Day-22

Name- Rishabh Manchanda

Task 1:

What do you mean by GOOD Code and BAD CODE?

Good code is readable and can be understood even by someone unfamiliar with the code. It is maintainable and easy to fix bugs. It can be broken into smaller and focused classes and can be reused.

A Bad code is messy and has inconsistent naming. Methods are long, usually doing too many things. Everything is in one class. It is difficult to test since it's tightly coupled. Also it uses a lot of static or direct calls.

10.40 to 10.44

Task 2:

What do you understand by databinding?

Data Binding is a programming technique that connects the UI with the data model, so that when one changes, the other updates automatically.

10.45 to 10.48

Task 3:

What do you know about continuous development?

It is aimed at delivering changes quickly, reliably, and frequently. It's a key part of DevOps and Agile methodologies.

It refers to a pipeline of processes that automate and streamline the entire lifecycle of software: from writing code to testing, building, releasing, and deploying.

Task 04

Polymorphism in Java means "many forms" allowing us to use one interface (or method) in different ways. It is a core OOP principle and has two types: compile-time (method overloading) and runtime (method overriding).

Compile Time

Achieved by defining multiple methods with the same name but different parameters in the same class.

Run Time

Achieved when a subclass provides a specific implementation of a method that is already defined in its parent class.

Task-5

What is, why is it used , where is it used..

TDD and BDD approach..

TDD is a software development process where tests are written before the actual code. The main steps in TDD are:

1. Write a Test: Before writing any functional code, you write a test for the new functionality you want to implement. This test will initially fail since the functionality does not exist yet.
2. Run the Test: Execute the test to confirm that it fails. This step ensures that the test is valid and that the functionality is indeed not implemented.
3. Write the Code: Write the minimum amount of code necessary to make the test pass.
4. Run the Test Again: Execute the test again to see if it passes. If it does, you know that the code works as intended.
5. Refactor: Clean up the code while ensuring that the test still passes. This step helps improve code quality and maintainability.
6. Repeat: Continue this cycle for each new piece of functionality.

In Java, TDD can be implemented using testing frameworks like JUnit or TestNG.

### **Behavior-Driven Development (BDD)**

BDD is an extension of TDD that focuses on the behavior of the application from the end user's perspective. It encourages collaboration between developers, QA, and non-technical stakeholders. The main steps in BDD are:

1. Define Behavior: Write specifications in a natural language format that describes the behavior of the application. These specifications are often written in a Given-When-Then format.
2. Automate Tests: Create automated tests based on the specifications. These tests verify that the application behaves as expected.
3. Implement Code: Write the code necessary to make the tests pass, similar to TDD.
4. Run Tests: Execute the tests to ensure that the application behaves as specified.
5. Refactor: Clean up the code while ensuring that all tests pass.
6. Repeat: Continue this process for new features or changes.

In Java, BDD can be implemented using frameworks like Cucumber or JBehave, which allow you to write specifications in a human-readable format.

Task  
  
List of Manual and automated testing tools

### **Manual Testing Tools**

1. TestRail: A web-based test case management tool that helps teams manage, track, and organize testing efforts.
2. JIRA: Primarily a project management tool, JIRA can also be used for bug tracking and test case management.
3. QTest: A test management tool that allows teams to manage their testing processes and track defects.
4. Postman: A popular tool for testing APIs manually, allowing users to send requests and view responses.
5. Selenium IDE: A browser extension that allows users to record and playback tests manually.
6. Bugzilla: An open-source bug tracking system that helps teams manage and track defects.
7. TestLink: An open-source test management tool that allows for the creation and management of test cases and test plans.

### **Automated Testing Tools**

1. JUnit: A widely used testing framework for Java applications, primarily for unit testing.
2. TestNG: A testing framework inspired by JUnit, designed to cover a wider range of testing needs, including unit, functional, and end-to-end testing.
3. Selenium: A powerful tool for automating web browsers, allowing for functional testing of web applications.
4. Cucumber: A BDD tool that allows writing tests in a natural language format, making it easier for non-technical stakeholders to understand.
5. Mockito: A mocking framework for Java that allows for the creation of mock objects in unit tests.
6. RestAssured: A Java library for testing RESTful web services, making it easy to validate API responses.
7. Appium: An open-source tool for automating mobile applications on both Android and iOS platforms.
8. JMeter: Primarily used for performance testing, JMeter can also be used for functional testing of web applications.
9. Postman (Newman): While Postman is used for manual API testing, Newman is its command-line companion that allows for automated testing of APIs.
10. QUnit: A JavaScript unit testing framework that can be used for testing JavaScript code in web applications.