

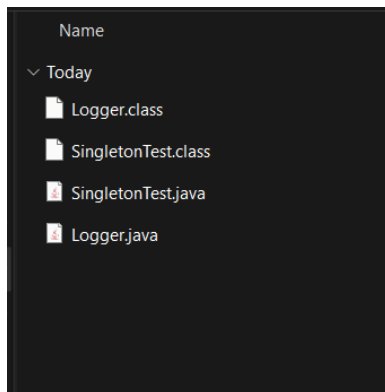
WEEK 1

Exercise 1: Implementing the Singleton Pattern

Command prompt :

```
C:\Users\jutur\Downloads>mkdir SingletonPatternExample  
C:\Users\jutur\Downloads>cd SingletonPatternExample  
C:\Users\jutur\Downloads\SingletonPatternExample>notepad Logger.java  
C:\Users\jutur\Downloads\SingletonPatternExample>notepad SingletonTest.java
```

Files:



Logger.java

```
public class Logger {  
    private static Logger instance;  
  
    private Logger() {  
        System.out.println("Logger instance created for Deekshitha");  
    }  
  
    public static Logger getInstance() {  
        if (instance == null) {  
            instance = new Logger();  
        }  
        return instance;  
    }  
  
    public void log(String message) {  
        System.out.println("LOG: " + message);  
    }  
}
```

SingletonTest.java

```
import java.util.Scanner;

public class SingletonTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter your name: ");
        String name = scanner.nextLine();

        Logger logger1 = Logger.getInstance();
        logger1.log("First log from " + name);

        Logger logger2 = Logger.getInstance();
        logger2.log("Second log from " + name);

        System.out.println("Same instance? " + (logger1 == logger2));
    }
}
```

Output:

```
C:\Users\jutur\Downloads\SingletonPatternExample>javac Logger.java SingletonTest.java
C:\Users\jutur\Downloads\SingletonPatternExample>java SingletonTest
Enter your name: Juturu
Logger instance created for Deekshitha
LOG: First log from Juturu
LOG: Second log from Juturu
Same instance? true
C:\Users\jutur\Downloads\SingletonPatternExample>|
```

Exercise 2: Implementing the Factory Method Pattern

Command prompt:

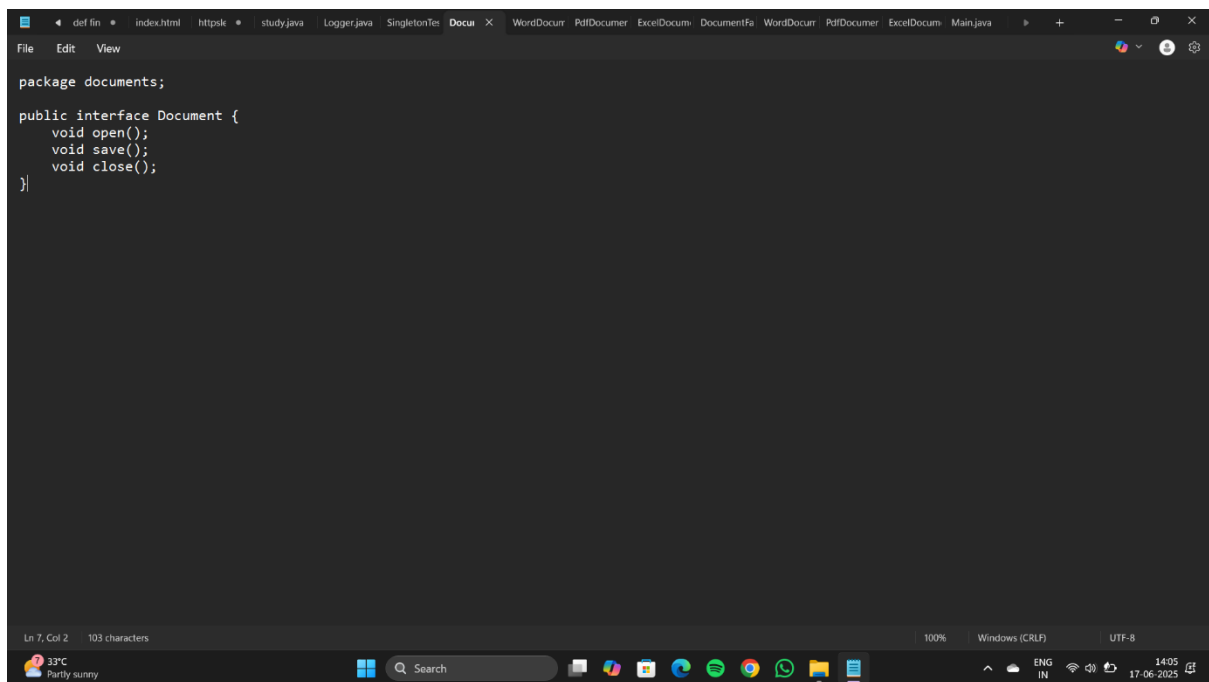
```
Command Prompt
Microsoft Windows [Version 10.0.26100.4061]
(c) Microsoft Corporation. All rights reserved.

C:\Users\jutur>mkdir FactoryMethodPatternExample
C:\Users\jutur>cd FactoryMethodPatternExample
C:\Users\jutur\FactoryMethodPatternExample>mkdir src
C:\Users\jutur\FactoryMethodPatternExample>mkdir src\documents
C:\Users\jutur\FactoryMethodPatternExample>mkdir src\factories
C:\Users\jutur\FactoryMethodPatternExample>echo. > src\documents\Document.java
C:\Users\jutur\FactoryMethodPatternExample>echo. > src\documents\WordDocument.java
C:\Users\jutur\FactoryMethodPatternExample>echo. > src\documents\PdfDocument.java
C:\Users\jutur\FactoryMethodPatternExample>echo. > src\documents\ExcelDocument.java
C:\Users\jutur\FactoryMethodPatternExample>echo. > src\factories\DocumentFactory.java
C:\Users\jutur\FactoryMethodPatternExample>echo. > src\factories\WordDocumentFactory.java
C:\Users\jutur\FactoryMethodPatternExample>echo. > src\factories\PdfDocumentFactory.java
C:\Users\jutur\FactoryMethodPatternExample>echo. > src\factories\ExcelDocumentFactory.java
C:\Users\jutur\FactoryMethodPatternExample>echo. > src\Main.java
```

File hierarchy:

```
C:\Users\jutur\FactoryMethodPatternExample>tree /F
Folder PATH listing for volume Windows
Volume serial number is 90F5-4C94
C:.
├──src
│   ├──Main.java
│   ├──documents
│   │   ├──Document.java
│   │   ├──ExcelDocument.java
│   │   ├──PdfDocument.java
│   │   └──WordDocument.java
│   └──factories
│       ├──DocumentFactory.java
│       ├──ExcelDocumentFactory.java
│       ├──PdfDocumentFactory.java
│       └──WordDocumentFactory.java
```

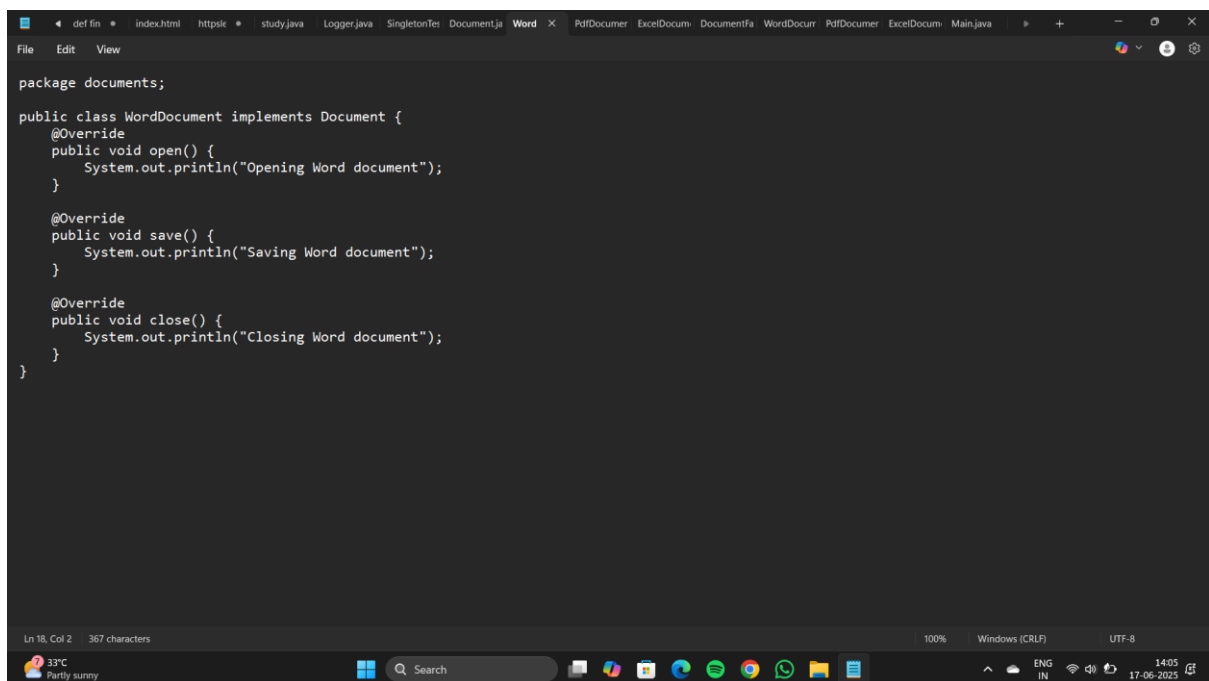
Document.java



```
package documents;

public interface Document {
    void open();
    void save();
    void close();
}
```

WordDocument.java



```
package documents;

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

    @Override
    public void save() {
        System.out.println("Saving Word document");
    }

    @Override
    public void close() {
        System.out.println("Closing Word document");
    }
}
```

PdfDocument.java

```
package documents;

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

    @Override
    public void save() {
        System.out.println("Saving PDF document");
    }

    @Override
    public void close() {
        System.out.println("Closing PDF document");
    }
}
```

Ln 18, Col 2 363 characters 100% Windows (CRLF) UTF-8

33°C Partly sunny 14:05 17-06-2025

ExcelDocument.java

```
package documents;

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

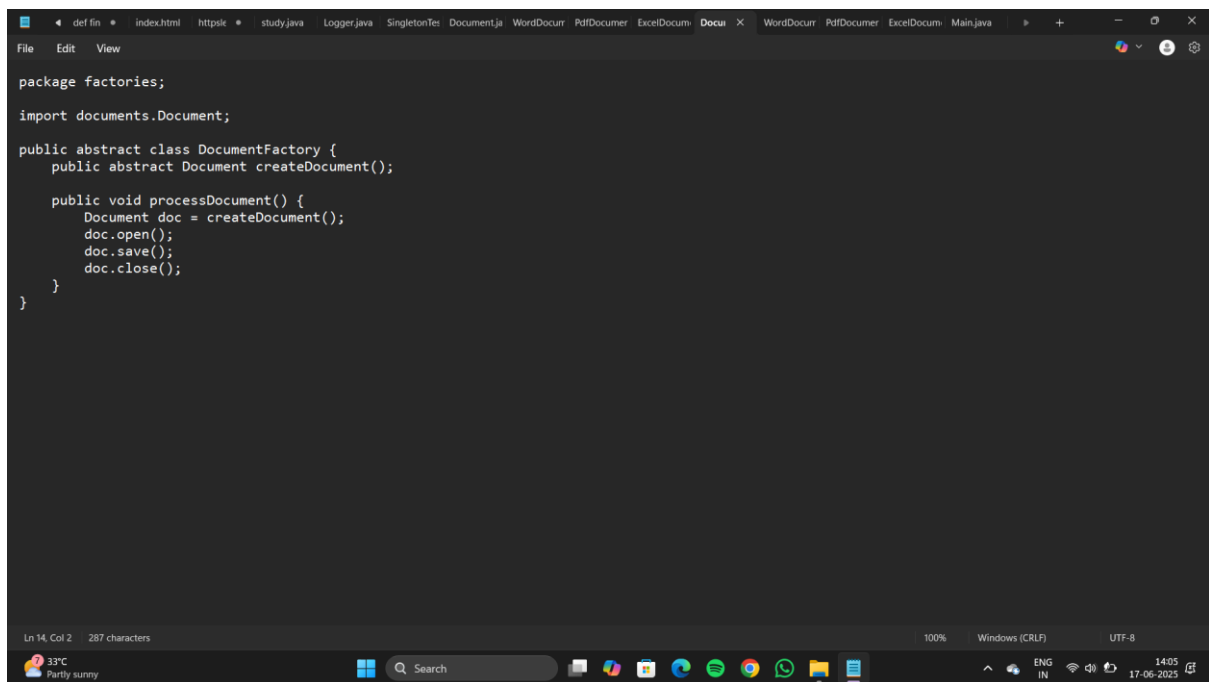
    @Override
    public void save() {
        System.out.println("Saving Excel document");
    }

    @Override
    public void close() {
        System.out.println("Closing Excel document");
    }
}
```

Ln 18, Col 2 371 characters 100% Windows (CRLF) UTF-8

33°C Partly sunny 14:05 17-06-2025

DocumentFactory.java

A screenshot of an IDE window displaying the DocumentFactory.java file. The window has a dark theme and multiple tabs at the top. The code is as follows:

```
package factories;

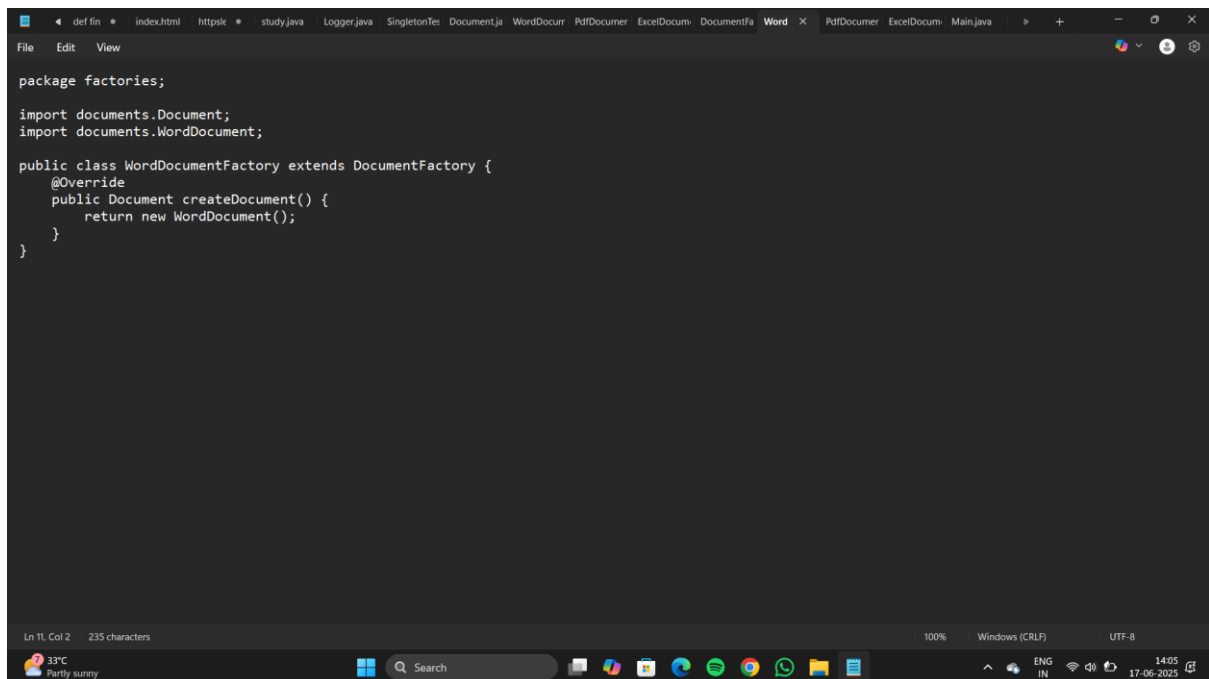
import documents.Document;

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

    public void processDocument() {
        Document doc = createDocument();
        doc.open();
        doc.save();
        doc.close();
    }
}
```

The status bar at the bottom indicates 'Ln 14, Col 2' and '287 characters'. The Windows taskbar is visible at the very bottom with a search bar and various icons.

WordDocumentFactory.java

A screenshot of an IDE window displaying the WordDocumentFactory.java file. The window has a dark theme and multiple tabs at the top. The code is as follows:

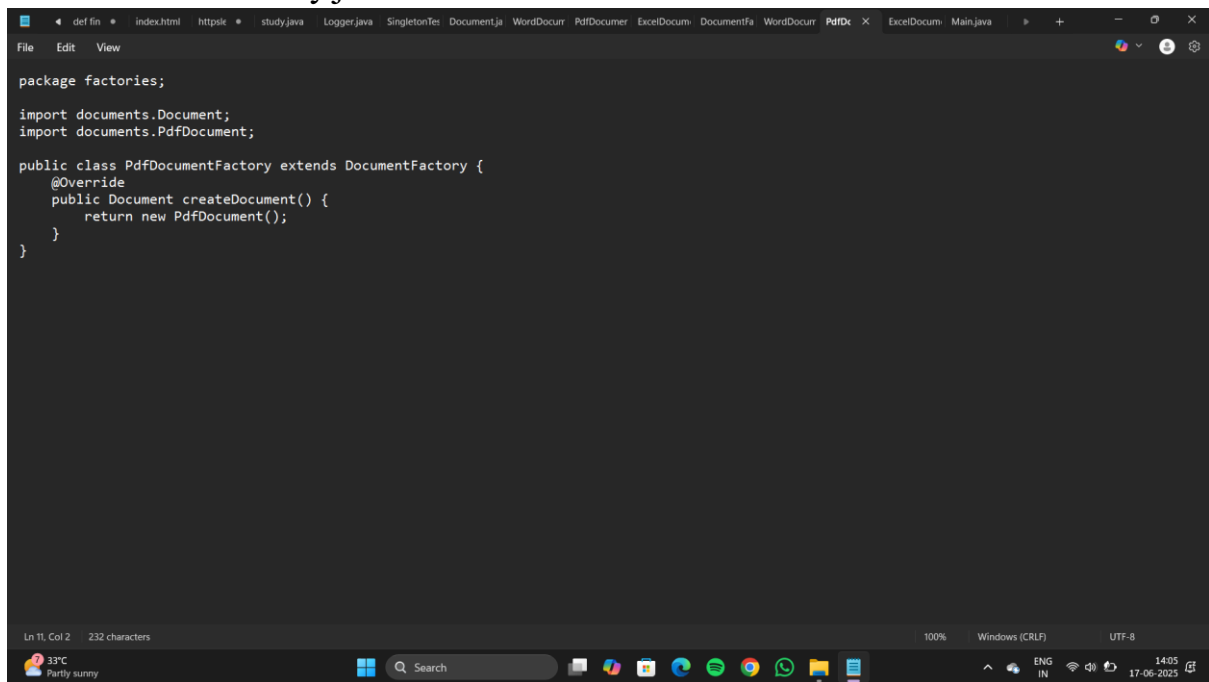
```
package factories;

import documents.Document;
import documents.WordDocument;

public class WordDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new WordDocument();
    }
}
```

The status bar at the bottom indicates 'Ln 11, Col 2' and '235 characters'. The Windows taskbar is visible at the very bottom with a search bar and various icons.

PdfDocumentFactory.java

A screenshot of a code editor window titled 'PdfDocumentFactory.java'. The editor shows the following Java code:

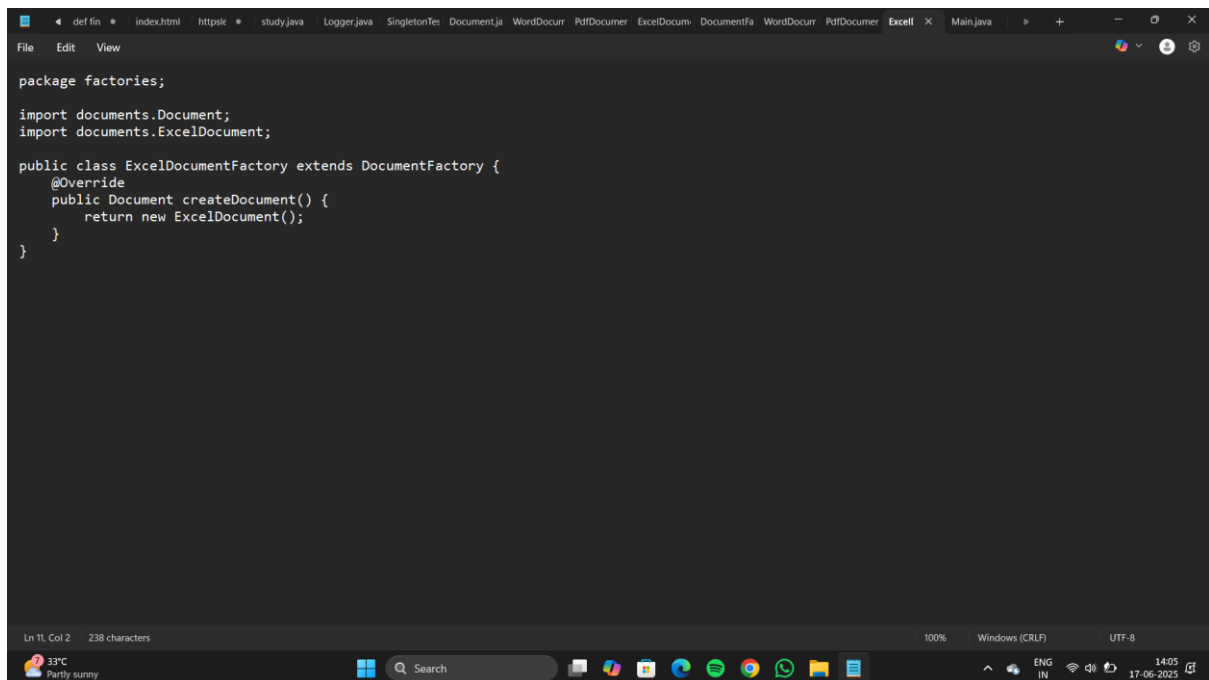
```
package factories;

import documents.Document;
import documents.PdfDocument;

public class PdfDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new PdfDocument();
    }
}
```

The IDE interface includes a menu bar (File, Edit, View), a toolbar with icons for undo, redo, and search, and a status bar at the bottom showing 'Ln 11, Col 2', '232 characters', '100%', 'Windows (CRLF)', and 'UTF-8'. The Windows taskbar is visible at the bottom of the screen.

ExcelDocumentFactory.java

A screenshot of a code editor window titled 'ExcelDocumentFactory.java'. The editor shows the following Java code:

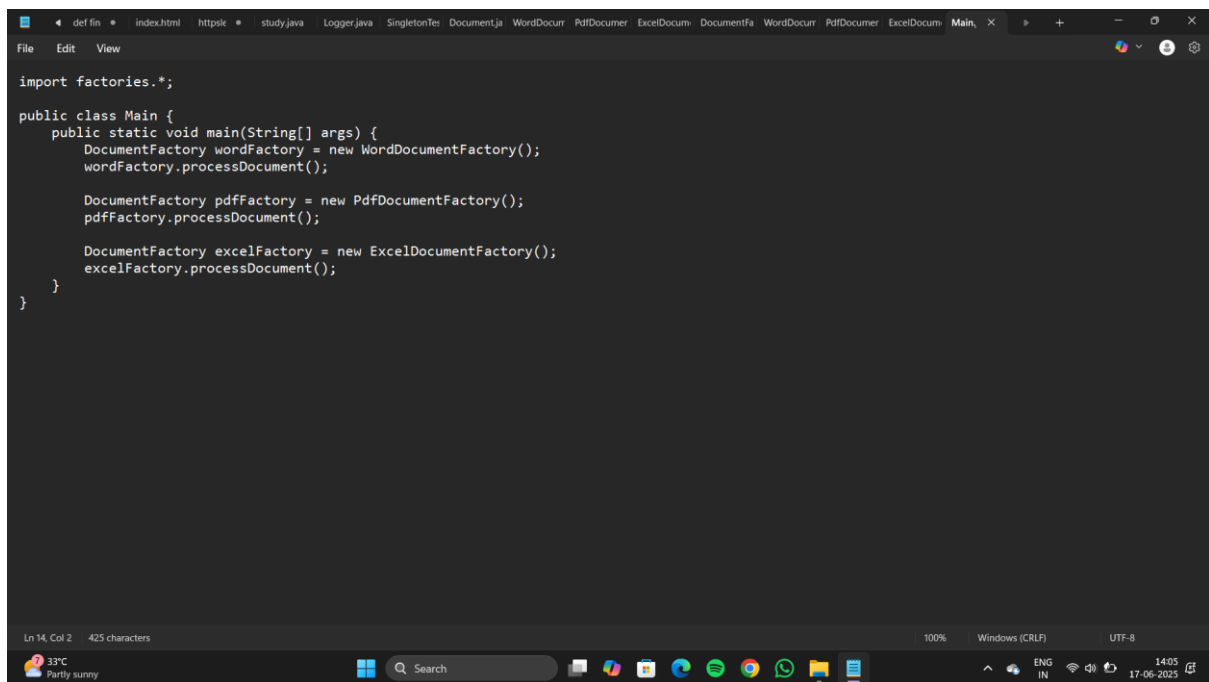
```
package factories;

import documents.Document;
import documents.ExcelDocument;

public class ExcelDocumentFactory extends DocumentFactory {
    @Override
    public Document createDocument() {
        return new ExcelDocument();
    }
}
```

The IDE interface is similar to the previous screenshot, with a menu bar, toolbar, and status bar. The status bar shows 'Ln 11, Col 2', '238 characters', '100%', 'Windows (CRLF)', and 'UTF-8'. The Windows taskbar is also visible at the bottom.

Main.java



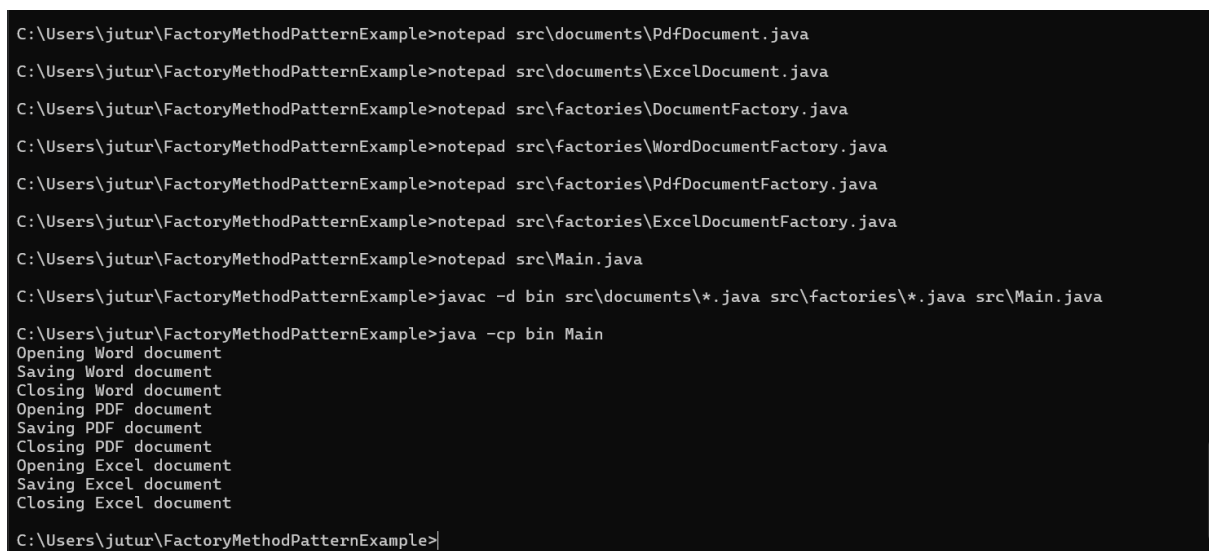
```
import factories.*;

public class Main {
    public static void main(String[] args) {
        DocumentFactory wordFactory = new WordDocumentFactory();
        wordFactory.processDocument();

        DocumentFactory pdfFactory = new PdfDocumentFactory();
        pdfFactory.processDocument();

        DocumentFactory excelFactory = new ExcelDocumentFactory();
        excelFactory.processDocument();
    }
}
```

Output:



```
C:\Users\jutur\FactoryMethodPatternExample>notepad src\documents\PdfDocument.java
C:\Users\jutur\FactoryMethodPatternExample>notepad src\documents\ExcelDocument.java
C:\Users\jutur\FactoryMethodPatternExample>notepad src\factories\DocumentFactory.java
C:\Users\jutur\FactoryMethodPatternExample>notepad src\factories\WordDocumentFactory.java
C:\Users\jutur\FactoryMethodPatternExample>notepad src\factories\PdfDocumentFactory.java
C:\Users\jutur\FactoryMethodPatternExample>notepad src\factories\ExcelDocumentFactory.java
C:\Users\jutur\FactoryMethodPatternExample>notepad src>Main.java
C:\Users\jutur\FactoryMethodPatternExample>javac -d bin src\documents\*.java src\factories\*.java src>Main.java
C:\Users\jutur\FactoryMethodPatternExample>java -cp bin Main
Opening Word document
Saving Word document
Closing Word document
Opening PDF document
Saving PDF document
Closing PDF document
Opening Excel document
Saving Excel document
Closing Excel document
C:\Users\jutur\FactoryMethodPatternExample>
```


Exercise 2: E-commerce Platform Search Function

Steps:

1. Understand Asymptotic Notation:

Big O notation is a mathematical notation used to describe the upper bound of an algorithm's running time or space requirements in terms of the input size. It helps in analyzing algorithms by providing a high-level understanding of their efficiency, allowing developers to compare different approaches and choose the most optimal one for large inputs.

For search operations:

- **Best-case scenario:** The target element is found immediately (e.g., the first element in linear search or the middle element in binary search).
- **Average-case scenario:** The target element is found after checking a significant portion of the elements.
- **Worst-case scenario:** The target element is not present, or it is the last element checked (e.g., checking all elements in linear search or traversing the entire search space in binary search).

2. Setup:

Create a class Product with attributes for searching, such as productId, productName, and category.

```
public class Product {  
    private int productId;  
    private String productName;  
    private String category;  
    public Product(int productId, String productName, String category) {  
        this.productId = productId;  
        this.productName = productName;  
        this.category = category;  
    }  
    public int getProductId() { return productId; }  
    public String getProductName() { return productName; }  
}
```

```
    public String getCategory() { return category; }  
}
```

3. Implementation:

Implement linear search and binary search algorithms. Store products in an array for linear search and a sorted array for binary search.

Linear Search:

```
public static int linearSearch(Product[] products, int targetId) {  
    for (int i = 0; i < products.length; i++) {  
        if (products[i].getProductId() == targetId) {  
            return i;  
        }  
    }  
    return -1;  
}
```

Binary Search (requires sorted array):

```
public static int binarySearch(Product[] sortedProducts, int targetId) {  
    int left = 0;  
    int right = sortedProducts.length - 1;  
  
    while (left <= right) {  
        int mid = left + (right - left) / 2;  
        int midId = sortedProducts[mid].getProductId();  
  
        if (midId == targetId) {  
            return mid;  
        } else if (midId < targetId) {  
            left = mid + 1;  
        }  
    }  
}
```

```
    } else {  
        right = mid - 1;  
    }  
}  
return -1;  
}
```

4. Analysis:

- **Linear Search:**

- Time Complexity: $O(n)$ in the worst and average cases, $O(1)$ in the best case.
- Suitable for small or unsorted datasets where sorting overhead is not justified.

- **Binary Search:**

- Time Complexity: $O(\log n)$ in the worst and average cases, $O(1)$ in the best case.
- Requires a sorted array, making it ideal for large datasets where search performance is critical.

Discussion:

For an e-commerce platform with a large product catalog, binary search is more suitable due to its $O(\log n)$ time complexity, ensuring faster search operations. However, maintaining a sorted array adds overhead during insertions and updates. If the platform frequently updates the product list, a hybrid approach or advanced data structures like hash tables or B-trees might be considered. Linear search is simpler but inefficient for large datasets, making it suitable only for small or dynamic lists where sorting is impractical.

Final Choice:

Binary search is preferred for the e-commerce platform's search functionality due to its superior performance with large datasets, provided the product list remains sorted.

Exercise 7: Financial Forecasting

Step 1: Understand Recursive Algorithms

Recursion is a programming technique where a function calls itself to solve a problem by breaking it down into smaller, more manageable sub-problems. Each recursive call works on a smaller instance of the problem until it reaches a base case, which stops the recursion. Recursion simplifies problems that can be divided into similar sub-problems, such as tree traversals, factorial calculations, and certain mathematical series.

In financial forecasting, recursion can predict future values by applying growth rates to past data. For example, if you know the past values and growth rates, you can recursively compute future values by applying the growth rate to the previous value.

Step 2: Setup

To calculate future values recursively, we need:

1. A base case to stop the recursion.
2. A recursive case where the function calls itself with updated parameters.

The method will take:

- Current value.
- Growth rate.
- Number of future periods to predict.

Step 3: Implementation

The recursive implementation to predict future values:

```
def predict_future_value(current_value, growth_rate, periods):  
    if periods == 0:  
        return current_value  
    new_value = current_value * (1 + growth_rate / 100)  
    return predict_future_value(new_value, growth_rate, periods - 1)
```

Example usage:

```
initial_value = 1000
```

```
annual_growth_rate = 5
```

```
years = 10
```

```
future_value = predict_future_value(initial_value, annual_growth_rate, years)
```

```
print(f'Future value after {years} years: {future_value:.2f}')
```

Step 4: Analysis

1. Time Complexity:

- The recursive algorithm makes n calls for n periods, leading to a time complexity of $O(n)$, where n is the number of periods. Each call performs a constant amount of work (multiplication and subtraction), so the total work scales linearly with the number of periods.

2. Optimization:

- **Memoization:** If the same subproblems are solved repeatedly (e.g., in more complex recursive formulas), memoization can cache results to avoid redundant calculations. However, in this simple linear recursion, each step is unique, so memoization doesn't help.
- **Iterative Approach:** Recursion can be replaced with an iterative loop to avoid stack overflow for large n and reduce overhead from function calls. Here's the iterative version:

```
def predict_future_value_iterative(current_value, growth_rate, periods):  
    for _ in range(periods):  
        current_value *= (1 + growth_rate / 100)  
    return current_value
```

- **Tail Recursion:** Some languages optimize tail-recursive functions (where the recursive call is the last operation). Python does not optimize tail recursion, but in languages that do, this can prevent stack overflow. The provided recursive solution is tail-recursive.

3. Trade-offs:

- Recursion is elegant and mirrors the mathematical formulation but may risk stack overflow for very large n .
- Iteration is more efficient in terms of memory and is generally preferred for simple linear computations like this one.

