Week 1 - Design Patterns

# 1. Singleton Pattern

✅ Code:

class Singleton {  
 private static Singleton instance;  
  
 private Singleton() {  
 System.out.println("Singleton instance created");  
 }  
  
 public static Singleton getInstance() {  
 if (instance == null)  
 instance = new Singleton();  
 return instance;  
 }  
}  
  
public class SingletonExample {  
 public static void main(String[] args) {  
 Singleton s1 = Singleton.getInstance();  
 Singleton s2 = Singleton.getInstance();  
 System.out.println("Are both instances same? " + (s1 == s2));  
 }  
}

📸 Output: A screenshot of a computer program

AI-generated content may be incorrect.

💡 Explanation: Singleton ensures only one instance is created. s1 and s2 refer to the same object.

# 2. Factory Pattern

✅ Code:

interface Animal {  
 void speak();  
}  
  
class Dog implements Animal {  
 public void speak() {  
 System.out.println("Dog says: Woof!");  
 }  
}  
  
class Cat implements Animal {  
 public void speak() {  
 System.out.println("Cat says: Meow!");  
 }  
}  
  
class AnimalFactory {  
 public static Animal getAnimal(String type) {  
 if ("dog".equalsIgnoreCase(type)) return new Dog();  
 else if ("cat".equalsIgnoreCase(type)) return new Cat();  
 else return null;  
 }  
}  
  
public class FactoryPatternExample {  
 public static void main(String[] args) {  
 Animal animal1 = AnimalFactory.getAnimal("dog");  
 Animal animal2 = AnimalFactory.getAnimal("cat");  
  
 if (animal1 != null) animal1.speak();  
 if (animal2 != null) animal2.speak();  
 }  
}

📸 Output: A screenshot of a computer program

AI-generated content may be incorrect.

💡 Explanation: Factory creates Dog or Cat objects based on input. Main class uses the factory method to get object without knowing exact class.

**📘 Overview: What Are Design Patterns?**

Design patterns are typical solutions to common problems in software design. They are blueprints that help developers create flexible, reusable, and maintainable software systems by following proven best practices.

**🔑 Types of Design Patterns (with examples):**

1. **Creational Patterns**
   * Focus on how objects are created
   * Examples: Singleton, Factory Method, Builder
2. **Structural Patterns**
   * Focus on how classes and objects are composed
   * Examples: Adapter, Decorator, Proxy
3. **Behavioral Patterns**
   * Focus on object interaction and responsibility
   * Examples: Strategy, Observer, Command
4. **Architectural Patterns** (optional/advanced)
   * Examples: MVC (Model-View-Controller), Dependency Injection

**🎯 Learning Objectives from Handbook**

After completing the module, you will be able to:

* ✅ Recognize and apply **Creational Patterns**
* ✅ Use **Structural Patterns** to build robust systems
* ✅ Employ **Behavioral Patterns** for effective object interaction
* ✅ Improve code reusability and reduce duplication through design patterns
* ✅ Solve common software design problems using appropriate patterns
* ✅ Evaluate and choose the right design pattern for a given problem context