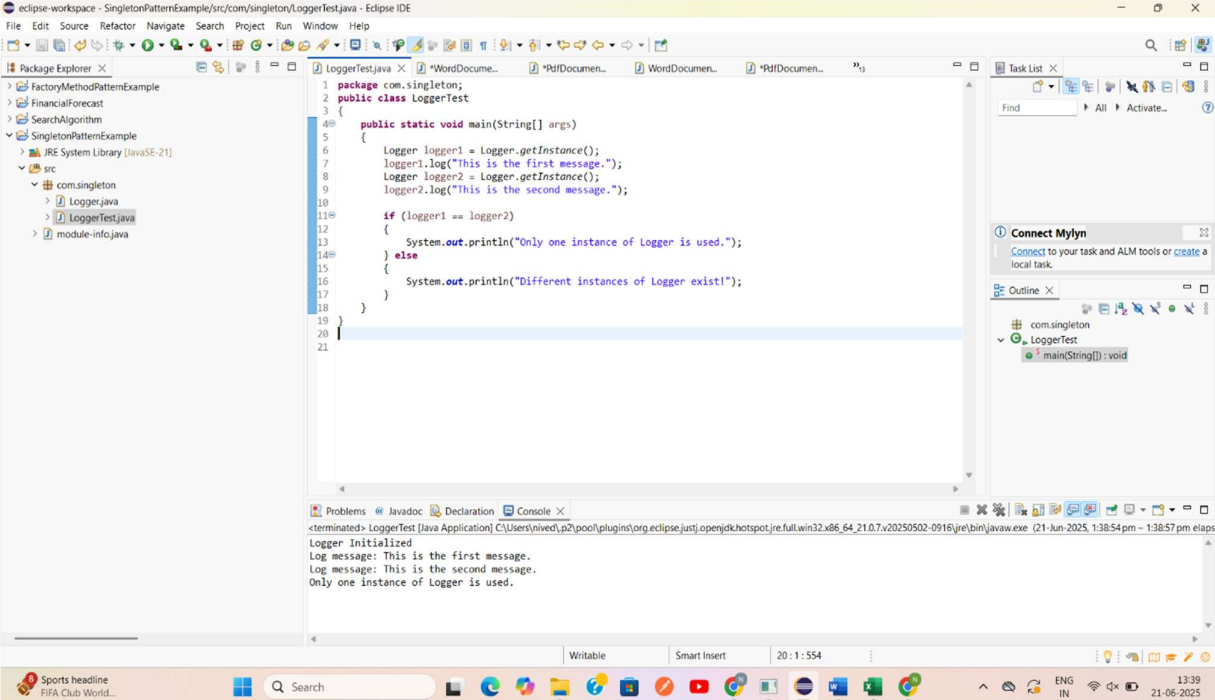


## WEAK 1:COGNIZANT HANDS-ON PROBLEMS

### TOPIC:1 DESIGN PRICIPLES AND PATTERNS

#### Exercise 1:Implement the singleton pattern

Output:



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows the project structure: SingletonPatternExample > src > com.singleton > LoggerTest.java. The main editor window shows the code for LoggerTest.java, which implements the Singleton pattern. The code includes a package declaration, a class declaration, and a main method that creates two logger instances and checks if they are the same. The console at the bottom shows the output of the program, confirming that only one instance of the Logger is used.

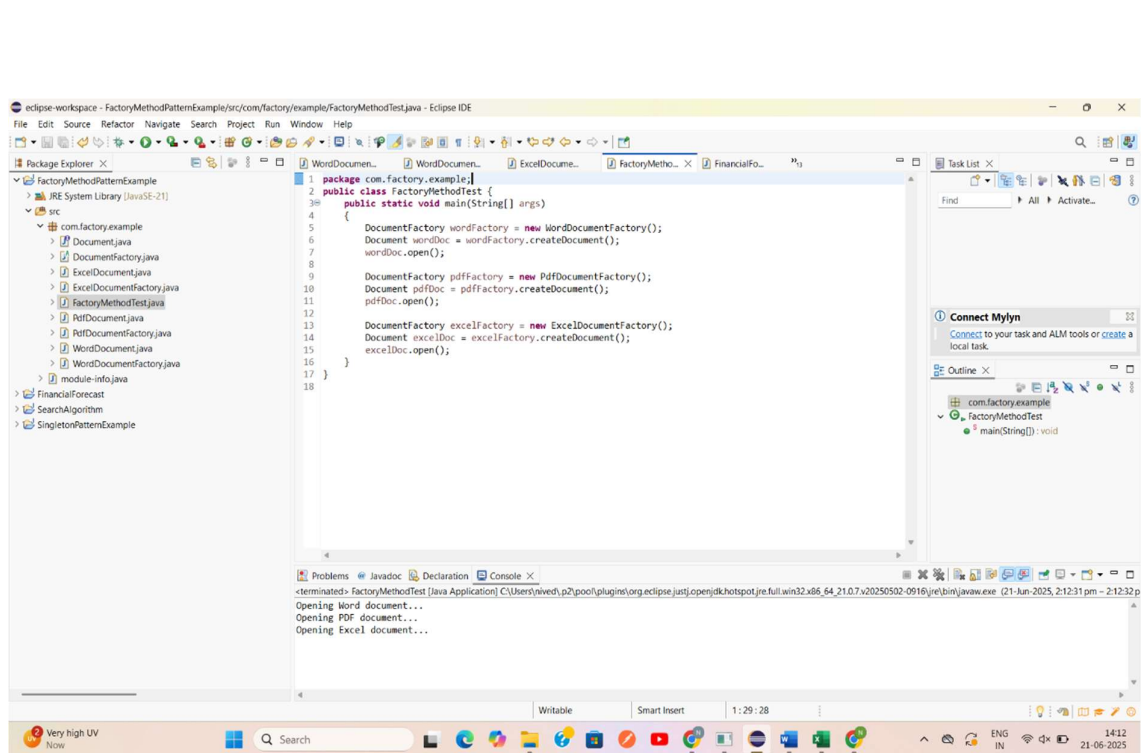
```
1 package com.singleton;
2 public class LoggerTest
3 {
4     public static void main(String[] args)
5     {
6         Logger logger1 = Logger.getInstance();
7         logger1.log("This is the first message.");
8         Logger logger2 = Logger.getInstance();
9         logger2.log("This is the second message.");
10
11         if (logger1 == logger2)
12         {
13             System.out.println("Only one instance of Logger is used.");
14         }
15         else
16         {
17             System.out.println("Different instances of Logger exist!");
18         }
19     }
20 }
21
```

Console Output:

```
<terminated> LoggerTest [Java Application] C:\Users\nivedi.p2\poo\plugins\org.eclipse.justi.openjdk hotspot\jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (21-Jun-2025, 1:38:54 pm - 1:38:57 pm elapsed)
Logger Initialized
Log message: This is the first message.
Log message: This is the second message.
Only one instance of Logger is used.
```

## Exercise 2: Implementing the Factory Method pattern

Output:



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows the project structure for 'FactoryMethodPatternExample', including the 'src' directory with files like 'Document.java', 'ExcelDocumentFactory.java', 'PdfDocumentFactory.java', 'WordDocumentFactory.java', 'WordDocument.java', and 'FactoryMethodTest.java'. The main editor window shows the code for 'FactoryMethodTest.java', which implements the Factory Method pattern by creating instances of 'WordDocument', 'PdfDocument', and 'ExcelDocument' through their respective factories. The console at the bottom shows the output of the program, indicating that the Word, PDF, and Excel documents are being opened successfully.

```
1 package com.factory.example;
2 public class FactoryMethodTest {
3     public static void main(String[] args)
4     {
5         DocumentFactory wordFactory = new WordDocumentFactory();
6         Document wordDoc = wordFactory.createDocument();
7         wordDoc.open();
8
9         DocumentFactory pdfFactory = new PdfDocumentFactory();
10        Document pdfDoc = pdfFactory.createDocument();
11        pdfDoc.open();
12
13        DocumentFactory excelFactory = new ExcelDocumentFactory();
14        Document excelDoc = excelFactory.createDocument();
15        excelDoc.open();
16    }
17 }
18
```

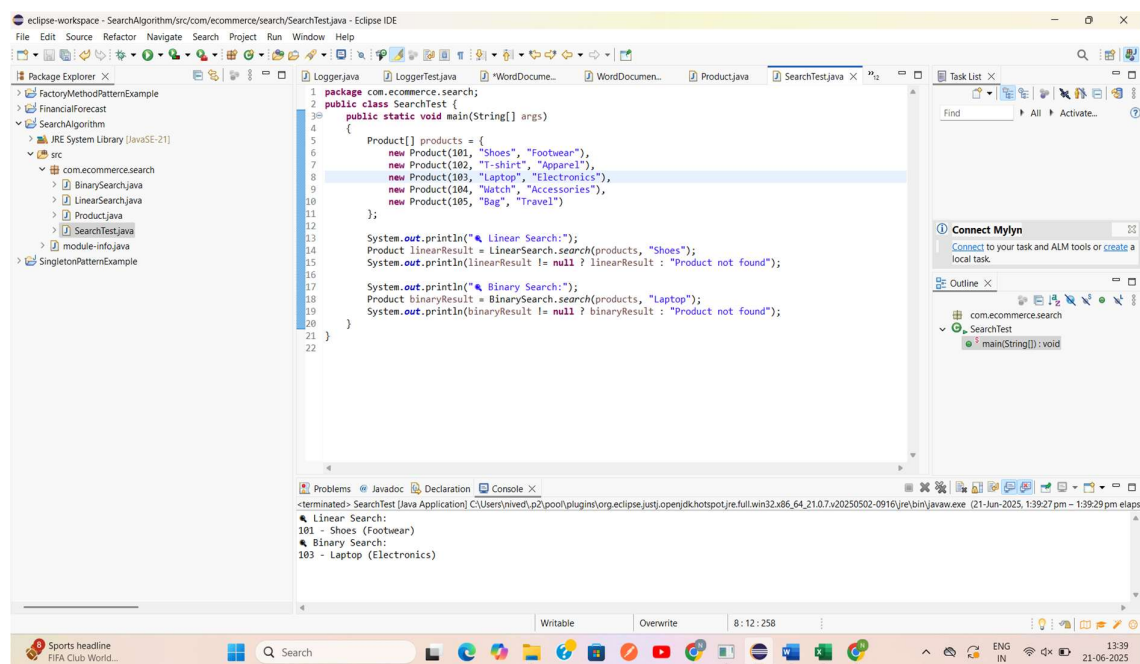
Console Output:

```
<terminated> FactoryMethodTest [Java Application] C:\Users\javed\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (21-Jun-2025, 2:12:31 pm - 2:12:32 p
Opening Word document...
Opening PDF document...
Opening Excel document...
```

## TOPIC 2:DATA STRUCTURE AND ALGORITHMS

### Exercise 2:E-commerce platform and search function

Output:



The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays the project structure, including the 'com.ecommerce.search' package and its sub-packages. The main editor window shows the 'SearchTest.java' file, which contains a public class 'SearchTest' with a 'main' method. The 'main' method initializes a list of products and performs linear and binary searches. The console at the bottom shows the output of the program, which includes the results of the linear and binary searches.

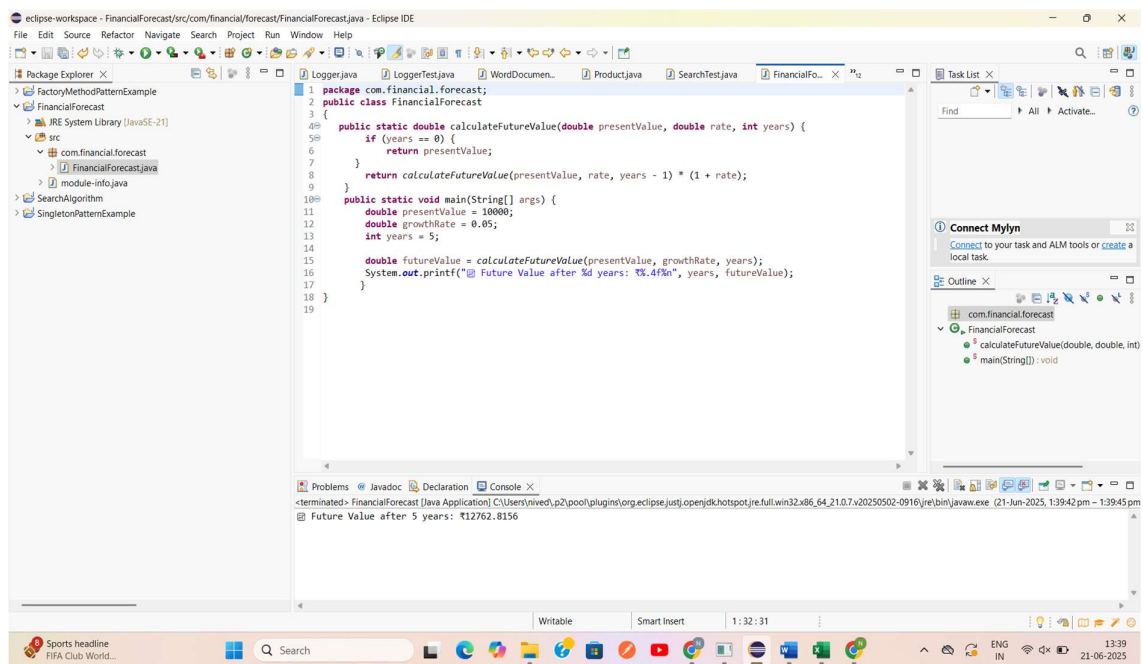
```
1 package com.ecommerce.search;
2 public class SearchTest {
3     public static void main(String[] args)
4     {
5         Product[] products = {
6             new Product(101, "Shoes", "Footwear"),
7             new Product(102, "T-shirt", "Apparel"),
8             new Product(103, "Laptop", "Electronics"),
9             new Product(104, "Watch", "Accessories"),
10            new Product(105, "Bag", "Travel")
11        };
12
13        System.out.println("Linear Search:");
14        Product linearResult = LinearSearch.search(products, "Shoes");
15        System.out.println(linearResult != null ? linearResult : "Product not found");
16
17        System.out.println("Binary Search:");
18        Product binaryResult = BinarySearch.search(products, "Laptop");
19        System.out.println(binaryResult != null ? binaryResult : "Product not found");
20    }
21 }
22 }
```

Console Output:

```
<terminated> SearchTest (Java Application) C:\Users\nivedh.p2\poo\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (21-Jun-2025, 1:39:27 pm - 1:39:29 pm elapsed)
Linear Search:
101 - Shoes (Footwear)
Binary Search:
103 - Laptop (Electronics)
```

## Exercise 7: Financial Forecasting

Output:



The screenshot shows the Eclipse IDE interface. The main editor displays the `FinancialForecast.java` file with the following code:

```
1 package com.financial.forecast;
2 public class FinancialForecast
3 {
4     public static double calculateFutureValue(double presentValue, double rate, int years) {
5         if (years == 0) {
6             return presentValue;
7         }
8         return calculateFutureValue(presentValue, rate, years - 1) * (1 + rate);
9     }
10    public static void main(String[] args) {
11        double presentValue = 10000;
12        double growthRate = 0.05;
13        int years = 5;
14
15        double futureValue = calculateFutureValue(presentValue, growthRate, years);
16        System.out.printf("Future Value after %d years: %.4f\n", years, futureValue);
17    }
18 }
19
```

The Package Explorer on the left shows the project structure, including the `com.financial.forecast` package. The Task List on the right shows the `main` method. The Console at the bottom displays the output of the program:

```
<terminated> FinancialForecast [Java Application] C:\Users\nived\p2\pools\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.21.0.7.v20250502-0916\jre\bin\javaw.exe (21-Jun-2025, 1:39:42 pm - 1:39:45 pm)
Future Value after 5 years: ₹12762.8156
```