**Exercise 1**

**2.** **Two use cases to demonstrate two creational design pattern**

**Use Case 1: Factory Pattern**

**Problem:** In a e-commerce system, we want to create objects of different types of products (e.g., Book, CD, DVD) without exposing the creation logic to the client code.

**Solution:** We can use the Factory pattern to achieve this.

// Abstract Product interface

public interface Product {

void display();

}

// Concrete Products

public class Book implements Product {

public void display() {

System.out.println("Book");

}

}

public class CD implements Product {

public void display() {

System.out.println("CD");

}

}

public class DVD implements Product {

public void display() {

System.out.println("DVD");

}

}

// Abstract Factory interface

public abstract class Factory {

public abstract Product createProduct();

}

// Concrete Factories

public class BookFactory extends Factory {

@Override

public Product createProduct() {

return new Book();

}

}

public class CDFactory extends Factory {

@Override

public Product createProduct() {

return new CD();

}

}

public class DVDFactory extends Factory {

@Override

public Product createProduct() {

return new DVD();

}

}

// Client code

public class ECommerceSystem {

public static void main(String[] args) {

Factory factory = new BookFactory();

Product product = factory.createProduct();

product.display(); // Output: Book

factory = new CDFactory();

product = factory.createProduct();

product.display(); // Output: CD

factory = new DVDFactory();

product = factory.createProduct();

product.display(); // Output: DVD

}

}

**Use Case 2: Singleton Pattern**

**Problem:** In a logging system, we want to ensure that only one instance of the logger is created and shared across the application.

**Solution:** We can use the Singleton pattern to achieve this.

public class Logger {

private static Logger instance;

private String message;

private Logger() {}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

this.message = message;

System.out.println("Logger: " + message);

}

}

// Client code

public class LoggingSystem {

public static void main(String[] args) {

Logger logger = Logger.getInstance();

logger.log("Hello, world!"); // Output: Logger: Hello, world!

logger = Logger.getInstance(); // Same instance is returned

logger.log("Hello again!"); // Output: Logger: Hello again!

}

}