# Design Patterns & Algorithm Exercises

## Exercise 1: Singleton Pattern

The Singleton Pattern ensures only one instance of a class exists and provides global access to it.

### Java Code: Logger.java

public class Logger {  
 private static Logger instance;  
  
 private Logger() {  
 System.out.println("Logger Initialized");  
 }  
  
 public static Logger getInstance() {  
 if (instance == null) {  
 instance = new Logger();  
 }  
 return instance;  
 }  
  
 public void log(String message) {  
 System.out.println("[LOG] " + message);  
 }  
}

### Java Code: Main.java

public class Main {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.getInstance();  
 logger1.log("First message");  
  
 Logger logger2 = Logger.getInstance();  
 logger2.log("Second message");  
  
 System.out.println("Are both loggers the same instance? " + (logger1 == logger2));  
 }  
}

output

A screenshot of a computer

AI-generated content may be incorrect.

## Exercise 2: Factory Method Pattern

The Factory Method Pattern provides an interface for creating objects in a superclass but allows subclasses to alter the type of objects that will be created.

### Java Code: Document.java

public interface Document {  
 void open();  
}

### Java Code: WordDocument, PdfDocument, ExcelDocument

public class WordDocument implements Document {  
 public void open() {  
 System.out.println("Opening Word document...");  
 }  
}  
  
public class PdfDocument implements Document {  
 public void open() {  
 System.out.println("Opening PDF document...");  
 }  
}  
  
public class ExcelDocument implements Document {  
 public void open() {  
 System.out.println("Opening Excel document...");  
 }  
}

### Java Code: DocumentFactory.java

public abstract class DocumentFactory {  
 public abstract Document createDocument();  
}

### Java Code: Concrete Factories

public class WordDocumentFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new WordDocument();  
 }  
}  
  
public class PdfDocumentFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new PdfDocument();  
 }  
}  
  
public class ExcelDocumentFactory extends DocumentFactory {  
 public Document createDocument() {  
 return new ExcelDocument();  
 }  
}

### Java Code: Main.java

public class Main {  
 public static void main(String[] args) {  
 DocumentFactory wordFactory = new WordDocumentFactory();  
 Document word = wordFactory.createDocument();  
 word.open();  
  
 DocumentFactory pdfFactory = new PdfDocumentFactory();  
 Document pdf = pdfFactory.createDocument();  
 pdf.open();  
  
 DocumentFactory excelFactory = new ExcelDocumentFactory();  
 Document excel = excelFactory.createDocument();  
 excel.open();  
 }  
}

output

A screenshot of a computer

AI-generated content may be incorrect.

## Exercise 3: E-commerce Platform Search Function

This exercise focuses on implementing linear and binary search for product lookup in an e-commerce platform.

### Java Code: 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;  
 }  
  
 @Override  
 public String toString() {  
 return "Product{" + productId + ", " + productName + ", " + category + "}";  
 }  
}

### Java Code: SearchUtils.java

public class SearchUtils {  
 public static Product linearSearch(Product[] products, String targetName) {  
 for (Product product : products) {  
 if (product.productName.equalsIgnoreCase(targetName)) {  
 return product;  
 }  
 }  
 return null;  
 }  
  
 public static Product binarySearch(Product[] products, String targetName) {  
 int low = 0, high = products.length - 1;  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 int compare = products[mid].productName.compareToIgnoreCase(targetName);  
 if (compare == 0) return products[mid];  
 else if (compare < 0) low = mid + 1;  
 else high = mid - 1;  
 }  
 return null;  
 }  
}

### Java Code: Main.java

import java.util.Arrays;  
import java.util.Comparator;  
  
public class Main {  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(101, "Laptop", "Electronics"),  
 new Product(102, "Shampoo", "Beauty"),  
 new Product(103, "Shoes", "Footwear"),  
 new Product(104, "Phone", "Electronics"),  
 new Product(105, "T-Shirt", "Clothing")  
 };  
  
 Product foundLinear = SearchUtils.linearSearch(products, "Phone");  
 System.out.println(foundLinear != null ? foundLinear : "Not Found");  
  
 Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));  
 Product foundBinary = SearchUtils.binarySearch(products, "Phone");  
 System.out.println(foundBinary != null ? foundBinary : "Not Found");  
 }  
}

output:

A screenshot of a computer

AI-generated content may be incorrect.

## Exercise 7: Financial Forecasting

This exercise uses recursion to forecast future financial values based on annual growth.

### Java Code: FinancialForecast.java

public class FinancialForecast {  
 public static double futureValueRecursive(double presentValue, double rate, int years) {  
 if (years == 0) return presentValue;  
 return futureValueRecursive(presentValue, rate, years - 1) \* (1 + rate);  
 }  
  
 public static double futureValueMemo(double presentValue, double rate, int years, double[] memo) {  
 if (years == 0) return presentValue;  
 if (memo[years] != 0) return memo[years];  
 memo[years] = futureValueMemo(presentValue, rate, years - 1, memo) \* (1 + rate);  
 return memo[years];  
 }  
}

### Java Code: Main.java

public class Main {  
 public static void main(String[] args) {  
 double presentValue = 10000;  
 double rate = 0.08;  
 int years = 10;  
  
 double resultRecursive = FinancialForecast.futureValueRecursive(presentValue, rate, years);  
 System.out.println("Recursive Future Value: ₹" + resultRecursive);  
  
 double[] memo = new double[years + 1];  
 double resultMemo = FinancialForecast.futureValueMemo(presentValue, rate, years, memo);  
 System.out.println("Memoized Future Value: ₹" + resultMemo);  
 }  
}

output

A screenshot of a computer

AI-generated content may be incorrect.