DESIGN PRINCIPLES AND PATTERN

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.

```java
interface Document {
    void open();
}

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

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

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

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

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

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

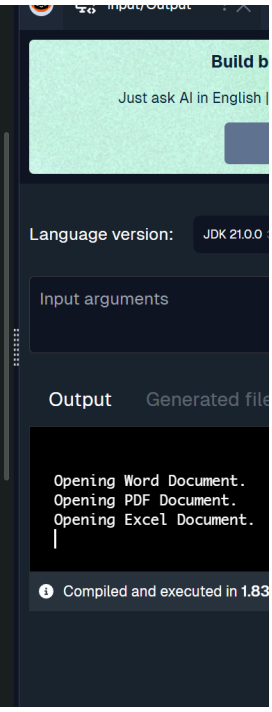
```
Build beautiful
Just ask AI in English | Español |
Build wi
Language version: JDK 21.0.0
Input arguments
Output  Generated files
Opening Word Document.
Opening PDF Document.
Opening Excel Document.
Compiled and executed in 1.83 sec(s)
```

Exercise 2: Implementing the Factory Method Pattern

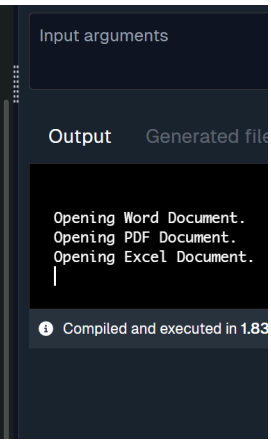
Scenario:

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.

```java
interface Document {
    void open();
}

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

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

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

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

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

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




```java
class ExcelFactory extends DocumentFactory {
    public Document createDocument() {
        return new ExcelDocument();
    }
}

public class Main {
    public static void main(String[] args) {
        DocumentFactory wordFactory = new WordFactory();
        Document wordDoc = wordFactory.createDocument();
        wordDoc.open();

        DocumentFactory pdfFactory = new PdfFactory();
        Document pdfDoc = pdfFactory.createDocument();
        pdfDoc.open();

        DocumentFactory excelFactory = new ExcelFactory();
        Document excelDoc = excelFactory.createDocument();
        excelDoc.open();
    }
}
```



DATA STRUCTURES AND ALGORITHMS

Exercise 2: 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.

```
class Product {
public:
    int productId;
    string productName;
    string category;

    Product(int id, string name, string cat) {
        productId = id;
        productName = name;
        category = cat;
    }

    void display() {
        cout << "Product ID: " << productId
             << ", Name: " << productName
             << ", Category: " << category << endl;
    }
};

Product* linearSearch(vector<Product>& products, string name) {
    for (auto& product : products) {
        if (product.productName == name) {
```

```
---- Linear Search ----
Product ID: 102, Name: Laptop, Category: Electronics
---- Binary Search ----
Product ID: 102, Name: Laptop, Category: Electronics


=== Code Execution Successful ===
```

Exercise 7: Financial Forecasting

Scenario:

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

```
double forecast(double presentValue, double rate, int years) {
    if (years == 0)
        return presentValue;
    return forecast(presentValue, rate, years - 1) * (1 + rate);
}

double forecastMemo(double presentValue, double rate, int years, vector
    <double>& memo) {
    if (years == 0)
        return presentValue;
    if (memo[years] != 0)
        return memo[years];
    memo[years] = forecastMemo(presentValue, rate, years - 1, memo) * (1 +
        rate);
    return memo[years];
}

int main() {
    double presentValue = 10000.0;
    double growthRate = 0.08;
    int years = 5;
```

```
Future Value (Recursion): ₹14693.28
Future Value (Memoized): ₹14693.28


=== Code Execution Successful ===
```