# **Day 28 - 29 August 2025**

**Document Name:**Day 28 - hmuvvala@ - Hari Gopal Muvvala

### Task 3

**Question:**

class A {

public static void main(String[] args) {

int a = 5;

int b = 10;

int c = 15;

System.out.println((a > b) && (b < c));

}

}

**Options:**

1. Compilation error
2. True
3. False
4. Runtime error

**Correct Option:** **3. False**

**Notes/Explanation:**

* a > b → 5 > 10 → false
* b < c → 10 < 15 → true
* false && true → false
* No compilation/runtime errors → final answer is **false**.

### Task 4

**Question:** Finding inheritance during requirement analysis is it important in OOAD … why so?

**Options:**

1. It removes the need for encapsulation in the system design
2. It helps identify objects with the shared behavior to promote code reuse and logical hierarchy
3. It forces a flat class design improving performance by reducing polymorphic calls
4. Ensures all classes are instantiated using interfaces

**Correct Option:** **2. It helps identify objects with the shared behavior to promote code reuse and logical hierarchy**

**Notes/Explanation:**

* Inheritance captures **“is-a” relationships**.
* Promotes **code reuse** and a **hierarchical structure**.
* Identified early → reduces duplication later.

### Task 5

**Question:** Which characteristic best defines polymorphism in OOP?

**Options:**

1. Ensures each class has its own copy of data members
2. It restricts method access to specific roles within a system
3. It allows a single function or operator to behave differently based on its parameters or calling object
4. It serializes different objects into a common file format for persistence

**Correct Option:** **3. It allows a single function or operator to behave differently based on its parameters or calling object**

**Notes/Explanation:**

* Polymorphism = **“many forms”**.
* Example: method overriding (run() in Car vs Bike).
* Example: method overloading (same method name, diff params).

### Task 6

**Question:** What best explains **data hiding**?

**Options:**

1. Removing data from memory when no longer in use
2. Using access specifiers to restrict direct access to class members, enabling controlled interaction through methods
3. Storing object data in secure databases during runtime
4. Deleting unused attributes from objects after creation

**Correct Option:** **2. Using access specifiers …**

**Notes/Explanation:**

* Achieved with **private/protected** access.
* Example: private balance; public getBalance().
* Prevents misuse and enforces controlled access.

### Task 7

**Question:** In OOAD, what is the **primary value of Requirements Analysis**?

**Options:**

1. It helps define class inheritance structure before testing
2. It identifies system behavior and user needs to model objects and interactions meaningfully
3. It configures application deployment scripts for testing
4. It automatically generates interface documentation from class files

**Correct Option:** **2. It identifies system behavior and user needs …**

**Notes/Explanation:**

* Requirement analysis = **understanding what system should do**.
* Provides foundation for modeling objects and interactions.

### Task 8

**Question:** Code snippet with private static ClassName instance; and getInstance() returning a single instance.

**Options:**

1. Factory Method
2. Singleton
3. Prototype
4. Builder

**Correct Option:** **2. Singleton**

**Notes/Explanation:**

* **Singleton** ensures only **one instance** of class.
* Example: Runtime.getRuntime() in Java.

### Task 9

**Question:** Why is Interface preferred in Java for polymorphism?

**Options:**

1. Enforce tight coupling
2. Interfaces offer default constructors and static fields
3. Interfaces allow multiple inheritance of behavior, promoting decoupling and flexibility
4. Interfaces provide direct access to private logic

**Correct Option:** **3. Interfaces allow multiple inheritance …**

**Notes/Explanation:**

* Java doesn’t support multiple inheritance of classes.
* Interfaces solve this → a class can implement many interfaces.

### Task 10

**Question:** Role of **Inception Phase** in RUP?

**Options:**

1. Final phase where deployment & training occur
2. Defines runtime environment
3. Helps establish business case, scope, and feasibility
4. Focuses on UI + DB integration

**Correct Option:** **3. Helps establish business case, scope, feasibility**

**Notes/Explanation:**

* Inception = **“why and what”** of project.
* Defines project vision and feasibility.

### Task 11

**Question:** What aspect makes **UML diagrams** crucial?

**Options:**

1. Detailed flowcharts for logic
2. Runtime logs for monitoring
3. Visually capture structure/behavior of system (classes, objects, interactions)
4. Replace testing frameworks

**Correct Option:** **3. Visually capture structure/behavior …**

**Notes/Explanation:**

* UML = communication tool.
* Models both **structure** (class diagrams) and **behavior** (sequence diagrams).

### Task 12

**Question:** Why is **refactoring** continuous?

**Options:**

1. Performed only at release end
2. Replaces debugging
3. Ensures design evolves with requirements, reduces debt, improves health
4. Removes dependencies to minimize source control conflicts

**Correct Option:** **3. Ensures design evolves …**

**Notes/Explanation:**

* Refactoring = **improving design without changing functionality**.
* Continuous → avoids technical debt build-up.

### Task 13

**Question:** Importance of **Elaboration Phase** in RUP?

**Options:**

1. Prepares production pipelines
2. Major architectural decisions validated via prototypes & risk mitigation
3. Finalize UI designs
4. Refactor legacy code

**Correct Option:** **2. Major architectural decisions validated …**

**Notes/Explanation:**

* Elaboration = prove architecture works.
* Remove risks before Construction begins.

### Task 14

**Question:** How are **Active Objects** represented?

**Options:**

1. Static utility classes
2. Objects that encapsulate their own thread of control and asynchronously handle requests
3. Serialized containers
4. JavaBeans for UI binding

**Correct Option:** **2. Objects encapsulating thread of control + async requests**

**Notes/Explanation:**

* Active objects manage concurrency themselves.
* Example: ExecutorService in Java.

### Task 15

**Question:** What makes **Composite Pattern** useful?

**Options:**

1. Replaces collections
2. Allows treating individuals and groups uniformly via a common interface
3. Automatically serializes tree objects
4. Optimizes memory

**Correct Option:** **2. Treat individuals + groups uniformly**

**Notes/Explanation:**

* Composite = **tree structures** (e.g., directory–file).
* Client sees both leaf & composite via same interface.

### Task 16

**Question:** Code with ABC interface, OperationAdd, OperationSubtract, Context → executes via abc.doOperation().

**Options:**

1. Observer
2. Strategy
3. Builder
4. Prototype

**Correct Option:** **2. Strategy**

**Notes/Explanation:**

* Strategy = define family of algorithms, encapsulate, make interchangeable.
* Context delegates operation to chosen strategy object.