**Roadmap of stuff I’ve learned**

**Projects:**

* **File Content Tree** – Read the contents of a text file and sort them based on a custom sorting order, store these sorted values in a Tree Object (TreeSet or TreeMap) without modifying the order and return the object.
* **JDBC CRUD Operations** – Establish connection with a MySQL database through Java JDBC and perform CRUD operations on this database.
* **Spring Boot JPA CRUD** – Create a simple Spring Boot Application which uses Restful services to store data in a repository and perform CRUD operations on this repository using Restful services, then implement Swagger configuration for this application and then write unit test cases for this application along with documenting the code and generating a Javadoc for this application.
* **Loan-App –** Given a Test use case, implement this application using whatever I had learned.

**Learnt:**

**Java:**

* **How to read line by line from a text file.**

This can be done using FileReader and BufferedReader.

Create a FileReader object to read from the file and enclose this object with BufferedReader, this allows us to read line by line from the file.

* **How to store and retrieve values from a HashMap**.

A HashMap is a collection of <Key, Value> pairs, it does not allow duplicate keys but allows duplicate values, to store a <Key, Value> pair use **put(Key, Value)**. To retrieve a value use **get(Key)**.

* **How to store and retrieve values from a TreeSet.**

A TreeSet is a collection which sorts every value inserted into it, values can be stored in a TreeSet using **add(Value)**. To retrieve values just print the TreeSet object.

* **Why Tree Collections in Java automatically sort values inserted into them.**

Tree Collections in java have an internal sorting mechanism which automatically sorts them in ascending order at the time of insertion, In case of TreeMap it is the **compareTo()** method and in TreeSet it is the **compare()** method.

* **How to override this sorting mechanism in order to implement my custom sorting mechanism.**

In order to implement our own sorting mechanism in TreeSet we need to override the **compare()** method in Comparator interface. The custom sorting logic can either be implemented inside the **compare()** method or outside it.

* **How to define a custom Exception and handle that Exception.**

Create an Exception class which extends RuntimeException and create a constructor which calls the immediate base class constructor. To handle the exception create an object which takes the exception as param and returns a response.

* **How to use lambda to call a method in HashMap.**

Set your method return type as Consumer or Producer and assign the method to a lambda.

**HashMap<Type, Type> name = new HashMap();**

**name.put(“key”, () -> object.methodname());**

* **Why we should be careful while using threads.**

Use threads cautiously as in a large enterprise application you cannot predict how a thread will run, this can lead to several issues later. Make sure you use threads as a last resort only.

**SQL:**

* **How to establish connection to a MySQL Database.**

Add the necessary mysql.jdbc.Driver.jar file to your library, then insert these two line into your java Class. Create a database in MySQL with custom name and set the username and password, pass these values to the Connection object in the URL.

**Class.forName("com.mysql.jdbc.Driver");**

**Connection con = DriverManager.getConnection( "jdbc:mysql://localhost:3306/yourDatabaseName?autoReconnect=true&useSSL=false","username","password");**

* **How to execute SQL queries in Java.**

Create a Statement object as below

**Statement stmt = con.createStatement();**

Store the SQL query you want to execute as a String and execute it using Statement object.

**stmt.execute(stored\_String);**

* **How Foreign Key mapping works in MySQL.**

A Foreign Key is mapped to a Primary Key in another table, Foreign Keys can contain duplicate values unlike Primary Keys. We cannot delete a Primary Key unless we delete all the Foreign Keys associated with it.

**Code Conventions:**

* **Why following code conventions are important and how to properly name classes, methods, variables, packages according to code conventions.**

This is important as not following these conventions makes maintaining your code much harder, having self-describing variable, method names can help other developers who read your code understand it much more clearly, the code convention standards are followed uniformly everywhere.

* **Why I should declare variables in global scope in a Class.**

Declaring variables in methods isn’t wrong but what if it needs to be used by another method, avoid declaring local variables unless necessary.

* **Why splitting up large code into methods is important (for readability).**

A method or Class with too many lines of code is much harder to read and understand. Splitting this code into several methods makes it look clean and is much easier to understand. Readability is important as it makes it much easier to debug, maintain and extend the logic.

* **How to manage redundant code.**

In case of redundant code enclose it in a separate method and call this method whenever you need you need it.

* **How to use constants to manage duplicate values.**

When you use repeated names or constant values in your code, you can declare them as **final static** variable and just call the variable wherever you use them.

* **Why we must follow proper design patterns.**

Design patterns are used to simplify your code, make it more flexible and easier to maintain, by following proper design patterns you are improving you code readability and maintainability.

* **Why packages are important and how to name packages according to Spring MVC.**

Packages improve readability and maintainability of your code. Packages must be named according to their role in the Spring architecture, (Controller, repository, model..etc).

* **How to maintain proper package structure.**

Always make sure you enclose your Classes and interfaces in separate packages, your interface implementation class package must be inside your interface package. Package names must be according to code conventions and test packages must have same names as main packages.

**SpringBoot, JPA and REST Services:**

* **What is Spring? And advantage of Spring**

Spring is a framework which provides support for several other frameworks such as Hibernate, Struts, JSF etc. Spring has several advantages which make it easier to test and maintain, the main one being it is very loosely coupled.

* **What are Restful Services? And the various REST service requests.**

A Restful Service is a Web Service architecture which uses Http requests to make calls between machines. The various REST service requests are:

GET – A read operation.

PUT – A write operation.

POST – A create/modify operation.

DELETE – A delete operation.

* **How to create a Spring Boot Application.**

Use <https://start.spring.io>. Select your version and dependencies and select generate, you will get a zip which contains your project which you can import into your preferred IDE.

* **How to add dependencies to Gradle.**

Goto <https://mvnrepository.com/> and search for the dependencies you want to add, select the version and select Gradle, copy the text and add it to your **build.gradle** file in your project.

* **What are the different repositories? And how to use these repositories.**

The repository I’ve used is crudrepository, this repository provides CRUD operations which I can use without having to define my own methods.

To use a repository create a repository interface for your model and implement crudrepository for it.

* **How to use Spring annotations and what they mean.**

Annotations define the behaviour of a Class or method. A Class with service annotation acts as a Service layer, methods with test annotation are Unit test methods.

* **How to implement JPA CRUD operations in Spring Boot.**

Crud repository has predefined methods for performing CRUD operations, create a repository that implements crudrepository and call the methods using this repository.

**Unit Testing using JUnit and Mockito:**

* **What is JUnit? And how to write unit tests for every method.**

Junit is a unit testing framework for Java, unit testing is a software testing method where individual units of source code are tested separately. Unit tests must be written in a separate test package which should follow proper design pattern structure and coding conventions as well.

* **What is Mockito? And how to use Mockito.**

Mockito is a mocking framework for unit testing Java applications, it is used to create a dummy implementation of the various methods used in our source code. Syntax is

**Mockito.when(method()).thenReturn(dummy\_output);**

* **How to initialize and inject Mocks.**

Initialize mocks with the @Mock annotation, injection of mocks is done with the @InjectMock annotation.

* **How to assert Test methods.**

Assertion is used to check whether the unit test has passed or failed, It checks the expected value with the actual value the unit test returns.

**assertEquals(expected\_value, actual\_value);**

* **How to generate unit tests for every method.**

IntelliJ can generate the test class for your source class, use this feature to generate the test methods you want to write unit tests for.

* **How to mock methods and the order of execution of tests.**

Methods used in source class can be mocked using Mockito, make sure you write the general mock cases in a setup() method annotated with @Before, for specific mock cases that fail conditional statements – write these within your @Test method. The structure of your @Test method must be:

Method\_name()

{

Mocks..

Call the source method you want to unit test…

Assert the values…

}

* **How to mock repositories.**

Create a repository test class and annotate with @DataJpaTest to use an embedded database for testing.

Autowire your repository and in setUp() initialize your dummy values in an object and save this object in your repository.

* **Why Autowiring repositories throws errors and how to resolve them.**

You cannot autowire your repository as the compiler will check for an implementation of your repo to create a Bean(which does not exist), hence you will need to mock a Bean using the @MockBean annotation. Annotate your repo object with both @Autowired and @MockBean(name = “repo name”) in this order.

* **What is MockBean.**

MockBean mocks a Bean which is useful for unit testing your repository methods.

* **How to add resources to test repositories.**

Create an application.properties file for your test package and then add the database and database properties you want to test.

**Other:**

* **How to return a JSON Response with custom response message.**

Create methods which return ResponseEntity objects, these objects are in format **ResponseEntity<Message, HttpStatus>,** this is the proper procedure while creating Restful services as you must return a HttpStatus for every operation along with a response message.

* **How to add comments to document my code.**

Adding descriptive comments in very important, every Class and method must be described so that any third-person accessing it can understand it.

Type /\*\* above the Class or method and press enter. IntelliJ automatically generates the parameters and return types which you can define later.

* **How to generate Javadoc.**

IntelliJ has a Generate Javadoc option under Tools, use this to automatically generate Javadoc for your code.

* **How to use Sonarlint to find issues and how to rectify those issues.**

Install SonarLint in IntelliJ and then run it for every class, it lists out the issues in your code in order of severity you can modify the code to resolve the issue.

* **How to return a JsonObject as response message.**

You cannot pass a JsonObject in ResponseEntity as it cannot be returned by ResponseEntity, to avoid this pass **JsonObject.toString()** instead.