**Design Pattern and Principles:**

**Exercise 1: Implementing the Singleton Pattern**

**Program:**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);  
 }  
}

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.");  
  
 if (logger1 == logger2) {  
 System.*out*.println("Both loggers are the same instance.");  
 } else {  
 System.*out*.println("Different logger instances found.");  
 }  
 }  
}

**Output:**

Logger initialized.

[LOG] First message.

[LOG] Second message.

Both loggers are the same instance.

**Exercise 2: Implementing the Factory Method Pattern**

public interface Document {

void open();

}

public class WordDocument implements Document {

@Override

public void open() {

System.out.println("Opening Word Document...");

}

}

public class PdfDocument implements Document {

@Override

public void open() {

System.out.println("Opening PDF Document...");

}

}

public class ExcelDocument implements Document {

@Override

public void open() {

System.out.println("Opening Excel Document...");

}

}

public abstract class DocumentFactory {

public abstract Document createDocument();

}

public class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

public class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

public class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Output:**

Opening Word Document...

Opening PDF Document...

Opening Excel Document...

**Data Structures and Algorithm**

**Exercise 2: E-commerce Platform Search Engine**

**Program:**  
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 "[" + productId + ", " + productName + ", " + category + "]";  
 }  
  
  
 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 - low) / 2;  
 int comparison = products[mid].productName.compareToIgnoreCase(targetName);  
 if (comparison == 0) {  
 return products[mid];  
 } else if (comparison < 0) {  
 low = mid + 1;  
 } else {  
 high = mid - 1;  
 }  
 }  
 return null;  
 }  
}

import java.util.\*;  
  
import static Week1.Product.*binarySearch*;  
import static Week1.Product.*linearSearch*;  
  
public class Main {  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(101, "Laptop", "Electronics"),  
 new Product(102, "Shoes", "Footwear"),  
 new Product(103, "Mobile", "Electronics"),  
 new Product(104, "Watch", "Accessories")  
 };  
  
  
 Product result1 = *linearSearch*(products, "Mobile");  
 System.*out*.println("Linear Search Result: " + result1);  
  
  
 Arrays.*sort*(products, Comparator.*comparing*(p -> p.productName.toLowerCase()));  
 Product result2 = *binarySearch*(products, "Mobile");  
 System.*out*.println("Binary Search Result: " + result2);  
 }  
}

**Output:**

Linear Search Result: [103, Mobile, Electronics]

Binary Search Result: [103, Mobile, Electronics]

**Time Complexity:**

Linear Search; O(n)

Binary Search: O(logn)

**Exercise 7: Financial Forecasting**

public class FinancialForecast {

**Recursive method**

public static double futureValue(double presentValue, double growthRate, int years) {

if (years == 0) {

return presentValue;

}

return futureValue(presentValue \* (1 + growthRate), growthRate, years - 1);

}

public static void main(String[] args) {

double presentValue = 5000; // initial amount

double rate = 0.06; // 6% annual growth

int years = 10;

double result = futureValue(presentValue, rate, years);

System.out.printf("Predicted future value after %d years: ₹%.2f%n", years, result);

}

}

**Iterative Method:**

public static double futureValueIterative(double presentValue, double growthRate, int years) {

for (int i = 0; i < years; i++) {

presentValue \*= (1 + growthRate);

}

return presentValue;

}

**Output:**

Predicted future value after 10 years: ₹8954.24