# **Day 22 - 1 August 2025**

**Document Name:**Day 22 - hmuvvala@ - Hari Gopal Muvvala

### Task 1: What do you mean by GOOD Code and BAD CODE?

#### Good Code:

* **Readable**: Easily understood by others (or by you in the future)
* **Maintainable**: Easy to update or fix bugs later
* **Well-structured**: Follows consistent coding standards and naming conventions
* **Modular**: Broken into small, reusable methods/classes
* **Follows SOLID principles**: Ensures good object-oriented design
* **Testable**: Easy to write unit tests for
* **Efficient**: Uses optimized logic for time and space
* **Documented**: Has comments where needed (not over-commented)

#### Bad Code:

* **Unclear logic**: Hard to understand even for the author
* **Spaghetti code**: Everything tangled together, no structure
* **Hardcoded values**: No flexibility or scalability
* **Duplicated code**: Same logic repeated instead of using functions
* **Violates design principles**: Breaks SRP, has tightly coupled classes
* **Difficult to test**: No modularity, too many dependencies
* **Poor naming**: Variable/method names don’t represent purpose

### Task 2: What do you understand by Data Binding?

* **Definition**: Connecting the data model (backend/data) to the UI layer so they stay in sync
* When data changes → UI updates automatically (in one-way or two-way binding)
* **One-way binding**: Data flows from model to UI  
  (e.g., displaying data from a Java object in a form field)
* **Two-way binding**: Data updates in both directions (UI → data and vice versa)  
  (common in Android using LiveData, ViewModel, or XML binding)
* Used in:
  + Android (Java/Kotlin)
  + JavaFX
  + Web frameworks like Angular/React/Vue
* **Advantages**:
  + Reduces boilerplate code (e.g., no need to manually update UI)
  + Helps separate business logic from UI logic

### Task 3: What do you know about Continuous Development?

* Continuous Development is a **practice in DevOps and Agile** where development, testing, and deployment are done continuously in small, frequent cycles.
* Includes 3 major components:
  + **Continuous Integration (CI)**: Developers regularly push code to a shared repo; automated tests run on each commit
  + **Continuous Delivery (CD)**: Code is always ready to be deployed; goes through automated build and test pipelines
  + **Continuous Deployment**: Final step where code is deployed automatically to production (no manual approval)
* **Benefits**:
  + Faster feedback and shorter release cycles
  + Bugs are caught early
  + Encourages automation and code quality
  + Enables real-time collaboration and rollback if needed

### Task 4: What are the conditions for Polymorphism?

#### What is Polymorphism?

* **Polymorphism = “many forms”**
* A single method name can behave differently depending on the object

#### Types:

1. **Compile-Time Polymorphism** (Static)
   * Achieved by **method overloading**
   * Same method name with different parameter types or count
   * Example:  
     void show(int x) and void show(String x)
2. **Runtime Polymorphism** (Dynamic)
   * Achieved through **method overriding**
   * Requires inheritance or interface
   * Method in child class overrides method in parent class
   * Example:

class Animal {

void speak() { System.out.println("Animal speaks"); }

}

class Dog extends Animal {

void speak() { System.out.println("Dog barks"); }

}

Animal a = new Dog(); // Runtime decides which speak() to call

#### Conditions for Polymorphism:

* For **compile-time**:
  + Same method name with different parameter signature
* For **runtime**:
  + Inheritance (class extends or implements)
  + Method override with same name and signature
  + Method called through parent class reference (upcasting)

### DRY – Don’t Repeat Yourself

* Every piece of logic should have a **single, unambiguous place** in the codebase
* Repeating code leads to inconsistencies and harder maintenance
* Solution: extract repeated code into methods or classes

📌 Example:

Instead of writing the same calculation in 5 places, create a method and reuse it.

### KISS – Keep It Simple, Stupid

* Design and write code in the **simplest way** that solves the problem
* Avoid unnecessary complexity, clever hacks, or over-engineering
* Simple code is easier to test, debug, and maintain

📌 Example:

Avoid using complex nested logic when a simple if-else does the job.

### YAGNI – You Aren’t Gonna Need It

* Don’t implement features until they are actually required
* Avoid writing code “just in case we might need it later”
* Saves time and reduces bloat in the codebase

📌 Example:

Don’t write code to handle 5 user roles when the app only supports 2 today.

### Task 5: What is TDD and BDD approach? Why is it used? Where is it used?

#### TDD – Test-Driven Development

##### What is it?

* TDD is a software development approach where **you write tests before writing the actual code**.
* It follows a **Red → Green → Refactor** loop:
  1. Write a test that fails (Red)
  2. Write minimum code to pass the test (Green)
  3. Refactor the code while keeping the test green

##### Why is it used?

* Ensures code correctness from the beginning
* Promotes clean, modular design
* Prevents over-coding or unnecessary features
* Increases confidence in future code changes

##### Where is it used?

* Mostly used in **unit testing**, particularly in backend logic
* Used heavily in **Agile development** and CI/CD pipelines
* Example: Java developers using **JUnit** to test service or DAO layer logic

#### BDD – Behavior-Driven Development

##### What is it?

* BDD focuses on **how the system should behave from the user’s point of view**
* Uses **natural language syntax (Gherkin)** to define scenarios and maps them to test code

Example:

Scenario: Successful login

Given the user is on the login page

When they enter valid credentials

Then they should be redirected to the dashboard

##### Why is it used?

* Makes testing more **collaborative** and **readable** for developers, testers, and business teams
* Ensures development is aligned with business goals and expected behaviors
* Reduces ambiguity in requirements

##### Where is it used?

* Often used in **acceptance testing** and **integration testing**
* Common in full-stack and UI testing scenarios
* Example: Using **Cucumber** (with Java + Selenium) to test login flow, registration, etc.

#### TDD vs BDD Summary

| **Feature** | **TDD** | **BDD** |
| --- | --- | --- |
| Focus | Code correctness | System behavior (user-focused) |
| Written by | Developers | Developers + Testers + BAs |
| Language | Programming Language (e.g., Java) | Natural Language (Gherkin) + code mapping |
| Test Type | Unit Tests | Integration / Acceptance Tests |
| Tool Example | JUnit | Cucumber |

### Task 6: What is Manual and Automated Testing? List down the tools used.

#### Manual Testing

##### Definition:

Testing performed **manually** by testers without using automation scripts.

##### Purpose:

Used for exploratory, usability, and ad-hoc testing where human observation is critical.

##### Examples of Manual Testing Tools:

* **TestLink** – Test case management
* **JIRA** – Bug tracking and test workflow
* **Bugzilla** – Issue tracking
* **Xray** – Manual test management in JIRA
* **PractiTest** – End-to-end test management

#### Automated Testing

##### Definition:

Testing performed using **automation scripts/tools** to execute test cases and compare actual vs expected output.

##### Purpose:

Used for repetitive, regression, and large-scale testing — improves speed and accuracy.

##### Examples of Automated Testing Tools:

* **JUnit / TestNG** – Unit testing in Java
* **Selenium** – Automates browser-based UI testing
* **Cucumber** – BDD (behavior-driven testing using Gherkin syntax)
* **Appium** – Mobile app automation
* **Postman** – API testing (can be automated with scripts)
* **JMeter** – Performance/load testing