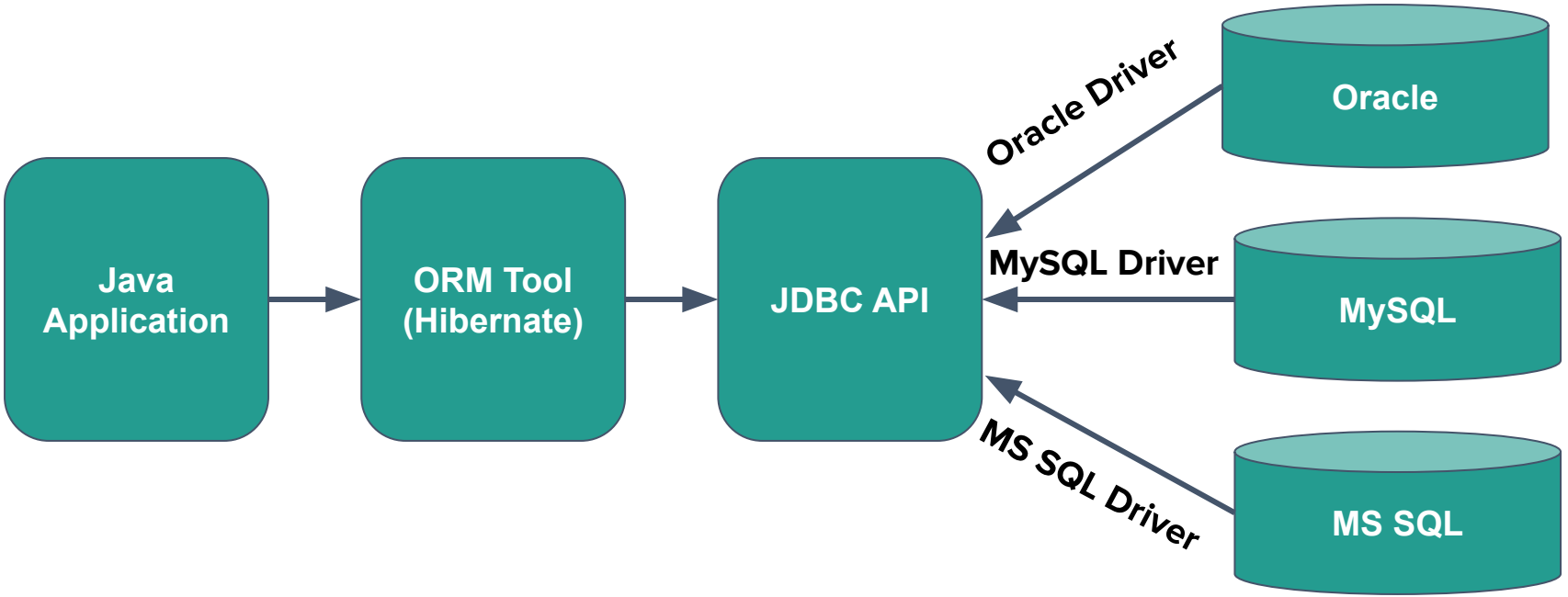
Introduction to ORM

upGrad

- You learnt that a lot of tasks that require developers' efforts while working with the plain JDBC API can be done automatically by ORM.
- However, ORM does not replace JDBC API but uses it internally. For example, when we are fetching data from the RDBMS table, ORM uses the SELECT query of JDBC API, gets the ResultSet and maps it to Collection objects.
- It supports primary key generation strategies based on the underlying RDBMS.
- It also supports database neutral SQL queries to operate on RDBMS.
- Examples of Java ORM frameworks are *Hibernate* (for RDBMS) and *Toplink* (for both RDBMS and non-RDBMS). In this course, we will be using Hibernate.

Introduction to ORM

upGrad



Introduction to ORM

upGrad

- In the previous diagram, the arrows represent the direction of the dependency.
- Thus, a Java application would be dependent on an ORM tool; an ORM tool would be dependent on JDBC API, and different drivers would also be dependent on JDBC API.
- Now, do you know why different drivers are dependent on JDBC API?

Introduction to ORM

upGrad

- The Dependency Inversion Principle (Do you remember this principle?) states that high-level modules (ORM) should not be dependent on low-level modules (JDBC Drivers), but both should be dependent on the abstraction (JDBC API).
- Hence, JDBC API provides only the interfaces, which are implemented by the JDBC Drivers.
- Thus, ORM uses JDBC API to interact with the RDBMS, and we can plug different JDBC Drivers based on the RDBMS that we are using.
- In future, if we want to change the RDBMS, we only need to unplug the current driver and plug the new driver, and ORM will manage the rest.

Poll 4 (15 seconds)

Which of the following statements is true with respect to ORM?
(Note: More than one option may be correct.)

A. ORM replaces JDBC API completely.

B. ORM internally uses JDBC API to interact with RDBMS.

C. ORM only works with RDBMS.

D. ORM performs the mapping between Java objects and RDBMS table rows.

Poll 4 (15 seconds)

Which of the following statements is true with respect to ORM? (Note: More than one option may be correct.)

A. ORM replaces JDBC API completely.

B. ORM internally uses JDBC API to interact with RDBMS.

C. ORM only works with RDBMS.

D. ORM performs the mapping between Java Objects and RDBMS table rows.

upGrad

JPA (Java Persistence API) and Hibernate

Java Specification

upGrad

- You learnt earlier that Java provides JDBC API as a collection of interfaces that is implemented by the JDBC Drivers.
- If JDBC API was not present, and if our code was directly dependent on the JDBC Drivers, then switching from one RDBMS to another would have been difficult, because our code would be tightly coupled with the JDBC Drivers.
- Also, JDBC Drivers may name methods differently. So, we would have to learn how to work with different drivers, which is not the case as of now.
- Thus, JDBC standardises everything while working with JDBC drivers by providing a set of interfaces and hence, our code will be written using the JDBC API interfaces.
- While running the application, we will plug the driver, and our application will start working with that database. Thus, JDBC API is a ***Java Specification***.

Java Specification

upGrad

- Specifications in Java define a set of interfaces, method declarations or annotations provided by the Java technology as a standard set of rules.
- Vendor companies must provide code implementation according to specifications, which ensures uniformity between different implementations.
- Some examples of specifications are as follows:
    - JDBC Specification: It is implemented by different RDBMS vendors such as Oracle and MySQL.
    - Servlet Specification: It is implemented by different Servlet containers such as Tomcat and JServ.

JPA

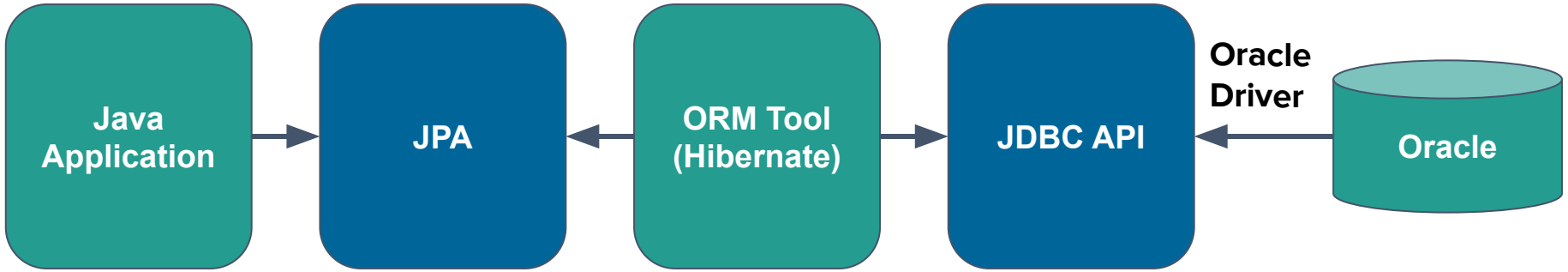
upGrad

- You learnt that we should not write code to implementations (concrete classes) but to specifications (interfaces).
- This means that your code should not contain any reference to the concrete classes. The properties, method parameters and return types of methods should be of the ‘interface’ type.
- But, while working with ORM tools, aren’t we coding again to implementations? Won’t it be difficult in future to change an ORM tool, for example, from Hibernate to TopLink?
- Does Java provide a specification for ORM as well?
- Yes, and that specification is called ***JPA (Java Persistence API).***

- Thus, we will be writing code to the specification (JPA) and plug the implementation as needed (Hibernate, for this course).
- The Hibernate code itself is written to the specification (JDBC API), so we have to plug its implementation as well (Oracle Driver, for this course).

Introduction to ORM

upGrad



Concrete Implementation

Specification

- In the previous diagram, you saw that no two concrete modules are tightly coupled with each other, as they are interacting with each other via a specification.
- Thus, JPA is a specification for an ORM technique, and Hibernate is one of its implementations, which is an ORM tool.
- Similarly, JDBC API is a specification to connect Java applications with RDBMS, and ojdbc is one of its implementations, which is a JDBC Driver that helps your Java application connect with Oracle Database.
- Note that you should always write code to specifications (or interfaces) to make your application loosely coupled.

Poll 5 (15 seconds)

Fill in the blank.

JPA is _____. (Note: More than one option may be correct.)

A. An ORM tool

B. An ORM specification

C. Provided by Java

D. Provided by Hibernate

Poll 5 (15 seconds)

Fill in the blank.

JPA is _____. (Note: More than one option may be correct.)

A. An ORM tool

B. An ORM specification

C. Provided by Java

D. Provided by Hibernate

Poll 6 (15 seconds)

Which of the following statements is true with respect to Hibernate?

A. Hibernate is a Java specification.

B. Hibernate is an ORM specification.

C. Hibernate is implemented by JPA.

D. Hibernate is dependent on JDBC API.

Poll 6 (15 seconds)

Which of the following statements is true with respect to Hibernate?

A. Hibernate is a Java specification.

B. Hibernate is an ORM specification.

C. Hibernate is implemented by JPA.

D. Hibernate is dependent on JDBC API.

Spring ORM

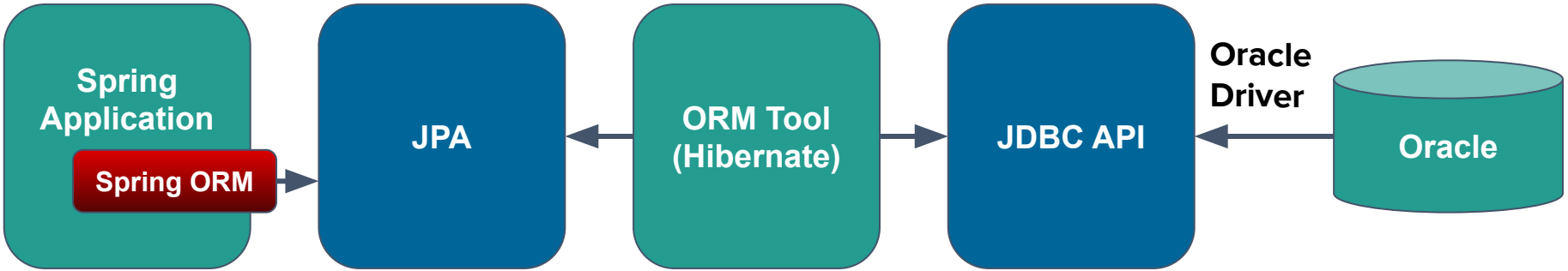
Spring ORM

upGrad

- Until now, you learnt that JPA is a Java specification (a set of interfaces), and Hibernate is one of its implementations (concrete classes that implement those interfaces).
- Hibernate helps you map Java objects to RDBMS table rows and vice versa. Basically, when you provide an object to Hibernate, it generates an SQL query using the data stored in that object and executes that query.
- However, it is not easy to integrate Spring applications with JPA or Hibernate and use them with all the benefits provided by Spring, such as IoC and DI.
- This is where Spring ORM comes into play.
- Spring ORM is a Spring module that helps you integrate your Spring applications with JPA or Hibernate such that you can use the ORM tools using IoC and DI.

Introduction to ORM

upGrad



Concrete Implementation

Specification

Entities

Database Setup

upGrad

- Let’s first set up the database so that we can run the code and see how Hibernate interacts with the database.
- As you learnt earlier, we will be writing code to the ***JPA specification*** and using ***Hibernate*** to run the show in the background.
- We also need ***spring-orm*** to integrate our application with JPA/Hibernate.
- Hibernate is itself dependent on ***JDBC API***.
- Also, we will need ***Oracle JDBC Driver,*** as we will be using Oracle Database.

Database Setup

upGrad

- To set up the database, we need the following dependencies:
  a. JPA specification
  b. Hibernate
  c. Spring ORM
  d. JDBC API
  e. Oracle JDBC Driver
- The first three dependencies can be added using only one starter POM, ***spring-boot-starter-data-jpa***.
- JDBC API is part of JDK, so there is no need to add it externally.
- Oracle JDBC Driver can be added using ***ojdbc8*** dependency. Now, let's add the dependencies.

Database Setup

upGrad

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
  <groupId>com.oracle.database.jdbc</groupId>
  <artifactId>ojdbc8</artifactId>
  <scope>runtime</scope>
</dependency>
```

Database Setup

upGrad

- We also have to provide certain properties to configure the database.

```
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=ishwar
spring.datasource.password=oracle

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.Oracle8iDialect
```

JDBC URL of the database.

Login username of the database.

Login password of the database.

To enable logging of SQL statements

Code Reference

Database Setup

upGrad

- The ***spring.jpa.hibernate.ddl-auto*** property is used to select the DDL mode for Hibernate. It can have one of the following values:
  - ***create***: It drops the existing tables and then creates the new ones.
  - ***update***: It updates the existing tables with the new changes done in the entity classes.
  - ***create-drop***: It **creates** and drops the tables in the end.
  - ***validate***: It matches the tables with the entity classes. If they do not match, then it throws an error.
  - ***none***: It turns off the DDL mode. This is also the default value.
- The ***spring.jpa.properties.hibernate.dialect*** property is used by Hibernate to decide for which database it has to generate queries. As we are using Oracle 11g Database, we need to use ***Oracle8iDialect***.

Poll 7 (15 seconds)

Which of the following datasource properties (spring.datasource.*) is **not** required to set up the database?

A. url

B. username

C. password

D. driver-class-name

Poll 7 (15 seconds)

Which of the following datasource properties (spring.datasource.*) is **not** required to set up the database?

A. url

B. username

C. password

D. driver-class-name

Entities

upGrad

- Now, let’s start with entity creation for the Movie Booking Application.
- In RDBMS, entities are objects that are stored as rows in the tables.
- Similarly, in JPA, entities are POJO (Plain Old Java Object) classes whose objects you want to save in the rows of the tables.
- For example, we want to save customer data into the database. Then, there would be a POJO class named ‘Customer’ in our Movie Booking Application.
- Now, let’s create the Customer class based on the customer table created in the database.

Entities

upGrad

- Entities in an application can be created using the following two different approaches:
  - Database-first
  - Entity-first
- In the Database-first approach, we first create the schema in the database. Then, we create the entire classes in the application. So, the table name would be the class name, and the columns would be the attributes of the class.
- In the Entity-first approach, we first create the entities in the application. Then, we use ORM tools, such as Hibernate, to create the schema for the application. So, the class name would be the table name, and the attributes of the class would be the columns in the table.
- Since we have already created a database for the Movie Booking Application, we will be using the Database-first approach.

Entities

upGrad

customer			
	customer_id PK	NUMBER(10)	PRIMARY KEY
	first_name	VARCHAR2(20)	NOT NULL
	last_name	VARCHAR2(20)	
	username	VARCHAR2(20)	NOT NULL, UNIQUE
	password	VARCHAR2(20)	NOT NULL, CHECK length>5 characters
	date_of_birth	DATE	NOT NULL
	user_type_id	NUMBER(10)	NOT NULL, FOREIGN KEY
	language_id	NUMBER(10)	NOT NULL, FOREIGN KEY

You can find the schema here. For now, let's ignore the foreign key mapping; this will be covered later in the course.

Entities

upGrad

- We have created a POJO class whose objects we want to save into the database.
- But, we will not be writing queries to save the objects into the database. Hibernate will be responsible for mapping this class to the table present in the database.
- So, we have to provide metadata about this class so that Hibernate can treat this class differently and map it to the table present in the database.
- Now, how do we provide metadata in our code?

Entities: @Entity

upGrad

- We use annotations to provide metadata in our code.
- Here, the annotation would be the **@Entity** annotation, which is used to mark POJO classes whose objects you want to save in the rows of the database tables.
- When you start the application, Hibernate will scan all the classes that are marked with the @Entity annotation and map them to the tables present in the database.
- As we have chosen ***create*** as the DDL mode, Hibernate will drop all the tables corresponding to the entities and create new ones.

Entities: @Entity

upGrad

```
@Entity
public class Customer {
    private int customerId;
    private String firstName;
    private String lastName;
    private String username;
    private String password;
    private LocalDateTime dateOfBirth;
    // getters, setters and toString
}
```

Entities: @Id

upGrad

- If you run the application now, then you will see that Hibernate is not creating any tables.
- This is because, every JPA entity must have a primary key, in the same way as every table in the database has a primary key.
- You can make any attribute of the entity class its primary key by marking that attribute with the **@Id** annotation.
- Thus, to create an entity class that can be mapped to a database table by Hibernate, there are two requirements as given below.
    - That class should be marked with the @Entity annotation.
    - One of its fields should be marked with the @Id annotation.

Entities: @Id

upGrad

```
@Entity
public class Customer {
    @Id
    private int customerId;
    private String firstName;
    private String lastName;
    private String username;
    private String password;
    private LocalDateTime dateOfBirth;
    // getters, setters and toString
}
```

Entities: @Table

upGrad

- Now, if you run the code, then you will see in the console that Hibernate will create a table corresponding to the Customer Entity.

  create table customer (customer_id number(10,0) not null, date_of_birth date, first_name varchar2(255), last_name varchar2(255), password varchar2(255), username varchar2(255), primary key (customer_id))

Entities: @Table

upGrad

- You will observe that:
  - The name of the table is the same as the entity class name ‘customer’;
  - All the attributes of the Customer class are treated as columns in the customer table;
  - The attribute names, which contain more than one word and were written using the lower camel case are not written using the underscore; and
  - int is mapped to number, String is mapped to varchar2 and LocalDateTime is mapped to date by Hibernate.

Entities: @Table

upGrad

- As you learnt earlier, Hibernate will map entities to the same table name in the database.
- If you want the table name to be different from the name of the entity class, then you need to mark the entity class with the ***@Table*** annotation.
- You can provide the table name using the ***name*** attribute of the @Table annotation.

Entities: @Table

upGrad

```
@Entity
@Table(name = "Cust")
public class Customer {
    @Id
    private int customerId;
    private String firstName;
    private String lastName;
    private String username;
    private String password;
    private LocalDateTime dateOfBirth;
    // getters, setters and toString
}
```

Entities: @Table

upGrad

- Now, if you run the code, then you will see that the table is created with the name ‘cust’.

  *create table **cust** (customer_id number(10,0) not null, date_of_birth date, first_name varchar2(255), last_name varchar2(255), password varchar2(255), username varchar2(255), primary key (customer_id))*

- As we want the table name to be the same as the entity name, let’s provide that name in the @Table annotation.

Entities: @Column

upGrad

- If you observe the ‘create table’ query, then you will notice that all the columns are created using the default values. But what if we want to apply certain constraints to our columns?
- For example, we want the username to be only 20 characters long. Also, it must not be null and must be unique.
- We can apply such constraints to the columns using the ***@Column*** annotation.
- Some of the most useful attributes of the @Column annotation are as follows:
    - ***name***: To provide custom name to the column inside the database table
    - ***length***: To specify the length of the column datatype
    - ***nullable***: To specify whether or not the column value can be null
    - ***unique***: To specify whether or not the column value is unique

Entities: @Column

upGrad

```
@Entity
@Table(name = "Customer")
public class Customer {
    @Id
    private int customerId;
    @Column(name = "first_name", length = 20, nullable = false)
    private String firstName;
    @Column(length = 20)
    private String lastName;
    @Column(length = 20, nullable = false, unique = true)
    private String username;
    @Column(length = 20, nullable = false)
    private String password;
    @Column(nullable = false)
    private LocalDateTime dateOfBirth;
    // getters, setters and toString
}
```

Entities: @Column

upGrad

- Now, if you run the code, then you will see that the table is created with the required constraints.

  create table customer (customer_id number(10,0) not null, date_of_birth date not null, first_name varchar2(20) not null, last_name varchar2(20), password varchar2(20) not null, username varchar2(20) not null, primary key (customer_id))

  alter table customer add constraint UK_irnrrncatp2fvw52vp45j7rlw unique (username)

Entities: @GeneratedValue

upGrad

- We have decided that ‘customerId’ would be the primary key for the customer table.
- However, we have not specified how the primary key would be generated. As you learnt, in Oracle Database, we have to use a sequence to generate primary keys, and here, we have not created any sequence so far.
- We can specify the strategy to be used to generate primary keys using the ***@GeneratedValue*** annotation.
- We can specify the primary key generation strategy using the ***strategy*** attribute of the @GeneratedValue annotation.

Entities: @GeneratedValue

upGrad

- The value of the strategy attribute can be one of the following four types:
  - ***IDENTITY***: It uses a database identity column (auto-increment). It is mainly used with MS SQL or MySQL.
  - ***SEQUENCE***: It uses a database sequence. It is mainly used with Oracle or PostgreSQL.
  - ***TABLE***: It uses an underlying database table to ensure uniqueness. It works with all databases.
  - ***AUTO***: It uses an appropriate strategy for the underlying database. This is default in case no strategy is specified. It is also the preferred type, as the ORM tool will choose the best strategy for the underlying database.

Entities: @GeneratedValue

upGrad

```
@Entity
@Table(name = "Customer")
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int customerId;
    @Column(name = "first_name", length = 20, nullable = false)
    private String firstName;
    @Column(length = 20)
    private String lastName;
    @Column(length = 20, nullable = false, unique = true)
    private String username;
    @Column(length = 20, nullable = false)
    private String password;
    @Column(nullable = false)
    private LocalDateTime dateOfBirth;
    // getters, setters and toString
}
```

Code Reference

Entities: @GeneratedValue

upGrad

- Now, if you run the code, then you will see that Hibernate will also create a sequence to generate primary keys.

  create sequence hibernate_sequence start with 1 increment by 1

- By default, the name of the sequence is ***hibernate_sequence***, and Hibernate uses the same sequence to generate primary keys for all the tables.

Entities

upGrad

Now, let's create the movie entity as well.

movie			
	movie_id	NUMBER(10)	PRIMARY KEY
	movie_name	VARCHAR2(50)	NOT NULL, UNIQUE
	movie_desc	VARCHAR2(500)	NOT NULL
	release_date	DATE	NOT NULL
	duration	NUMBER(3)	NOT NULL, CHECK (duration> 60)
	cover_photo_url	VARCHAR2(500)	NOT NULL
	trailer_url	VARCHAR2(500)	NOT NULL
	status_id	NUMBER(10)	NOT NULL, FORIEGN KEY

Entities

upGrad

```
@Entity
public class Movie {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int movieId;
    @Column(length = 50, nullable = false, unique = true)
    private String movieName;
    @Column(name = "movie_desc", length = 500, nullable = false)
    private String movieDescription;
    @Column(nullable = false)
    private LocalDateTime releaseDate;
    @Column(nullable = false)
    private int duration;
    @Column(length = 500, nullable = false)
    private String coverPhotoUrl;
    @Column(length = 500, nullable = false)
    private String trailerUrl;
}
```

Code Reference

Entities

upGrad

- Note that the length attribute of the @Column annotation does not work with integer values.

Poll 8 (15 seconds)

Choose the correct option from those given below.

How to provide custom names for the database tables corresponding to the entity classes?

A. Through @Table("Customer")

B. Through @Entity("Customer")

C. Through @Table(name = "Customer")

D. Through @Entity(name = "Customer")

Poll 8 (15 seconds)

Choose the correct option from those given below.

How to provide custom names for the database tables corresponding to the entity classes?

A. Through @Table("Customer")

B. Through @Entity("Customer")

C. Through @Table(name = "Customer")

D. Through @Entity(name = "Customer")

Poll 9 (15 seconds)

Which of the following annotations is a **must** for defining entity classes? (Note: More than one option may be correct.)

A. @Entity

B. @Id

C. @Table

D. @Column

Poll 9 (15 seconds)

Which of the following annotations is a **must** for defining entity classes? (Note: More than one option may be correct.)

A. @Entity

B. @Id

C. @Table

D. @Column

Poll 10 (15 seconds)

Which of the following primary key generation strategies works with Oracle Database? (Note: More than one option may be correct.)

A. GenerationType.AUTO

B. GenerationType.IDENTITY

C. GenerationType.SEQUENCE

D. GenerationType.TABLE

Poll 10 (15 seconds)

Which of the following primary key generation strategies works with Oracle Database? (Note: More than one option may be correct.)

A. GenerationType.AUTO

B. GenerationType.IDENTITY

C. GenerationType.SEQUENCE

D. GenerationType.TABLE

All the codes used in today's session can be found in the link provided below (branch session3-demo).

https://github.com/ishwar-soni/tm-spring-movie-booking-application/tree/session3-demo

upGrad

Important Concepts and Questions

Important Questions

upGrad

1. Can you explain the difference between JPA and Hibernate?
2. What problems does Hibernate solve?
3. If the database gets changed, then what are the configuration and code changes required in Hibernate?
4. How to configure entity classes to work with Hibernate?
5. Mention the steps to connect a Spring Boot application to a database using JDBC.
6. How is Hibernate chosen as the default implementation for JPA without any configuration?
7. Mention the dependencies needed to start up a JPA application and connect to Oracle Database with Spring Boot.
8. Where is the database connection information specified?

upGrad

Doubt Clearing Window

Today, you learnt about the following:

upGrad

1. Web applications are built in two segments: Front end and back end.
2. Back end consists of three layers: Data Access Layer, Service Layer and Controller Layer.
3. Introduction to the Movie Booking Application
4. ORM tools are used to map Java objects to database tables so that you can save and retrieve objects directly without writing SQL queries.
5. JPA is the ORM specification that lets you use ORM tools in a loosely coupled manner.
6. Spring ORM is a Spring module to integrate your Spring application with ORM tools.
7. You can create entity classes using the @Entity and @Id annotations.
8. Other useful annotations while working with entities are @Table, @Column and @GeneratedValue.

Homework

upGrad

In today's session, you learnt how to create entity classes using the database schema. Now, you should try to set up the database, create the entity classes for the following entities and run the application to see how Hibernate is mapping these entity classes to the database table.

1. UserType
2. Status
3. Theatre
4. City
5. Language
6. Booking

Use the following command to check out to the current state.
git checkout 66c418e

Homework

upGrad

You can get the schema from here. You can leave the foreign key mapping for now. This will be covered later in the course.

Code Reference

Tasks to Complete After Today's Session

MCQs
Coding Questions
Homework
Project Checkpoint 2

Next Steps

Checkout this code (**session3-homework** branch) before proceeding to the next session.

upGrad

#RahoAmbitious

Thank You!