Design Principles and Patterns

Exercise 1: Implementing the Singleton Pattern

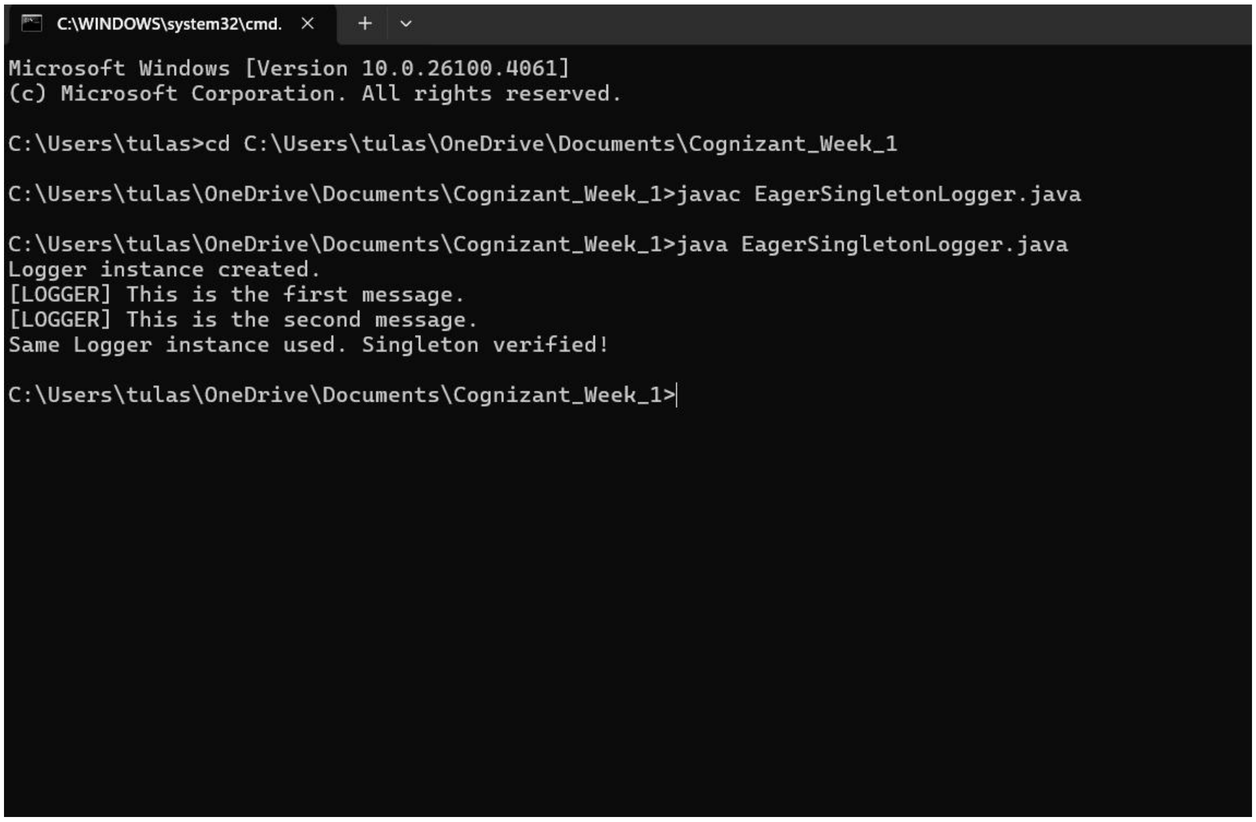
Scenario:

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

PROGRAM:

```
public class EagerSingletonLogger {
    static class Logger {
        private static final Logger instance = new Logger();
        private Logger() {
            System.out.println("Logger instance created.");
        }
        public static Logger getInstance() {
            return instance;
        }
        public void log(String message) {
            System.out.println("[LOGGER] " + message);
        }
    }
    public static void main(String[] args) {
        Logger loggerA = Logger.getInstance();
        loggerA.log("This is the first message.");
        Logger loggerB = Logger.getInstance();
        loggerB.log("This is the second message.");
        if (loggerA == loggerB) {
            System.out.println("Same Logger instance used. Singleton verified!");
        } else {
            System.out.println("Different Logger instances. Singleton failed!");
        }
    }
}
```

OUTPUT:



C:\WINDOWS\system32\cmd.

```
Microsoft Windows [Version 10.0.26100.4061]
(c) Microsoft Corporation. All rights reserved.

C:\Users\tulas>cd C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>javac EagerSingletonLogger.java

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>java EagerSingletonLogger.java
Logger instance created.
[LOGGER] This is the first message.
[LOGGER] This is the second message.
Same Logger instance used. Singleton verified!

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>
```

Exercise 2: Implementing the Factory Method Pattern

Scenario:

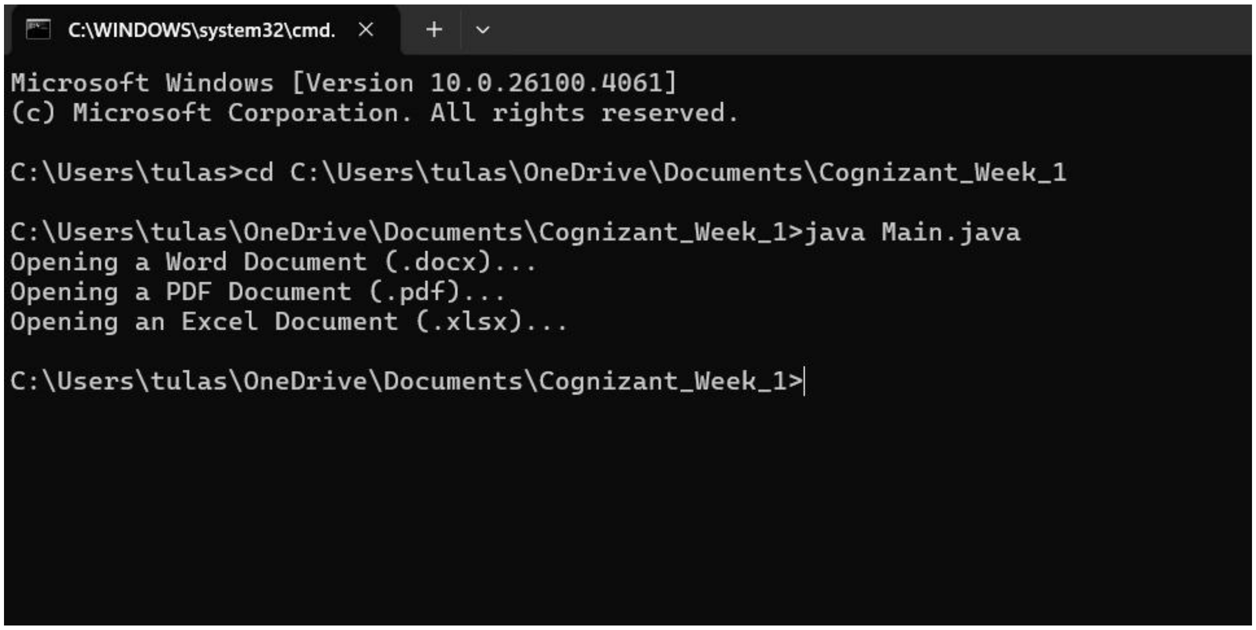
You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

PROGRAM:

```
interface Document {
    void render();
}

class WordDocument implements Document {
    public void render() {
        System.out.println("Opening a Word Document (.docx)...");
    }
}

class PdfDocument implements Document {
    public void render() {
        System.out.println("Opening a PDF Document (.pdf)...");
    }
}

class ExcelDocument implements Document {
    public void render() {
        System.out.println("Opening an Excel Document (.xlsx)...");
    }
}

interface DocumentFactory {
    Document createDocument();
}
```

```
class WordFactory implements DocumentFactory {
    public Document createDocument() {
        return new WordDocument();
    }
}

class PdfFactory implements DocumentFactory {
    public Document createDocument() {
        return new PdfDocument();
    }
}

class ExcelFactory implements DocumentFactory {
    public Document createDocument() {
        return new ExcelDocument();
    }
}

public class Main {
    public static void main(String[] args) {
        DocumentFactory wordFactory = new WordFactory();
        Document word = wordFactory.createDocument();
        word.render();
        DocumentFactory pdfFactory = new PdfFactory();
        Document pdf = pdfFactory.createDocument();
        pdf.render();



        DocumentFactory excelFactory = new ExcelFactory();
        Document excel = excelFactory.createDocument();
        excel.render();
    }
}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.26100.4061]
(c) Microsoft Corporation. All rights reserved.

C:\Users\tulas>cd C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>java Main.java
Opening a Word Document (.docx)...
Opening a PDF Document (.pdf)...
Opening an Excel Document (.xlsx)...

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>
```

Data Structures and Algorithms

Exercise 1: E-commerce Platform Search Function

Scenario:

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

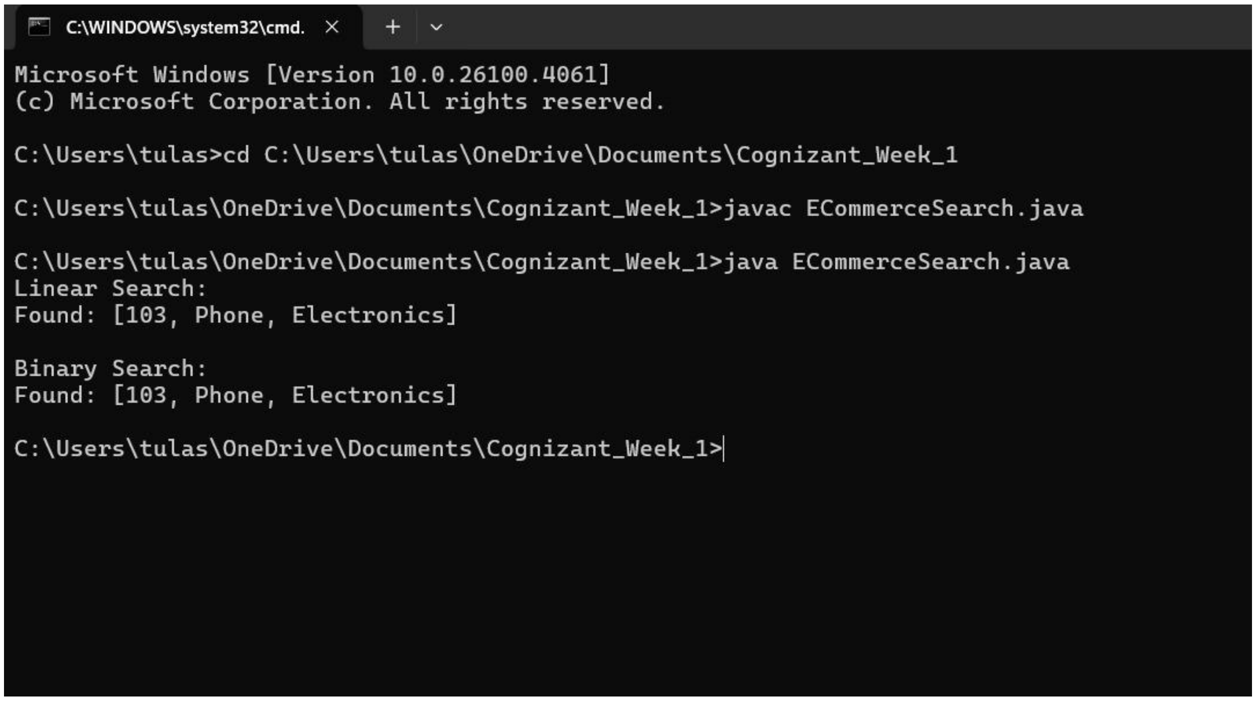
PROGRAM:

```
import java.util.Arrays;
import java.util.Comparator;
public class ECommerceSearch {
    static class Product {
        int productId;
        String productName;
        String category;
        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 name) {
        for (Product product : products) {
            if (product.productName.equalsIgnoreCase(name)) {
                return product;
            }
        }
        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;
}
public static void main(String[] args) {
    Product[] products = {
        new Product(101, "Laptop", "Electronics"),
        new Product(102, "Chair", "Furniture"),
        new Product(103, "Phone", "Electronics"),
        new Product(104, "Table", "Furniture"),
        new Product(105, "Headphones", "Electronics")
    };
    System.out.println("Linear Search:");
    Product result1 = linearSearch(products, "Phone");
```

```
        System.out.println(result1 != null ? "Found: " + result1 : "Product not found.");

        Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

        System.out.println("\nBinary Search:");

        Product result2 = binarySearch(products, "Phone");

        System.out.println(result2 != null ? "Found: " + result2 : "Product not found.");

    }

}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.26100.4061]
(c) Microsoft Corporation. All rights reserved.

C:\Users\tulas>cd C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>javac ECommerceSearch.java

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>java ECommerceSearch.java
Linear Search:
Found: [103, Phone, Electronics]

Binary Search:
Found: [103, Phone, Electronics]

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>
```

Exercise 2: Financial Forecasting

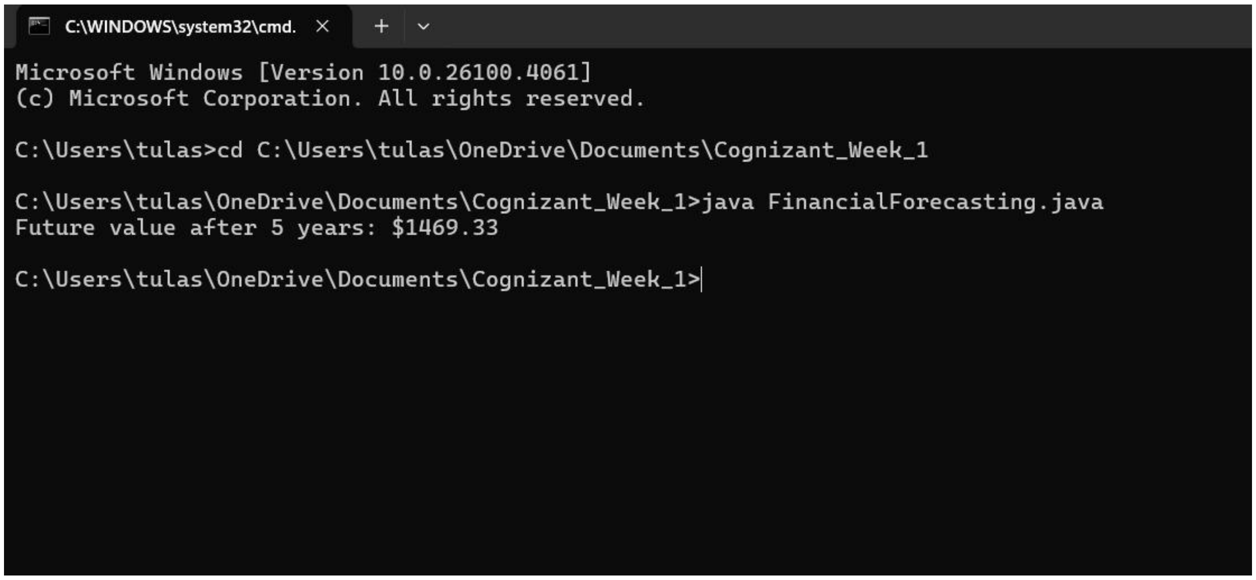
Scenario:

You are developing a financial forecasting tool that predicts future values based on past data.

PROGRAM:

```
public class FinancialForecasting {
   public static double forecastRecursive(double currentValue, double growthRate,
int years) {
       if (years == 0) {
          return currentValue;
       }
       return forecastRecursive(currentValue, growthRate, years - 1) * (1 +
growthRate);
   }
public static void main(String[] args) {
       double presentValue = 1000.0;
       double annualGrowthRate = 0.08;
       int forecastYears = 5;
       double futureValue = forecastRecursive(presentValue, annualGrowthRate,
forecastYears);
       System.out.printf("Future value after %d years: $%.2f\n", forecastYears,
futureValue);
   }
}
```

OUTPUT:



C:\WINDOWS\system32\cmd.

```
Microsoft Windows [Version 10.0.26100.4061]
(c) Microsoft Corporation. All rights reserved.

C:\Users\tulas>cd C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>java FinancialForecasting.java
Future value after 5 years: $1469.33

C:\Users\tulas\OneDrive\Documents\Cognizant_Week_1>
```