Deep Skilling:

- Design pattern and principles

i) Singleton Pattern:

They ensures that a class has only one Instance and provide a global access point to it.

Only one object of the class exits in the memory (single Instance)

- Provides a static method to get the instance (Global access)

(thread safety) and (Lazy or Eager Initialization)

use synchronized in method - give access to only one thread at a time.

eg. public static synchronized Singleton getInstance()

Design pattern:

Creational	Structural	Behavioral
• Factory method	Adapter	Observer
• Builder	Decorator	Strategy
• Abstract factory	Facade	command
• Singleton	Proxy	State

Singleton : Ensures a class has one instance

Factory : Creates object without specifying the exact class

Builder : Simplifies complex object creation

Adapter : Makes incompatible interfaces work together

Decorator : Adds behavior to objects dynamically

Observer : Notifies dependents of state changes

Factory design pattern :

They provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created

helps creating object without exposing the instantiation logic to client

Builder Pattern :

allows to construct complex objects step by step

flexible, extensible & easy to maintain

Interface : blueprint for class, defining a contract that implementing class

concrete class : a class that gives full implementation of all its method

abstract class : mix b/w a normal class and an interface . can have • abstract method (no body)
• Concrete method (with body)

Gavina Verma

SME - week - 1 (Laxman)

SOLID principle :

S- single responsibility Principle : class should have only one reason to change, it should have only one duty .

eg : if class student, there should be no irrelevant variable

O - open/closed Principle : open for extension
closed for modification

L- Liskov substitution principle ; obj of superclass should be replaceable with objects of subclass without affecting the correctness of pgm

I- Interface Segregation Principle : no client is forced to depend on method

D- dependency inversion : high level module do not depend on low level

MVC - model view controller

D.S ⁿ

LIST → array list, linked list

SET → hash set, Tree set (cannot allow duplicate element)

MAP → hash map, Tree map

Adapter pattern:

connect two incompatible classes by creating a mid layer (adapter) that translates one interface to another

eg: multiple payment gateways

Method Overloading: (one fn, many version)

Same method name, diff i/p types or no. of i/p

Method Overriding: child class changes behavior of a parent class method.

Decorator Pattern:

add behaviors to object dynamically, without modifying the code: (adds layer)

Eg: Payment (if one notifier object is triggered for any kind of message)

ORM: helps interact with a database using Java objects, without writing SQL.

ORM - object Relational Mapping

Proxy Pattern: It is a class that control access to another class:

adds functionality like lazy loading, access control, caching.

Eg: Image Viewer (load fully only when we scroll

Observer Pattern :

This defines one to many relationship b/w objects so that when one object changes state all its dependents (observer) are notified automatically

Eg : → Youtube (when creator posts, subscriber notified)
→ Stock market

Strategy Pattern :

They allows to define a family of algorithm, put each in seperate class & make them interchangable at runtime.

Eg : Navigation app (it switches strategy drive → walk → bike)

Command Pattern :

This turns a request into standalone object that can be stored, passed around and executed later.

Eg : TV Remote (power, volume)
They don't need to know how TV works thus they encapsulate req as object.

MVC pattern : (model - view - control)

It is a fundamental pattern used in web & desktop application

Eg: ATM machine

Model → Acc balance, transaction logic

View → ATM screen

Ctrl → Button handling

Dependency Injection:

dependency of a class are provided from outside instead of the class creating them itself

DSA:

Array: Fixed size, Random access using index, Stored in contiguous memory.

* ArrayList is a resizable array by Java's Collection Framework -

→ Vector implements dynamic array, thread safe

Hashing:

converting a key into unique index in a table using hash function.

Hash map - Stores key-value pairs (non synchro

Hash table - legacy version of HashMap (synchronized

Hash set - Stores only unique values.