**Task 1**

**GOOD Code** refers to code that is clean, readable, maintainable, efficient, and follows best practices. It uses meaningful variable names, proper indentation, comments where necessary, and is modular and reusable.

**BAD Code** is often messy, hard to read or maintain, lacks structure, contains redundant or repeated logic, and does not follow naming or formatting conventions. It may work, but is difficult for others (or even the original author) to understand or modify later.

**Task 2**

**Databinding** is a programming technique that binds UI elements to data sources so that changes in the data are automatically reflected in the UI and vice versa. It is commonly used in frameworks like Angular, React, and JavaFX to synchronize the state of the application with the user interface efficiently.

**Task 3**

**Continuous Development** is a software engineering practice that involves continuously updating and delivering code in small, incremental changes. It includes concepts like Continuous Integration (CI), Continuous Delivery (CD), and Continuous Deployment to ensure that software can be released reliably at any time with automated testing and integration.

**Task 4**

**Conditions for Polymorphism:**

1. There must be **inheritance** — a subclass must inherit from a superclass.
2. The subclass must **override** or implement the method from the superclass.
3. The method signature should be the same if overriding (for runtime polymorphism).
4. In case of **method overloading** (compile-time polymorphism), the method names are same but parameter lists are different.
5. The object should be accessed via the **reference of the parent class** (for dynamic method dispatch at runtime).

**Task 5**

**TDD (Test-Driven Development):**

* **What is it:** TDD is a software development approach where **tests are written before the actual code**. The development cycle follows the pattern: **Write Test → Write Code → Refactor**.
* **Why is it used:** It ensures that code is tested thoroughly from the beginning. It helps in preventing bugs early, improves design, and ensures high test coverage.
* **Where is it used:** Commonly used in **unit testing**, agile environments, backend services, and systems requiring high reliability (e.g., finance, health-tech).

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

* **What is it:** BDD is an extension of TDD that focuses on the **behavior of the application from the user's perspective**. It involves writing test cases in natural language using tools like **Cucumber**, **SpecFlow**, or **Behave**.
* **Why is it used:** It bridges the gap between **developers, testers, and non-technical stakeholders**. It ensures everyone has a clear understanding of how the system should behave.
* **Where is it used:** Widely used in **acceptance testing**, **UI/UX validation**, and projects with strong stakeholder collaboration needs.

**Task 6**

**Manual Testing Tools:**

1. **TestLink** – Test case management and execution tool.
2. **PractiTest** – End-to-end test management platform.
3. **Zephyr** – Jira-integrated test management tool.
4. **qTest** – Test management and collaboration platform.
5. **Xray** – Test management for Jira.
6. **TestRail** – Web-based test case and test plan management.

**Automated Testing Tools:**

1. **Selenium** – Open-source tool for automating web browsers.
2. **JUnit/TestNG** – Java frameworks for unit testing.
3. **Appium** – Automation for mobile applications (Android/iOS).
4. **Cypress** – JavaScript end-to-end testing tool for modern web apps.
5. **Postman** – API testing and automation.
6. **Robot Framework** – Keyword-driven automation tool.
7. **Cucumber** – BDD testing tool using Gherkin syntax.
8. **Katalon Studio** – Integrated automation testing platform.
9. **JMeter** – Load and performance testing tool for web applications.
10. **Playwright** – Modern end-to-end testing for web apps.