Day 28 - 20th Aug 2025

Name : Aravind Kasanagottu

Id: mvsnarav

=

**Task 2**

Task 2:

Give output:

Class A {

psvm() {

Int a = 5;

Int b = 10;

Int c = 15:

sout((a > b ) && (b < c))

}

}

a) Compilation error  
b) True  
c) False  
d) Runtime error

**Answer: c) False**

**Task 3**

Finding Inheritance during requirement analysis is it important in OOAD .. why so?

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

**Answer: b) It helps identify objects with the shared behavior to promote code reuse and logical hierarchy**

**Task 4**

Which characteristics best defines polymorphism in OOP?

a) Ensures each class has its own copy of data members  
b) It restricts method access to specific roles within a system  
c) It allows a single function or operator to behave differently based on its parameters or calling object  
d) It serialized different objects into a common file format for persistence

**Answer: c) It allows a single function or operator to behave differently based on its parameters or calling object**

**Task 5**

Which design pattern is being used?

a) Factory Method  
b) Singleton  
c) Strategy  
d) Builder

**Answer: c) Strategy**

**Task 6**

Which best explains data hiding?

a) Removing data from memory when no longer in use  
b) Using access specifiers to restrict direct access to class members, enabling controlled interaction through methods  
c) Storing object data in secure databases during runtime  
d) Deleting unused attributes from objects after object creation

**Answer: b) Using access specifiers to restrict direct access to class members, enabling controlled interaction through methods**

**Task 7**

Primary value of Requirements Analysis?

a) Helps define class inheritance structure before testing  
b) Identifies system behavior and user needs to model objects and interactions meaningfully  
c) Configures application deployment scripts for testing  
d) Automatically generates interface documentation from class files

**Answer: b) Identifies system behavior and user needs to model objects and interactions meaningfully**

**Task 8**

Which design pattern is implemented?

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

**Answer: 2. Singleton**

**Task 9**

Why Interfaces are preferred in Java for polymorphism?

1. Interfaces enforce tight coupling between child and parent classes
2. Interfaces offer default constructors and static fields
3. Interfaces allow a class to inherit from multiple sources of behavior, promoting decoupling and flexibility
4. Interfaces provide direct access to private implementation logic

**Answer: 3. Interfaces allow a class to inherit from multiple sources of behavior, promoting decoupling and flexibility**

**Task 10**

Role of Inception Phase?

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

**Answer: 3. Helps establish business case, scope and feasibility**

**Task 11**

Aspect of UML Diagrams in OOAD?

1. Provide detailed flowcharts for programming logic
2. Represent runtime logs for monitoring
3. Visually capture the structure and behavior of systems through elements like classes, objects, and interactions
4. Replace testing frameworks

**Answer: 3. Visually capture the structure and behavior of systems through elements like classes, objects, and interactions**

**Task 12**

Why refactoring is continuous?

1. Performed only at release cycle
2. Replaces debugging with patching
3. Continuous refactoring ensures design evolves with changing requirements, reduces technical debt and improves code health
4. Removes dependencies to minimize conflicts

**Answer: 3. Continuous refactoring ensures design evolves with changing requirements, reduces technical debt and improves code health**

**Task 13**

Importance of Elaboration Phase?

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

**Answer: 2. Major architectural decisions validated via prototypes and risk mitigation**

**Task 14**

How are Active Objects represented?

1. As static utility classes
2. As objects that encapsulate their own thread of control and asynchronously handle requests
3. As serialized containers
4. As Java Beans

**Answer: 2. As objects that encapsulate their own thread of control and asynchronously handle requests**

**Task 15**

Composite Pattern usefulness?

1. Replaces collections
2. Treating individual objects and compositions uniformly via a common interface
3. Automatically serializes tree objects
4. Optimizes memory

**Answer: 2. Treating individual objects and compositions uniformly via a common interface**

**Task 17**

Business Logic Layer role?

1. Manages physical data storage
2. Processes commands, performs validations, implements core logic
3. Defines system behavior under network traffic
4. Renders UI

**Answer: 2. Processes commands, performs validations, implements core logic**

**Task 18**

Role of Packages?

1. Only to store deprecated classes
2. Group related elements, modularize systems into subsystems with defined interfaces
3. Represent reusable libraries only
4. Define runtime performance model

**Answer: 2. Group related elements, modularize systems into subsystems with defined interfaces**

**Task 19**

Code smell?

a) Long Method  
b) Primitive Obsession  
c) Large Class  
d) Feature Envy

**Answer: c) Large Class**

**Task 20**

SCache implementation?

1. Command pattern
2. Double checked locking Singleton, ensures lazy and thread-safe initialization
3. Factory with static holder
4. Prototype with unnecessary locking

**Answer: 2. Double checked locking Singleton, ensures lazy and thread-safe initialization**

**Task 21**

Code smell?

a) Long Method  
b) Primitive Obsession  
c) Large Class  
d) Feature Envy

**Answer: d) Feature Envy**

**Task 22**

Principle violated?

1. LSP violated
2. Dependency Inversion violated
3. Open Closed violated
4. Interface Segregation Principle violated

**Answer: 4. Interface Segregation Principle violated**

**Task 23**

Design issue in Notification hierarchy?

1. Violates Interface Segregation
2. Violates Liskov Substitution Principle → use interfaces and split behaviors
3. No issue
4. Follows OCP

**Answer: 2. Violates Liskov Substitution Principle → use interfaces and split behaviors**

**Task 24**

Key benefit of Facade Pattern?

1. Eliminates middle layers
2. Direct low-level access
3. Inject multiple implementations
4. Simplifies access via unified interface

**Answer: 4. Simplifies access via unified interface**

**Task 25**

Proxy Pattern role?

1. Executes logic inside components
2. Logs calls only
3. Placeholder to control access (lazy loading, access control, caching)
4. Permanently replaces object

**Answer: 3. Placeholder to control access (lazy loading, access control, caching)**

**Task 26**

Open/Closed Principle?

1. Open for modification, closed for extension
2. Open for extension, closed for modification
3. Accept runtime changes without altering behavior
4. Code must be static

**Answer: 2. Open for extension, closed for modification**

**Task 27**

Builder vs Prototype?

1. Builder shallow copying, Prototype step by step
2. Builder separates construction from representation, Prototype clones objects
3. Builder clones quickly, Prototype builds with helpers
4. Same purpose, Builder compile time, Prototype runtime

**Answer: 2. Builder separates construction from representation, Prototype clones objects**

**Task 28**

Improving legacy system?

1. Merge related classes
2. Rewrite all modules
3. Refactor to follow SRP and identify code smells
4. Move business logic to frontend

**Answer: 3. Refactor to follow SRP and identify code smells**

**Task 29**

Anti-pattern in UserManager?

1. Violates OCP, use polymorphism instead of hard-coded conditions
2. No refactoring needed
3. Method already uses polymorphism
4. Move logic to DB

**Answer: 1. Violates OCP, use polymorphism instead of hard-coded conditions**

**Task 30**

Microservice notifications?

1. Centralized DB
2. Direct RPC calls
3. Asynchronous messaging with Publish-Subscribe
4. Add retry logic

**Answer: 3. Asynchronous messaging with Publish-Subscribe**

**Task 31**

Scalable architecture?

1. Decorator
2. 3-tier Architecture (UI, Business, Data)
3. Singleton per layer
4. Proxy classes

**Answer: 2. 3-tier Architecture (UI, Business, Data)**

**Task 32**

Characteristic of valuable unit test?

1. Test one method but involve multiple objects/external systems
2. Execute complex test scenarios
3. Independent, unrelated to software behavior
4. Repeatable, focused, single responsibility, clear outcomes

**Answer: 4. Repeatable, focused, single responsibility, clear outcomes**

**Task 33**

High test coverage but poor quality?

1. Add more assertions
2. Refactor tests to cover edge cases, boundary conditions, business logic
3. Migrate to performance tests
4. Replace unit tests with mocks

**Answer: 2. Refactor tests to cover edge cases, boundary conditions, business logic**

**Task 34**

Fetching data from multiple sources?

1. Singleton
2. Strategy Pattern
3. Prototype
4. Decorator

**Answer: 2. Strategy Pattern**

**Task 35**

Decoupling producers/consumers?

1. Direct REST calls
2. Publish-Subscribe Pattern
3. Shared DB
4. Adapter Pattern

**Answer: 2. Publish-Subscribe Pattern**