**Hands On Practice**

WEEK 1

1. Design Principles

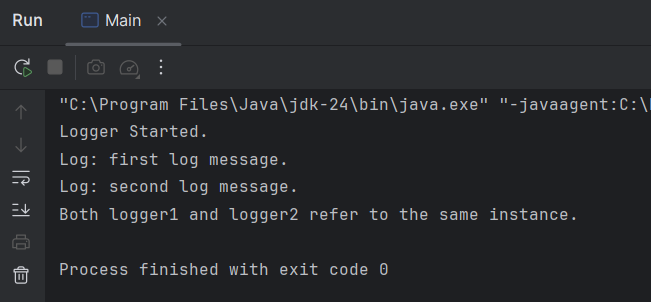
**Exercise 1: Implementing the Singleton Pattern**

**# Logger Class**

public class Logger {  
 private static Logger *log*;  
  
 private Logger() {  
 System.*out*.println("Logger Started.");  
 }  
  
 public static Logger getInstance() {  
 if (*log* == null) {  
 *log* = new Logger();  
 }  
 return *log*;  
 }  
  
 public void log(String message) {  
 System.*out*.println("Log: " + message);  
 }  
}

**# Main Class**

public class Main {  
 public static void main(String[] args) {  
 Logger logger1 = Logger.*getInstance*();  
 logger1.log("first log message.");  
  
 Logger logger2 = Logger.*getInstance*();  
 logger2.log("second log message.");  
  
 if (logger1 == logger2) {  
 System.*out*.println("Both logger1 and logger2 refer to the same instance.");  
 } else {  
 System.*out*.println("Different instances exist!");  
 }  
 }  
}

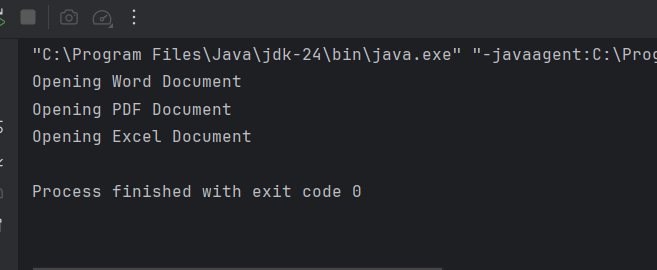
OUTPUT

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

public interface Document {  
 void open();  
}

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

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

**OUTPUT**

1. **Data Structures and Algorithms**

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

**# Product Class**  
public class Product {  
 private int productId;  
 private String productName;  
 private String category;  
  
 public Product(int id, String name, String category) {  
 this.productId = id;  
 this.productName = name;  
 this.category = category;  
 }  
 public int getProductId() {  
 return productId;  
 }  
 public String getProductName() {  
 return productName;  
 }  
 public String getCategory() {  
 return category;  
 }  
  
 @Override  
 public String toString() {  
 return productId + " - " + productName + " (" + category + ")";  
 }  
}

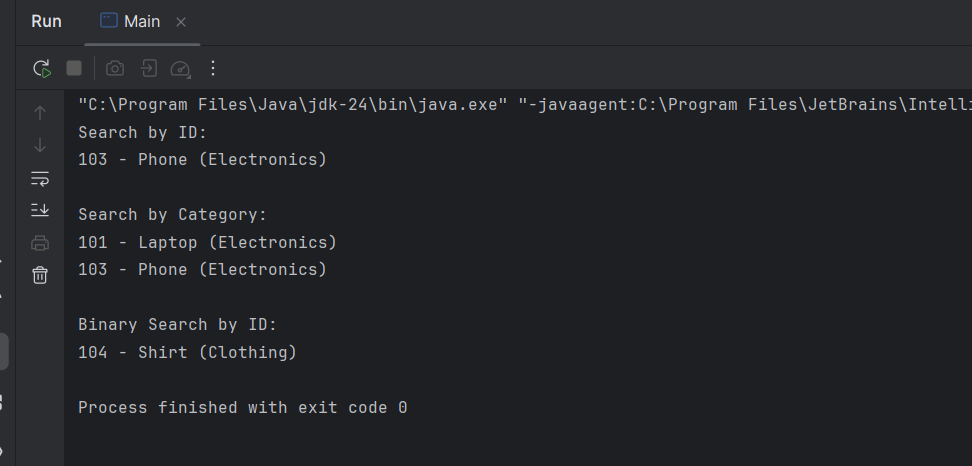
**# Seach Class**

import java.util.Arrays;  
import java.util.Comparator;  
  
public class Search {  
 public static Product linearSearchByName(Product[] products, String targetName) {  
 for (Product product : products) {  
 if (product.getProductName().equalsIgnoreCase(targetName)) {  
 return product;  
 }  
 }  
 return null;  
 }  
 public static Product linearSearchById(Product[] products, int targetId) {  
 for (Product product : products) {  
 if (product.getProductId() == targetId) {  
 return product;  
 }  
 }  
 return null;  
 }  
 public static Product[] linearSearchByCategory(Product[] products, String targetCategory) {  
 return Arrays.*stream*(products)  
 .filter(p -> p.getCategory().equalsIgnoreCase(targetCategory))  
 .toArray(Product[]::new);  
 }  
 public static Product binarySearchByName(Product[] products, String targetName) {  
 Arrays.*sort*(products, Comparator.*comparing*(Product::getProductName));  
 int left = 0, right = products.length - 1;  
  
 while (left <= right) {  
 int mid = (left + right) / 2;  
 String midName = products[mid].getProductName();  
 int cmp = midName.compareToIgnoreCase(targetName);  
  
 if (cmp == 0) return products[mid];  
 else if (cmp < 0) left = mid + 1;  
 else right = mid - 1;  
 }  
 return null;  
 }  
 public static Product binarySearchById(Product[] products, int targetId) {  
 Arrays.*sort*(products, Comparator.*comparingInt*(Product::getProductId));  
 int left = 0, right = products.length - 1;  
  
 while (left <= right) {  
 int mid = (left + right) / 2;  
 int midId = products[mid].getProductId();  
  
 if (midId == targetId) return products[mid];  
 else if (midId < targetId) left = mid + 1;  
 else right = mid - 1;  
 }  
 return null;  
 }  
}

**# Main Class**

public class Main {  
 public static void main(String[] args) {  
 Product[] products = {  
 new Product(101, "Laptop", "Electronics"),  
 new Product(102, "Shoes", "Footwear"),  
 new Product(103, "Phone", "Electronics"),  
 new Product(104, "Shirt", "Clothing")  
 };  
  
 System.*out*.println("Search by ID:");  
 System.*out*.println(Search.*linearSearchById*(products, 103));  
  
 System.*out*.println("\nSearch by Category:");  
 Product[] electronics = Search.*linearSearchByCategory*(products, "Electronics");  
 for (Product p : electronics) System.*out*.println(p);  
  
 System.*out*.println("\nBinary Search by ID:");  
 System.*out*.println(Search.*binarySearchById*(products, 104));  
 }  
}

**OUTPUT**



**Exercise 7: Financial Forecasting**

public class Main {  
  
 public static double futureValue(int n, double initialValue, double growthRate) {  
 if (n == 0) return initialValue;  
 return *futureValue*(n - 1, initialValue, growthRate) \* (1 + growthRate);  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.print("Enter the initial amount: ");  
 double initial = scanner.nextDouble();  
  
 System.*out*.print("Enter the annual growth rate (e.g., 0.05 for 5%): ");  
 double rate = scanner.nextDouble();  
  
 System.*out*.print("Enter the number of years: ");  
 int years = scanner.nextInt();  
  
 double result = *futureValue*(years, initial, rate);  
 System.*out*.printf("Future Value after %d years: $%.2f\n", years, result);  
  
 scanner.close();  
 }  
}

**OUTPUT**

