Week 1

# Design principles and Patterns

# Exercise 1: Singleton Pattern - Logger Class

// Logger.java  
public class Logger {  
 private static Logger instance;  
  
 private Logger() {}  
  
 public static Logger getInstance() {  
 if (instance == null) {  
 instance = new Logger();  
 }  
 return instance;  
 }  
  
 public void log(String message) {  
 System.out.println("Log: " + message);  
 }  
}  
  
// TestLogger.java  
public class TestLogger {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.getInstance();  
 Logger logger2 = Logger.getInstance();  
 logger1.log("This is a log message.");  
 System.out.println(logger1 == logger2); // true  
 }  
}

# Output:

# Exercise 2: Factory Method Pattern - Document Creation

// Document.java  
public interface Document {  
 void open();  
}  
  
// WordDocument.java  
public class WordDocument implements Document {  
 public void open() {  
 System.out.println("Opening Word Document");  
 }  
}  
  
// PdfDocument.java  
public class PdfDocument implements Document {  
 public void open() {  
 System.out.println("Opening PDF Document");  
 }  
}  
  
// ExcelDocument.java  
public class ExcelDocument implements Document {  
 public void open() {  
 System.out.println("Opening Excel Document");  
 }  
}  
  
// DocumentFactory.java  
public abstract class DocumentFactory {  
 public abstract Document createDocument();  
}  
  
// WordFactory.java  
public class WordFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new WordDocument();  
 }  
}  
  
// PdfFactory.java  
public class PdfFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new PdfDocument();  
 }  
}  
  
// ExcelFactory.java  
public class ExcelFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new ExcelDocument();  
 }  
}  
  
// TestFactory.java  
public class TestFactory {  
 public static void main(String[] args) {  
 DocumentFactory factory = new WordFactory();  
 Document doc = factory.createDocument();  
 doc.open();  
 }  
}

# Output:

# Algorithms & Data structures

# Exercise 7: Inventory Management System

// Product.java  
public class Product {  
 private int productId;  
 private String productName;  
 private int quantity;  
 private double price;  
  
 public Product(int productId, String productName, int quantity, double price) {  
 this.productId = productId;  
 this.productName = productName;  
 this.quantity = quantity;  
 this.price = price;  
 }  
  
 public int getProductId() { return productId; }  
 public String getProductName() { return productName; }  
 public int getQuantity() { return quantity; }  
 public double getPrice() { return price; }  
 public void setQuantity(int quantity) { this.quantity = quantity; }  
 public void setPrice(double price) { this.price = price; }  
  
 public String toString() {  
 return productId + " - " + productName + " - Qty: " + quantity + " - Price: " + price;  
 }  
}  
  
// Inventory.java  
import java.util.HashMap;  
  
public class Inventory {  
 private HashMap<Integer, Product> products = new HashMap<>();  
  
 public void addProduct(Product p) {  
 products.put(p.getProductId(), p);  
 }  
  
 public void updateProduct(int productId, int quantity, double price) {  
 Product p = products.get(productId);  
 if (p != null) {  
 p.setQuantity(quantity);  
 p.setPrice(price);  
 }  
 }  
  
 public void deleteProduct(int productId) {  
 products.remove(productId);  
 }  
  
 public void displayProducts() {  
 for (Product p : products.values()) {  
 System.out.println(p);  
 }  
 }  
}  
  
// Main.java  
public class Main {  
 public static void main(String[] args) {  
 Inventory inventory = new Inventory();  
 inventory.addProduct(new Product(101, "Monitor", 10, 8000));  
 inventory.addProduct(new Product(102, "Keyboard", 50, 500));  
 inventory.updateProduct(101, 12, 7800);  
 inventory.deleteProduct(102);  
 inventory.displayProducts();  
 }  
}

# Output:

# Exercise 2: E-commerce Platform Search Function

// Product.java  
public class Product {  
 int productId;  
 String productName;  
 String category;  
  
 public Product(int productId, String productName, String category) {  
 this.productId = productId;  
 this.productName = productName;  
 this.category = category;  
 }  
  
 public String toString() {  
 return productId + " - " + productName + " - " + category;  
 }  
}  
  
// SearchAlgorithms.java  
public class SearchAlgorithms {  
 public static Product linearSearch(Product[] products, String name) {  
 for (Product p : products) {  
 if (p.productName.equalsIgnoreCase(name)) {  
 return p;  
 }  
 }  
 return null;  
 }  
  
 public static Product binarySearch(Product[] products, String name) {  
 int low = 0, high = products.length - 1;  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 int cmp = products[mid].productName.compareToIgnoreCase(name);  
 if (cmp == 0) return products[mid];  
 else if (cmp < 0) low = mid + 1;  
 else high = mid - 1;  
 }  
 return null;  
 }  
}  
  
// Main.java  
import java.util.Arrays;  
  
public class Main {  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(201, "Mouse", "Electronics"),  
 new Product(202, "Phone", "Mobile"),  
 new Product(203, "Laptop", "Electronics"),  
 new Product(204, "Tablet", "Mobile")  
 };  
  
 Product result1 = SearchAlgorithms.linearSearch(products, "Laptop");  
 System.out.println("Linear Search Result: " + result1);  
  
 Arrays.sort(products, (a, b) -> a.productName.compareToIgnoreCase(b.productName));  
 Product result2 = SearchAlgorithms.binarySearch(products, "Laptop");  
 System.out.println("Binary Search Result: " + result2);  
 }  
}

# Output: