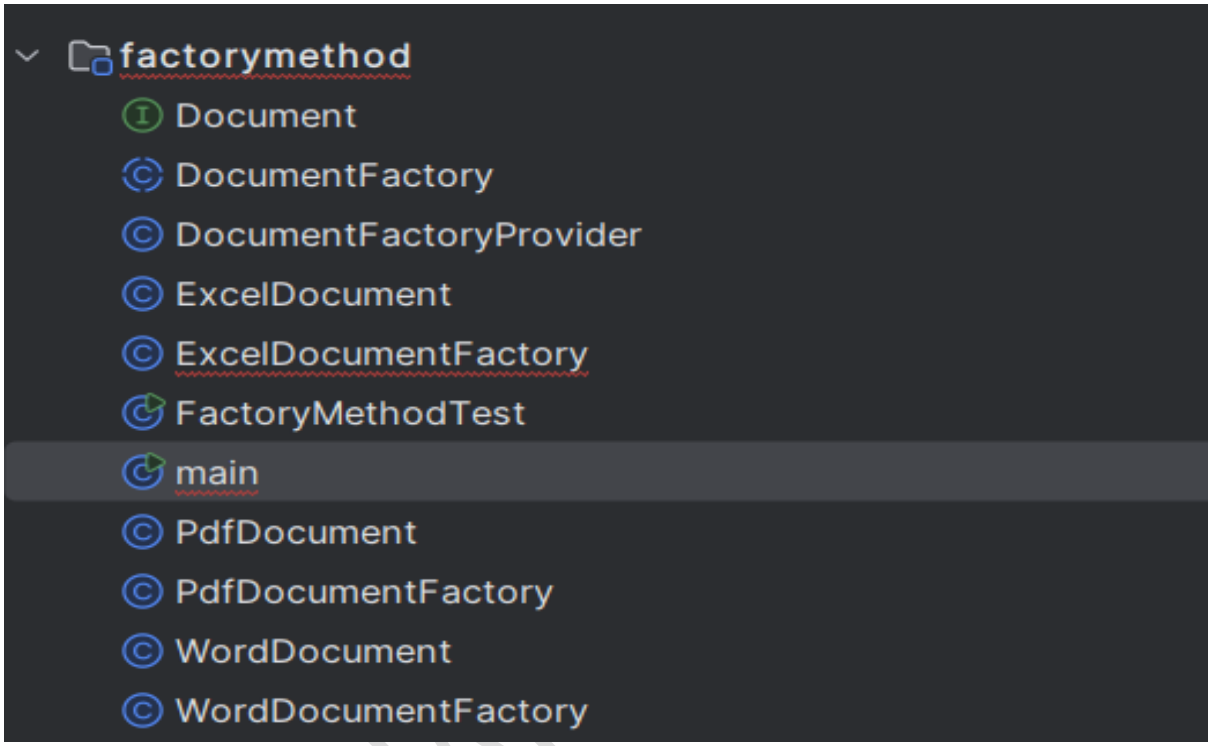
Implementing the Factory Method Pattern

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.

STRUCTURE OF CODE :



CODE :

Document.java

```
package factorymethod;

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

WordDocument.java

```
package factorymethod;

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

PdfDocument.java

```
package factorymethod;

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

ExcelDocument.java

```
package factorymethod;

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

DocumentFactory.java

```
package factorymethod;

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

WordDocumentFactory.java

```
package factorymethod;

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

PdfDocumentFactory.java

```
package factorymethod;

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

ExcelDocumentFactory.java

```
package factorymethod;

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

DocumentFactoryProvider.java

```
package factorymethod;

public class DocumentFactoryProvider {
    public static Document createDocument(String type) {
        switch (type.toLowerCase()) {
            case "word": return new WordDocument();
            case "pdf": return new PdfDocument();
            case "excel": return new ExcelDocument();
            default: throw new IllegalArgumentException("Unknown document
type: " + type);
        }
    }
}
```

FactoryMethodTest.java (Main class)

```
package factorymethod;

public class FactoryMethodTest {
    public static void main(String[] args) {

        // Using individual factories
        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();

        // Using centralized factory (optional)
        System.out.println("Using centralized factory:");
        Document doc = DocumentFactoryProvider.createDocument("pdf");
        doc.open();
    }
}
```

RESULTS:

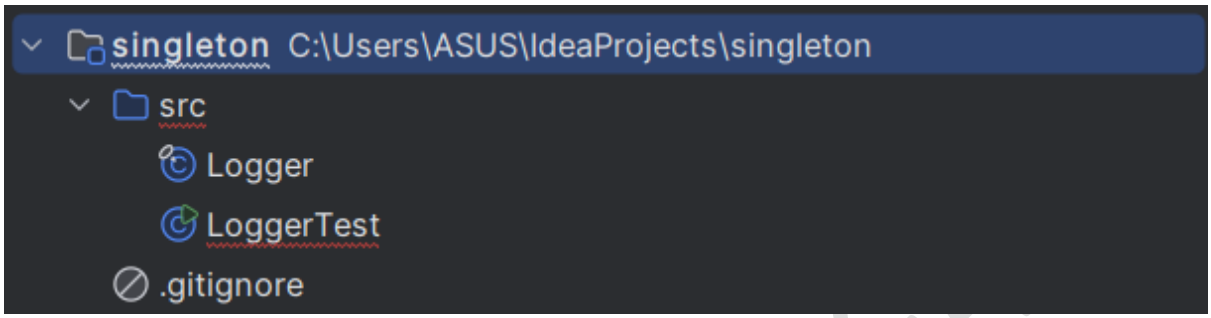
```
C:\Users\ASUS\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=50246" -Dfile.encoding=UTF-8 -D
Opening Word Document
Opening PDF Document
Opening Excel Document
Using centralized factory:
Opening PDF Document

Process finished with exit code 0
```

Implementing the Singleton Pattern

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

STRUCTURE OF CODE :



CODE :

LoggerTest

```
package singleton;

import java.util.stream.IntStream;

public class LoggerTest {
    public static void main(String[] args) {
        Logger a = Logger.getInstance();
        Logger b = Logger.getInstance();

        System.out.println("Same instance? " + (a == b)); // should print
true

        a.info("Starting application...");
        b.warn("This is a warning message.");
        Logger.getInstance().error("This is an error message.");

        IntStream.range(0, 5).parallel().forEach(i -> {
            Logger.getInstance().info("Parallel log " + i);
        });
    }
}
```

Logger

```
package singleton;

import java.io.Serializable;
import java.time.LocalDateTime;

public final class Logger implements Serializable {

    private Logger() {
        if (Holder.INSTANCE != null) {
            throw new IllegalStateException("Use getInstance()");
        }
    }
```

```java
    private static class Holder {
        private static final Logger INSTANCE = new Logger();
    }

    public static Logger getInstance() {
        return Holder.INSTANCE;
    }

    public void info(String msg) {
        log("INFO ", msg);
    }

    public void warn(String msg) {
        log("WARN ", msg);
    }

    public void error(String msg) {
        log("ERROR", msg);
    }

    private void log(String level, String msg) {
        System.out.printf("%s [%s] %s%n", LocalDateTime.now(), level, msg);
    }

    private static final long serialVersionUID = 1L;

    private Object readResolve() {
        return getInstance();
    }
}
```

RESULTS :

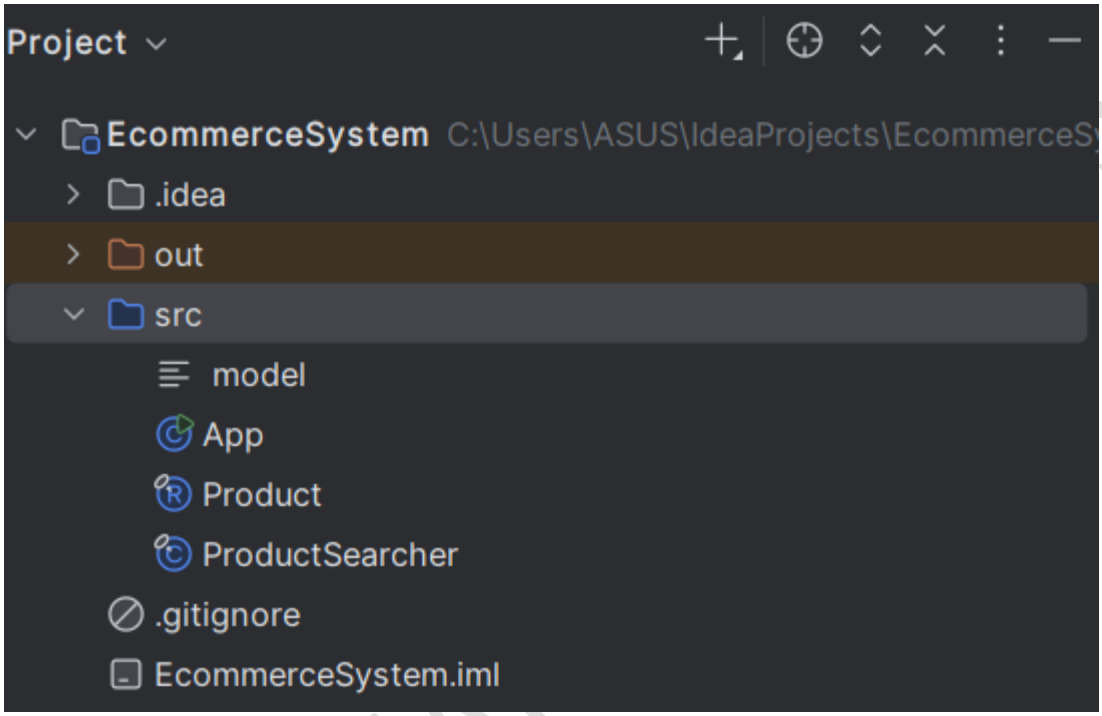
```
C:\Users\ASUS\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=50728" -Dfile.encoding=UTF-8 -D
Same instance? true
2025-06-22T00:17:27.010173900 [INFO ] Starting application...
2025-06-22T00:17:27.011334200 [WARN ] This is a warning message.
2025-06-22T00:17:27.011334200 [ERROR] This is an error message.
2025-06-22T00:17:27.018127600 [INFO ] Parallel log 2
2025-06-22T00:17:27.018127600 [INFO ] Parallel log 4
2025-06-22T00:17:27.018127600 [INFO ] Parallel log 3
2025-06-22T00:17:27.018127600 [INFO ] Parallel log 1
2025-06-22T00:17:27.018127600 [INFO ] Parallel log 0

Process finished with exit code 0
```

E-commerce Platform Search Function

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

STRUCTURE :



CODE :

ProductSearcher

```
package search;

import model.Product;
import java.util.Arrays;
import java.util.List;

public final class ProductSearcher {

    /** ---------- Linear Search (unsorted list or array) ---------- */
    public static int linearSearch(List<Product> products, long id) {
        for (int i = 0; i < products.size(); i++) {
            if (products.get(i).productId() == id) return i;    // hit
        }
        return -1;                                              // miss
    }

    /** ---------- Binary Search (requires array sorted by id) ---- */
    public static int binarySearch(Product[] sorted, long id) {
        int lo = 0, hi = sorted.length - 1;

        while (lo <= hi) {
            int mid = (lo + hi) >>> 1;              // no overflow
```

```
            long midId = sorted[mid].productId();

            if (midId < id)        lo = mid + 1;
            else if (midId > id)   hi = mid - 1;
            else                   return mid;       // hit
        }
        return -1;                                    // miss
    }

    /** Utility to get a sorted copy (by id) for binary search. */
    public static Product[] toSortedArray(List<Product> products) {
        Product[] arr = products.toArray(Product[]::new);
        Arrays.sort(arr);                             // uses compareTo
        return arr;
    }

    private ProductSearcher() {} // static-utility -- never instantiate
}
```

Product

```
package model;

/** Immutable domain object. */
public record Product(long productId, String productName, String category)
        implements Comparable<Product> {

    /** Natural order: by productId (ascending). */
    @Override public int compareTo(Product other) {
        return Long.compare(this.productId, other.productId);
    }
}
```

App.java

```
import model.Product;
import search.ProductSearcher;
import java.util.List;

public class App {
    public static void main(String[] args) {
        List<Product> catalog = List.of(
                new Product(101, "Laptop",         "Electronics"),
                new Product( 42, "Running Shoes", "Footwear"),
                new Product(555, "Desk Lamp",      "Home")
        );

        /* ---------- Linear search on ANY list ---------- */
        int idxLinear = ProductSearcher.linearSearch(catalog, 42);
        System.out.println("Linear index = " + idxLinear); // 1

        /* ---------- Binary search (needs sorted array) ---------- */
        var sorted = ProductSearcher.toSortedArray(catalog);
        int idxBinary = ProductSearcher.binarySearch(sorted, 42);
        System.out.println("Binary index = " + idxBinary); // 0 in sorted
view
    }
}
```

RESULT

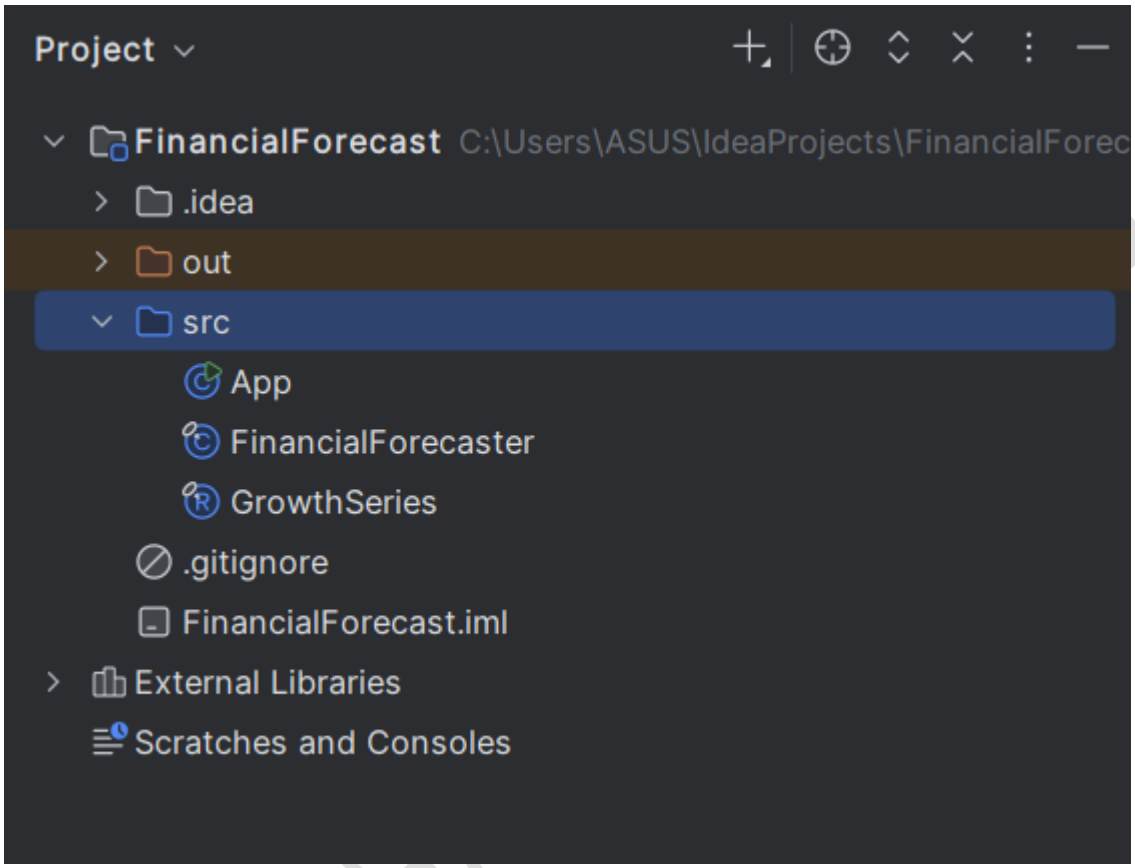
```
C:\Users\ASUS\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=50779" -Dfile.encoding=UTF-8 -
Linear index = 1
Binary index = 0

Process finished with exit code 0
```

Financial Forecasting

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

STRUCTURE :



CODE :

FinancialForecaster

```
package service;

import model.GrowthSeries;
import java.util.HashMap;
import java.util.Map;

public final class FinancialForecaster {

    // 1. Plain Recursive
    public static double futureValueRec(GrowthSeries gs, int n) {
        if (n == 0) return gs.initialValue();
        double prev = futureValueRec(gs, n - 1);
        return prev * (1.0 + gs.rateAt(n - 1));
    }

    // 2. Memoized Recursive (fixed to avoid
ConcurrentModificationException)
    public static double futureValueMemo(GrowthSeries gs, int n,
                                         Map<Integer, Double> cache) {
        if (cache.containsKey(n)) return cache.get(n);
```

```
        double value;
        if (n == 0) value = gs.initialValue();
        else value = futureValueMemo(gs, n - 1, cache) * (1 + gs.rateAt(n -
1));

        cache.put(n, value);
        return value;
    }

    public static double futureValueMemo(GrowthSeries gs, int n) {
        return futureValueMemo(gs, n, new HashMap<>());
    }

    // 3. Iterative
    public static double futureValueIter(GrowthSeries gs) {
        double fv = gs.initialValue();
        for (double r : gs.rates()) {
            fv *= (1 + r);
        }
        return fv;
    }

    private FinancialForecaster() {} // Utility class
}
```

GrowthSeries

```
package model;

import java.util.List;

/** Immutable wrapper for period-by-period growth rates (e.g., 0.05 = +5%).
*/
public record GrowthSeries(double initialValue, List<Double> rates) {

    public int periods() {
        return rates.size();
    }

    public double rateAt(int i) {
        return rates.get(i);
    }
}
```

App.java

```
import model.GrowthSeries;
import service.FinancialForecaster;
import java.util.List;

public class App {
    public static void main(String[] args) {
        // Sample data: initial value and growth rates over 4 periods
        GrowthSeries gs = new GrowthSeries(10_000, List.of(0.07, 0.05, -
0.02, 0.06));
        int n = gs.periods();

        double fvRec  = FinancialForecaster.futureValueRec(gs, n);
```

```java
        double fvMemo = FinancialForecaster.futureValueMemo(gs, n);
        double fvIter = FinancialForecaster.futureValueIter(gs);

        System.out.printf("Future Value (Recursion) : %.2f%n", fvRec);
        System.out.printf("Future Value (Memoized) : %.2f%n", fvMemo);
        System.out.printf("Future Value (Iterative): %.2f%n", fvIter);
    }
}
```

RESULTS :

```
C:\Users\ASUS\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.2\lib\idea_rt.jar=50836" -Dfile
Future Value (Recursion) : 11670.92
Future Value (Memoized) : 11670.92
Future Value (Iterative): 11670.92

Process finished with exit code 0
```