Important Topics / Key Points for Discussions

Architectural Pattern

- Building architecture
- Enterprise architecture
- Data architecture
- Business architecture
- Animation architecture
- Technical architecture

Design Pattern

- Interior design
- Visual design
- Technology design
- User Interface design
- Web design
- Experience design
- Product design

Anti-Patterns

- Anti-patterns are certain patterns in software development that are considered bad programming practices
- Anti-patterns are counter part of Design Pattern
- Define an industry vocabulary for the common defective processes and implementations within organizations.
- A higher-level vocabulary simplifies communication between software practitioners and enables concise description of higher-level concepts.
- To improve the developing of applications, the designing of software systems, and the effective management of software projects.

Command Pattern

•**When designing a banking transaction system using UML (Unified Modeling Language), several design patterns can be applied depending on the requirements of the system.**

Purpose: Encapsulates a request as an object, thereby allowing for parameterization of clients with queues, requests, and operations.

•Use Case: For banking transactions like deposit, withdrawal, and transfer.

•Implementation:

o**Command Interface**: Defines the common interface for all concrete commands (e.g., execute()).

o**Concrete Commands**: Implement the Command interface (e.g., Deposit Command, Withdraw Command, Transfer Command).

o**Invoker**: Triggers the execution of commands (e.g., Transaction Invoker).

o**Receiver**: The object that performs the actual action (e.g., BankAccount).

Factory Method Pattern

•Purpose: Defines an interface for creating an object, but lets subclasses alter the type of objects that will be created.

•Use Case: To create different types of transactions such as deposits, withdrawals, and transfers.

•Implementation:

- **Transaction Factory Interface**: Defines the interface for creating transactions (e.g., create Transaction()).
- **Concrete Factories**: Implement the creation of specific transactions (e.g., Deposit Transaction Factory, Withdraw Transaction Factory, Transfer Transaction Factory).
- **Product Interface**: Common interface for different transactions (e.g., Transaction).
- **Concrete Products**: Implement specific transactions (e.g., Deposit Transaction, Withdraw Transaction).

Strategy Pattern

• Purpose: Defines a family of algorithms, encapsulates each one, and makes them interchangeable.

• Use Case: To determine the fee calculation strategy for different types of transactions.

• Implementation:

✓Strategy Interface: Defines a common interface for fee calculation strategies (e.g., calculateFee()).

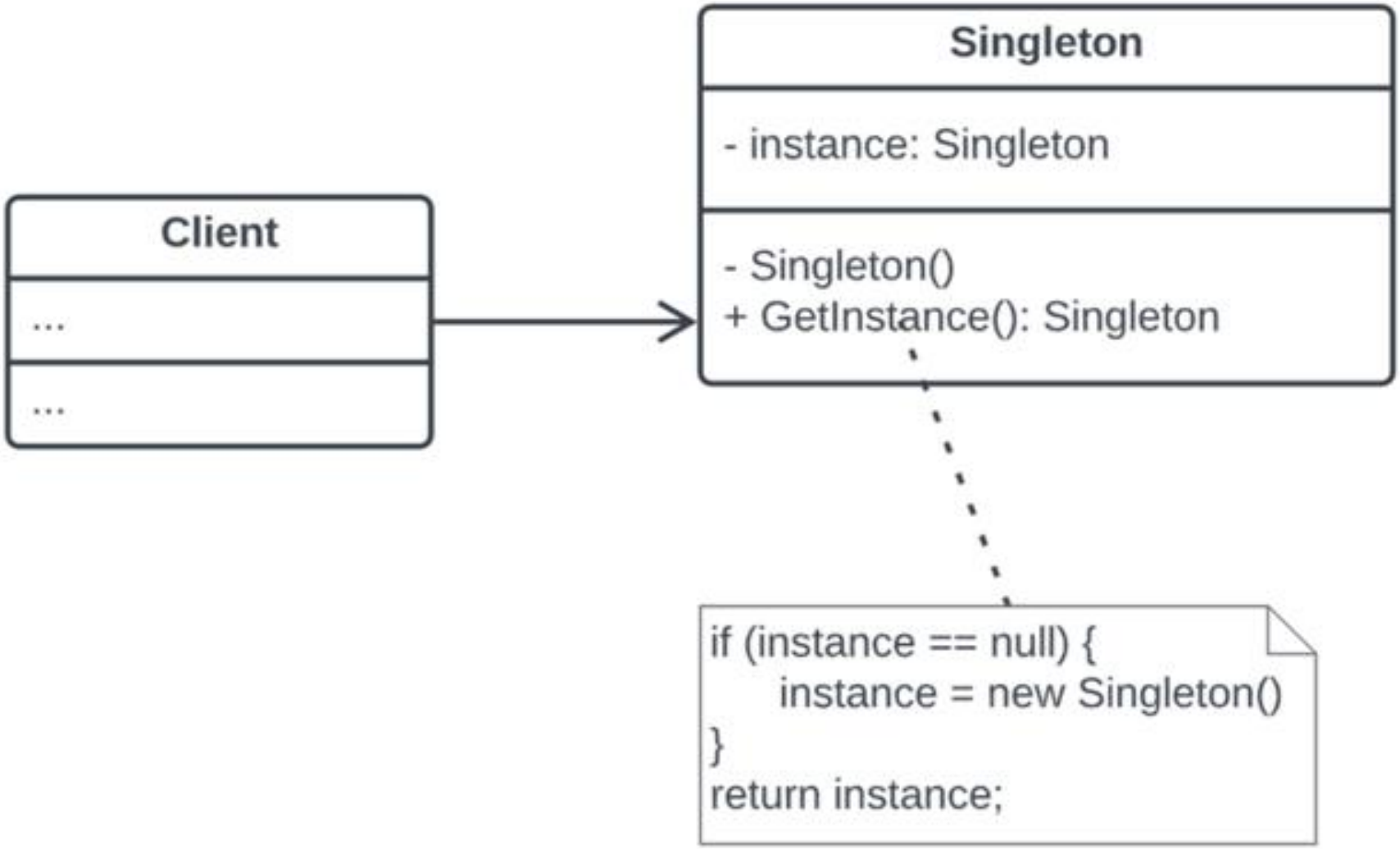
✓Concrete Strategies: Implement specific fee calculation algorithms (e.g., Fixed Fee Strategy, Percentage Fee Strategy, NoFeeStrategy).

✓Context: The class that uses a strategy to perform an operation (e.g., Transaction uses Fee Strategy).

Design Patterns

- Singleton Pattern: Used for `Logger` to ensure a single instance of the logging service.
- Factory Pattern: Used for creating instances of `Account` (SavingsAccount or CheckingAccount).
- Observer Pattern: Used between `Account` and NotificationService to notify customers of transactions.

Singleton Pattern



```
import java.io.*;
class Singleton {
    // static class
    private static Singleton instance;
    private Singleton()
    {
        System.out.println("Singleton is Instantiated.");
    }
    public static Singleton getInstance()
    {
        if (instance == null)
            instance = new Singleton();
        return instance;
    }
    public static void doSomething()
    {
        System.out.println("Somethong is Done.");
    }
}
class GFG {
    public static void main(String[] args)
    {
        Singleton.getInstance().doSomething();
    }
}
```

```
+-------------------+        +-------------------+
|     User          |        |   BakingProcess   |
|-------------------|        |-------------------|
| - username: String |        | - processDetails: |
| - password: String |        |       String      |
|-------------------|        |-------------------|
| + login(): void   | <------| + start(): void   |
|-------------------|        |-------------------|
        |                          |
        |                          |
        v                          |
+---------------------+            |
|   <<Singleton>>     |            |
|   LoginService      |------------+
|---------------------|
| - instance:         |
|     LoginService    |
| - loggedIn: Boolean |
|---------------------|
| + getInstance():    |
|     LoginService    |
| + login(User): void |
| + isLoggedIn():     |
|     Boolean         |
+-------------------+
```

When to use waterfall model

The Waterfall model is a traditional software development methodology that follows a linear and sequential approach. It is best suited for projects where requirements are simple, well-understood, Requirements are Clear and Stable.

- Simple or Small Projects
- Well-Documented Process
- Inexperienced Teams
- There is No Immediate Feedback

Applications of Waterfall Model

- Used to develop enterprise applications like
- Customer Relationship Management (CRM) systems
- Human Resource Management Systems (HRMS)
- Supply Chain Management Systems
- Inventory Management Systems
- Point of Sales (POS) systems for retail chains, etc.

Advantages of Waterfall Model

- Provides a way for large or changing teams to work together toward a common goal defined in the requirements phase
- Ensures a disciplined and structured organization
- Provides a simple method to understand, follow, and arrange tasks
- Facilitates management control and departmentalization based on deadlines
- Provides easy access to early system design and specification changes
- Defines milestones and deadlines clearly

Limitations of Waterfall Model

- Waterfall model can be effective for small projects with well-defined requirements.
- it is not an ideal one for large projects due to its inflexibility,
- lack of feedback, and dependence on upfront planning and design.
- Moreover, it is difficult to identify the challenges and risks in the earlier stages.

Disadvantages / Challenges

- Design flaws, when discovered, often mean starting over from scratch
- It doesn’t **incorporate mid-process feedback** from users or clients and makes changes based on results
- Delaying the testing until the end of development is common
- There’s no consideration for error correction
- The model doesn’t **accommodate changes, scope adjustments, and updates well**
- Work on different phases doesn’t overlap, which reduces the efficiency
- Projects don’t produce a working product until later stages
- Not an ideal model to use for **complex and high-risk projects**

Disadvantages / Challenges Conti…

- Customer involvement is limited
- High Risk and Uncertainty
- Limited to Specific Types of Projects

Agile Process Model

- The Agile process model is a popular approach in software development that emphasizes flexibility, collaboration, and customer satisfaction.
- It is an iterative and incremental model where development is carried out in small, manageable units called "sprints" or "iterations.

Key Principles of Agile:

- **Customer Collaboration**: Continuous interaction with customers to gather feedback and refine the product based on their needs.
- **Flexibility and Adaptability**: Agile embraces changes, even late in the development process. It allows teams to adapt to evolving requirements.
- **Incremental Delivery**: Deliver working software frequently, with a preference for shorter timescales.
- **Continuous Improvement**: Teams reflect on how to become more effective after each sprint and adjust their behavior accordingly.

Key Principles of Agile:

- **Cross-functional Teams**: Teams consist of members with various skills needed to complete the work, promoting collaboration and reducing bottlenecks.
- **Simplicity**: Focus on the simplest and most essential requirements to deliver immediate value.

Benefits of Agile:

- Increased customer satisfaction through continuous delivery.
- Higher product quality due to frequent testing and feedback.
- Enhanced team collaboration and communication.
- Flexibility to adapt to changes in requirements.
- Faster time to market with incremental releases.

Challenges of Agile:

- Requires significant cultural change within the organization.
- Can be difficult to scale in large organizations.
- Requires close collaboration, which can be challenging with distributed teams.
- May lead to scope creep if changes are not well-managed.

Agile Process Flow:

- **Requirements Gathering**: Collaborate with stakeholders to define high-level requirements and create a product backlog.
- **Sprint Planning**: Select a subset of items from the product backlog to work on during the sprint.
- **Development**: Design, develop, and test features in an iterative manner within the sprint.
- **Daily Standups**: Short meetings to track progress, identify roadblocks, and adjust plans.
- **Sprint Review**: At the end of the sprint, present the working increment of the product to stakeholders for feedback.
- **Sprint Retrospective**: Reflect on the sprint to discuss what went well, what didn't, and how to improve in the next sprint.
- **Release Planning**: After several sprints, a stable version of the product is released to customers.

	Waterfall	Agile
Implementation	Linear	Iterative
Detailed plan timeframe	Whole project	Typically 2-8 weeks
Stakeholder engagement	Mostly upfront	Throughout the entire project
Team structure	Traditional hierarchy	Often more self-organized
Main benefit	Consistent, reliable, and controlled outcomes	Adaptability to rapidly changing markets

Software Principles

- Software engineering principles help teams build highly reliable, efficient, and quality software applications that meet user requirements.
- By following specific principles, software engineers can create a product that is easy to understand, maintain, modify, and solve the user's pain points
- Improved Code Quality:
- Enhanced Collaboration:.
- Increased Efficiency
- Efficient Development Process

Software Principles Conti…

- **Scalability and Flexibility**: Applying design patterns and architectural principles helps in creating scalable and flexible systems. This makes it easier to add new features or modify existing ones without disrupting the entire system
- .**Risk Management**: Principles like testing, code reviews, and automated deployment reduce the likelihood of bugs and system failures. This proactive approach to risk management ensures higher reliability and stability of the software.
- **Clear Documentation**: Following software engineering practices often involves creating comprehensive documentation. This documentation serves as a reference for current and future team members, ensuring continuity even when there is a change in personnel.

Software Principles Conti…

- **Better Project Management**: Software engineering principles include methodologies for planning, tracking, and reviewing progress. This helps in better project management, ensuring that the development stays on track and meets the defined goals.
- **Cost Efficiency**: By reducing the amount of rework, minimizing bugs, and improving productivity, software engineering principles ultimately lead to cost savings. Well-structured code and clear documentation also reduce maintenance costs in the long run.
- **Customer Satisfaction**: Delivering high-quality, reliable software on time leads to higher customer satisfaction. By following best practices, the team can ensure that the final product meets or exceeds customer expectations.
- **Sustainability**: Adopting these principles promotes sustainability by making the codebase easier to maintain and evolve over time. This longevity is crucial for software that needs to adapt to changing business requirements.
- **Increased Reliability and Security**

Software Crisis for Team

- Unclear Requirements:
- Poor Project Management:
- Technical Debt:.
- Inadequate Testing:
- Rapid Technology Changes:
- Team Skill Gaps:
- Integration Issues:
- Communication Breakdown:
- Security Vulnerabilities:
- Resource Constraints:
- User Expectations:
- Maintaining Compatibility:
- .Changing Market Conditions:
- Scalability Issues:

	Booch Method	Rumbaugh Method	Jacobson Method
Approach	Object Oriented Approach	Object Oriented Approach	User Centric Approach
Phases Covered	Analysis, design and implementation phases	Analysis, design and implementation phase	All phases of life phase cycle.
Strength	Strong method for producing detailed object oriented design models	Strong method for producing object model static structure of the system	Strong method for producing user driven requirements and object oriented analysis model
	Analysis	Design	Requirement

	Booch Method	Rumbaugh Method	Jacobson Method
Weakness	Focus entirely on design and not on analysis	Cannot fully express the requirements	Do not treat OOP to the same level as other Methods.
Unidirectional Relationships	Uses.	Directed Relationships	
Diagrams used	Class diagram, state transition diagram, object diagram, timing diagram, Module diagram, process diagram	Data flow diagrams, state transmission diagram, class/object diagram	Use Case Diagram

Key artifacts

- Requirements Document (Functional and Non-Functional)
- User Details and Use Cases
- Technical Architecture Document
- Prototypes
- Test Plans and Test Cases

MCQ to test Understanding / Knowledge Level

1. Which one diagram Model static data structures.
(A). Object diagrams
(B). Class diagrams
(C). Activity diagrams
(D). Interaction diagrams
(E). All of the above

2. Use case descriptions consist of interaction_____?
a) Use case
b) product
c) Actor
d) Product & Actor
3. Diagrams which are used to distribute files, libraries, and tables across topology of hardware are called
A. deployment diagrams
B. use case diagrams
C. sequence diagrams
D. collaboration diagrams

4. How many views of the software can be represented through the Unified Modeling Language (UML)
a. Four
b. Five
c. Nine
d. None of the above

5. Which of the following views represents the interaction of the user with the software but tells nothing about the internal working of the software?
a. Use case diagram
b. Activity diagram
c. Class diagram
d. All of the above

6.which of these compartments divided in class?
A) Name
B) Attribute
C) Operation
D) All of the mentioned

7.Composition is another form of …
a) inheritance
b) encapsulation
c) aggregation
d) none of these

8.To hide the internal implementation of an object, we use …
a) inheritance
b**)** encapsulation
c) polymorphism
d) none of these

9.The vertical dimension of a sequence diagram shows
a) abstract
b) line
c) time
d) Messages

10.CRC approach and noun phrase approach are used to identify …
a) classes
b) colaborators
c) use cases
d) object

11. UML diagram that shows the interaction between users and system, is known as
A. Activity diagram
B. E-R diagram
C. Use case diagram
D. Class diagram

12.which diagrams are used to distribute files, libraries, and tables across topology of the hardware
A) **deployment**
B) use case
C) sequence
D) collaboration

13.which diagram that helps to show Dynamic aspects related to a system?
A) sequence
B) interaction
C) deployment
D) use case

14which diagram is used to show interactions between messages are classified as?
A) activity
B) state chart
C) collaboration
D) object lifeline

15.The diagram that helps in understanding and representing user requirements for a software project using UML (Unified Modeling Language) is

(A) ER Diagram

(B) Deployment Diagram

(C) Data Flow Diagram

(D) Use Case Diagram

16.In the spiral model of software development, the primary determinant in selecting activities in each iteration is

(A) Iteration Size

(B) Cost

(C) Adopted Process such as Relational Unified Process or Extreme Programming

(D) Risk

17.What is a design pattern in software development?

A) A fixed set of coding rules

B) A general reusable solution to a commonly occurring problem

C) A specific coding style

D) None of the above

18.How many types of design patterns are there?

A) 3

B) 5

C) 8

D) 10

19.What is the main benefit of using a design pattern?

A) It reduces the total codebase

B) It allows for the separation of responsibilities

C) It ensures that the code is easier to understand and debug

D) All of the above

20. Which of the following is NOT a creational design pattern?

A) Singleton Pattern

B) Factory Pattern

C) Bridge Pattern

D) Prototype Pattern

21. Which of the following is a behavioral design pattern?

A) Observer Pattern

B) Composite Pattern

C) Flyweight Pattern

D) Builder Pattern

22. Which structural pattern should be used when you want to add responsibilities to an object dynamically?

A) Bridge

B) Composite

C) Decorator

D) Adapter

How many design patterns are there total?

GoF of Design Patterns

- Erich Gamma
- Richard Helm
- Ralph Johnson and
- John Vlissides

Design Patterns - Elements of Reusable Object Oriented Software