Java 17

List of software’s required

1. JDK 17
2. Eclipse IDE

Java - platform independent and object oriented programming language

Customer - getDetails(), withdraw(), deposit()

Account - getBalance(), credit(), debit()

Two main building blocks of an object oriented language

1. class - template for an object
2. object - instance of a class

Java Fundamentals

* Datatypes & Variables
* Arrays
* Operators
* Conditional Statements
* Loops
* Classes & Objects

A class can have

1. variables
2. methods
3. constructors

Constructors: These names must be same as the class name, they are called when object is created, by default compiler creates a default constructor in case the class doesn’t have any explicit constructor, however if the class have any explicit constructor then compiler doesn’t create any constructor

OOPs principles

1. Inheritance - extends
2. Encapsulation - private data & public methods
3. Polymorphism - overloading & overriding
4. Abstraction - interface & abstract class

Day 2

* Inner classes - local, anonymous, nested, static inner classes
* Java 8 features - functional interface, lambda expression, default & static methods
* Packages & Access Specifiers like private, public, protected & no access specifier
* Factory design pattern to write a loosely coupled code
* Identify & Create the necessary java classes for your project (no need to create DB / Business logics right now) - create a separate java project for this

Day 3

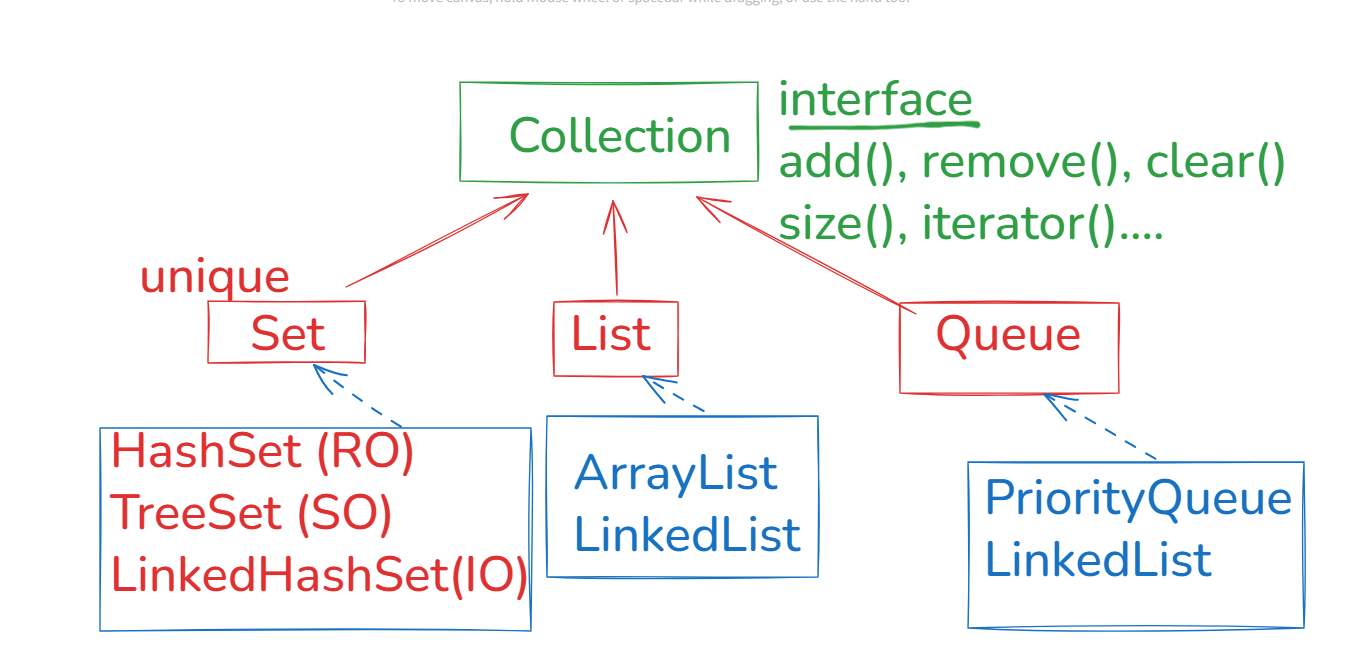
* Go through the TOC and try to cover all the topics that are in Day 3 & Day 4
* Important topics to cover that helps to solve the case study are below
* Exception Handling
* String processing
* Collection Framework
* Comparable & Comparator
* Map
* Using Collections with streams and applying lambda expressions

Note: Currently you can’t use DAO layer to interact with the Database, however collection acts like an in-memory database -> because it allows you to add/update/delete/read java objects

Collection Framework:

It provides all kinds of methods to maintain the data, its size is dynamic in nature

In Collection Framework there is a root interface called Collection



Queue -> Concentrates on processing the data while removing -> FIFO (LinkedList) -> Queue provides a method poll() to remove the element

PriorityQueue-> Removes the elements in sorted order

Main method -> View & Controller

View -> Scanner (Input) & System.out.println(..) (Output)  
 Controller -> based on conditions you can business logic

ex: 1 -> perform registration, 2 -> perform login

Service layer classes -> Business logics (some business logics might directly access DAO without having any code)  
DAO layer classes -> DB logics which are called from service layer

Java code wants to connect to the database (Oracle DB, MySQL, Postgres and so on)

JDBC Apis to interact with the database, these api’s are

1. Connection
2. Statement
3. PreparedStatement
4. CallableStatement
5. ResultSet
6. DriverManager

Steps to interact with the database

1. Load the JDBC driver (jdbc api implementation)

Class.forName(“oracle.jdbc.OracleDriver”);  
Class.forName(“com.mysql.jdbc.Driver”);

Note: From Java 8 onwards this is optional, because through jdbc url Java can load the drivers

1. Establish the connection

Connection con = DriverManager.getConnection(url, un, pw);

url = jdbc:oracle:thin:@localhost:1521:orcl (Oracle)  
url = jdbc:mysql://localhost:3306 (MySQL)

1. Create Statements

Statement stmt = con.createStatement();

1. Execute Statements

int count = stmt.executeUpdate(“insert into | update | delete…..”);  
ResultSet rs = stmt.executeQuery(“select \* from …”);

Note: You can use ResultSet methods to navigate over the sql results

1. Closing the resources

rs.close(); stmt.close(); con.close();

Project Setup

1. Create a java project / maven project
2. If its java project -> add jars by using build path, if maven project it will have pom.xml file that will download the jar files from the internet

Statement vs PreparedStatement

Statement is a super type for PreparedStatement, Statement is useful if the query is static, however if the query is dynamic you must use PreparedStatement

Statement -> “insert into employee values(“+id+”,”+name+”,”+dob);  
Statement stmt = con.createStatement();  
stmt.executeUpdate(“insert into employee values(“+id+”,”+name+”,”+dob);

PreparedStatement pstmt = con.prepareStatement(“insert into employee values(?,?,?)”);

pstmt.setInt(1, id);  
pstmt.setString(2, name);  
pstmt.setString(3, dob);  
pstmt.executeUpdate();

JDBC -> ORM

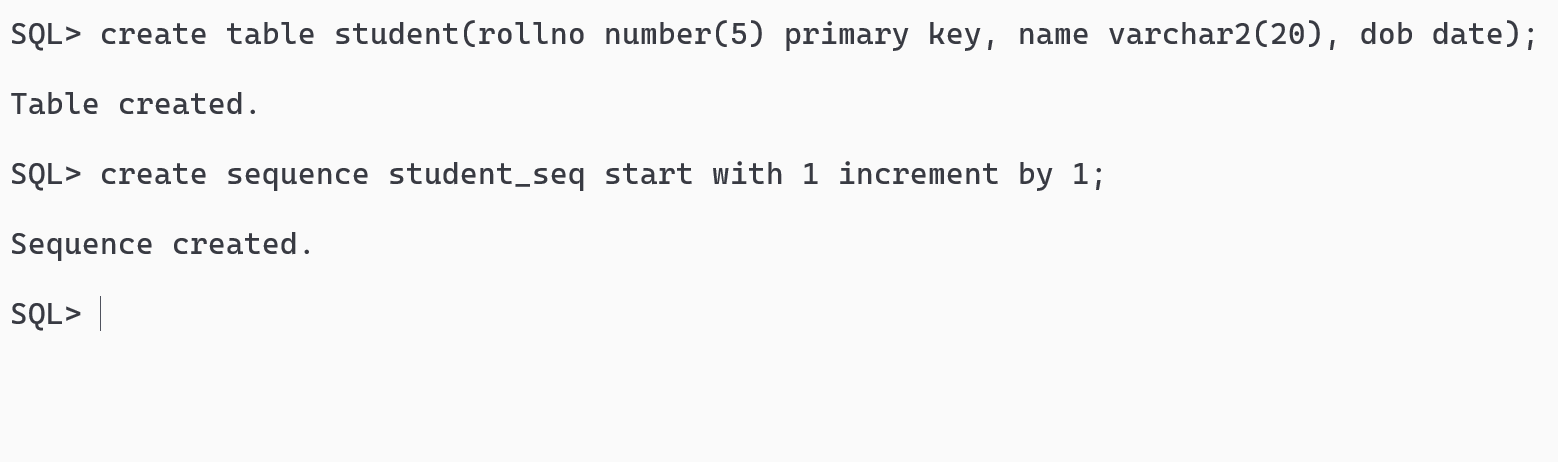
ORM (Object Relational Mapping) -> Here you can directly map java objects to the database table, here you don’t have to manually set values of each property and map to the table, instead you can directly persist the object, and ORM takes care of mapping each property into appropriate table using the metadata provided in the object class which is called as entity class.

JPA is the specification for the ORM that suggests what & how the object must be stored, JPA is a specification like an interface we need a provider i.e., like an implementation to work with the JPA, we have following providers for JPA

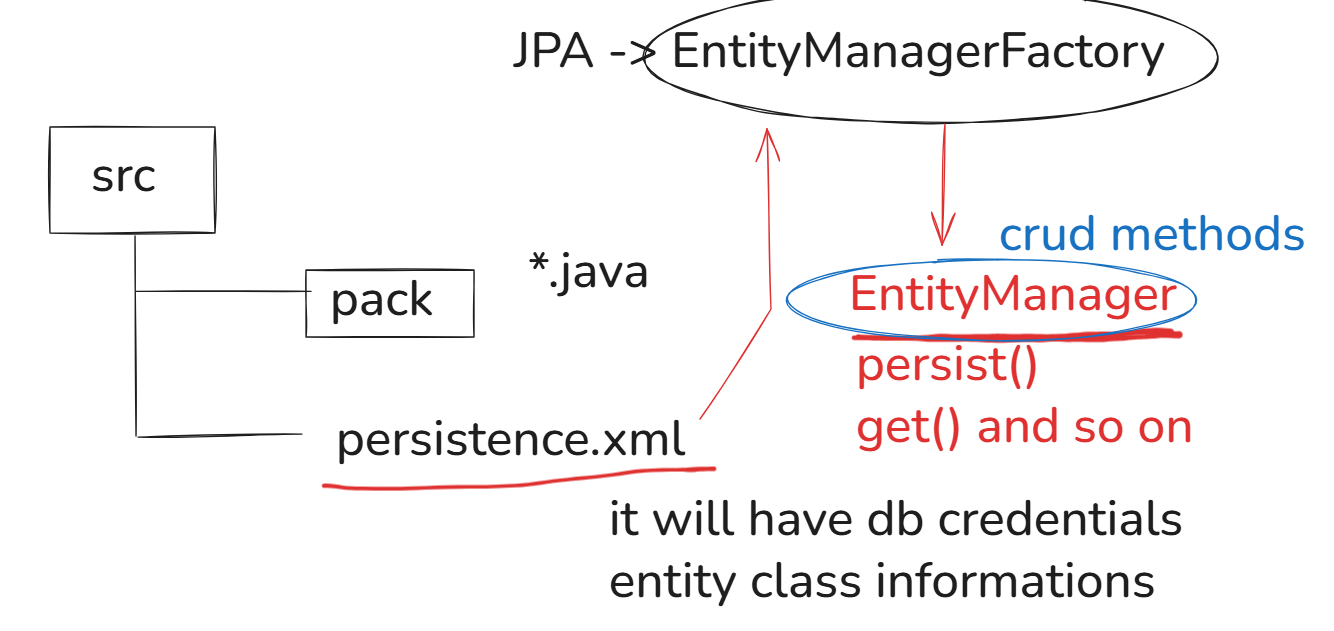
1. EclipseLink
2. Hibernate
3. Spring Data JPA
4. TopLink
5. iBatis

JPA is much simpler than the JDBC because it takes care of Exception handling, connections, type conversions

Create a table student & a sequence



JPA Project Structure



Steps to setup JPA Project

1. Convert Java Project to JPA Project
2. Select JPA 3.1 -> Generic 3.1 -> Eclipse Link 3.0.X -> Download the library
3. New Connection -> Select the driver definition -> Select the jar (ojdbc8) -> Enter the DB credentials -> Test Connection
4. Open persistence.xml -> Connection -> Change Transaction Type to Resource Local
5. In persistence.xml -> Source -> change javax to jakarta in the property name

JPA allows you to generate entity classes based on the table, we have student table for that it creates an entity class Student which looks as below

@Entity  
@Table(name=”Student”)  
class Student { … }

EntityManagerFactory factory = Persistence.createEntityManagerFactory(“jpa-demo”);  
EntityManager manager = factory.createEntityManager();  
EntityTransaction transaction = manager.getTransaction(); // required for DML operations  
manager.begin();  
manager.persist(new Student(…));  
manager.commit();

Self learning in JPA with eclipse-link

1. Perform CRUD operations using JPA
2. Implement the one to one, one to many, many to one & many to many associations in JPA using the annotations like @OneToOne, @OneToMany and etc.
3. Apply JPA in your project in the DAO layer
4. Make sure you are reusing EntityManager object creation instead of writing the code in every method.

Note: By Tuesday, Service & DAO layer must be ready, main method acts like View & Controller