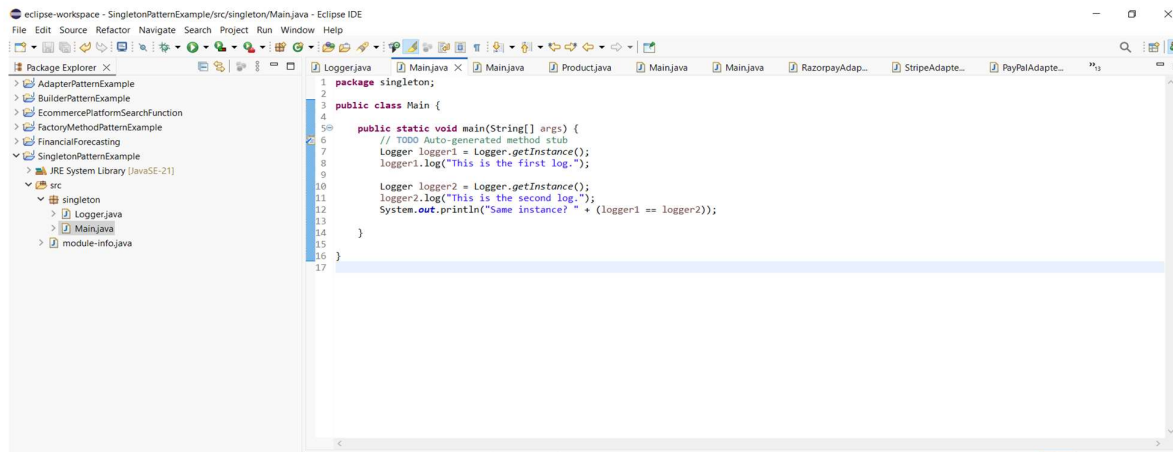
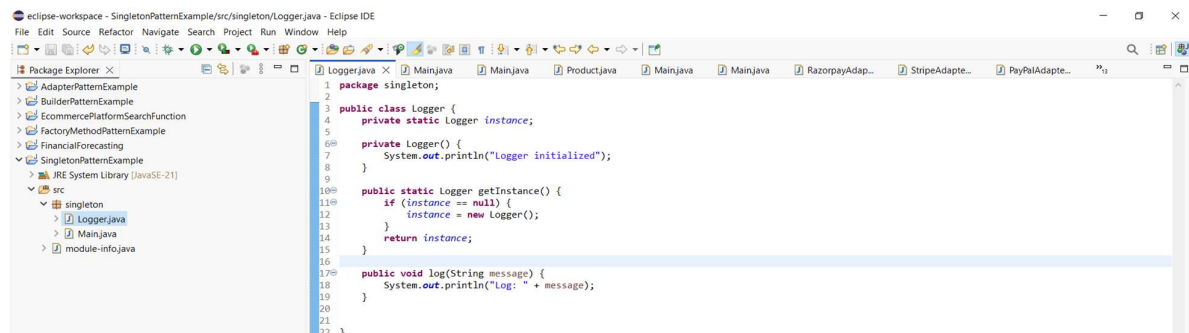


DESIGN PATTERNS AND PRINCIPLES

1) Implementing the Singleton Pattern

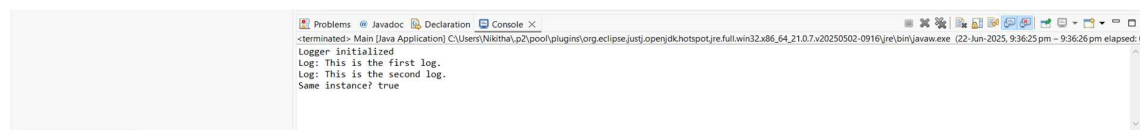


```
1 package singleton;
2
3 public class Main {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Logger logger1 = Logger.getInstance();
8         logger1.log("This is the first log.");
9
10        Logger logger2 = Logger.getInstance();
11        logger2.log("This is the second log.");
12        System.out.println("Same instance? " + (logger1 == logger2));
13    }
14 }
15
16
17 }
```



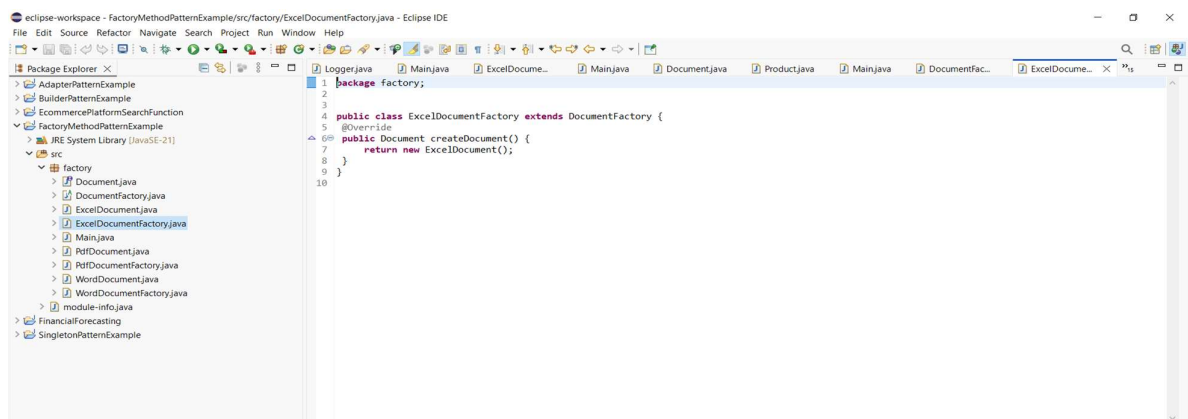
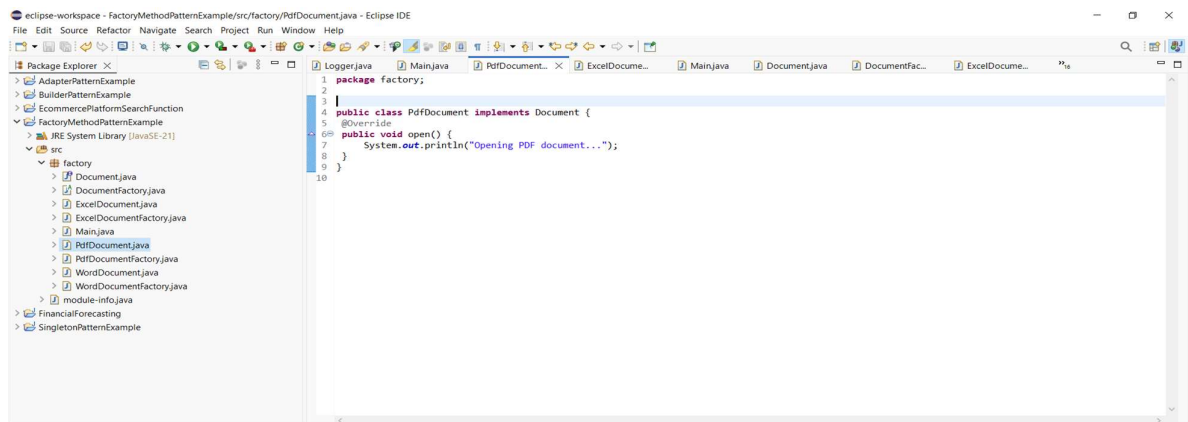
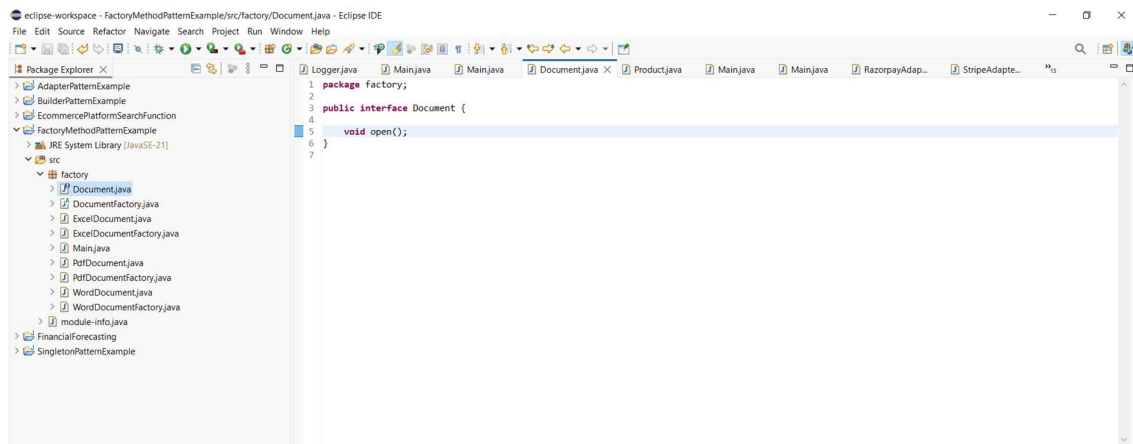
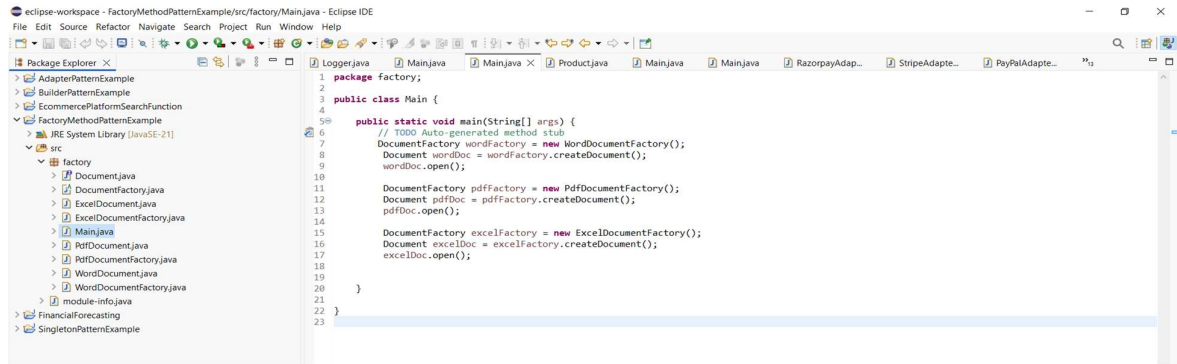
```
1 package singleton;
2
3 public class Logger {
4     private static Logger instance;
5
6     private Logger() {
7         System.out.println("Logger initialized");
8     }
9
10    public static Logger getInstance() {
11        if (instance == null) {
12            instance = new Logger();
13        }
14        return instance;
15    }
16
17    public void log(String message) {
18        System.out.println("Log: " + message);
19    }
20 }
21
22 }
```

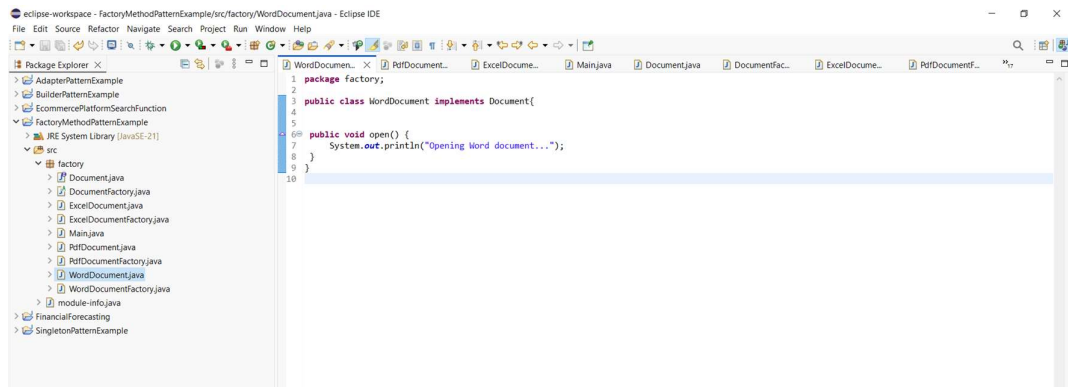
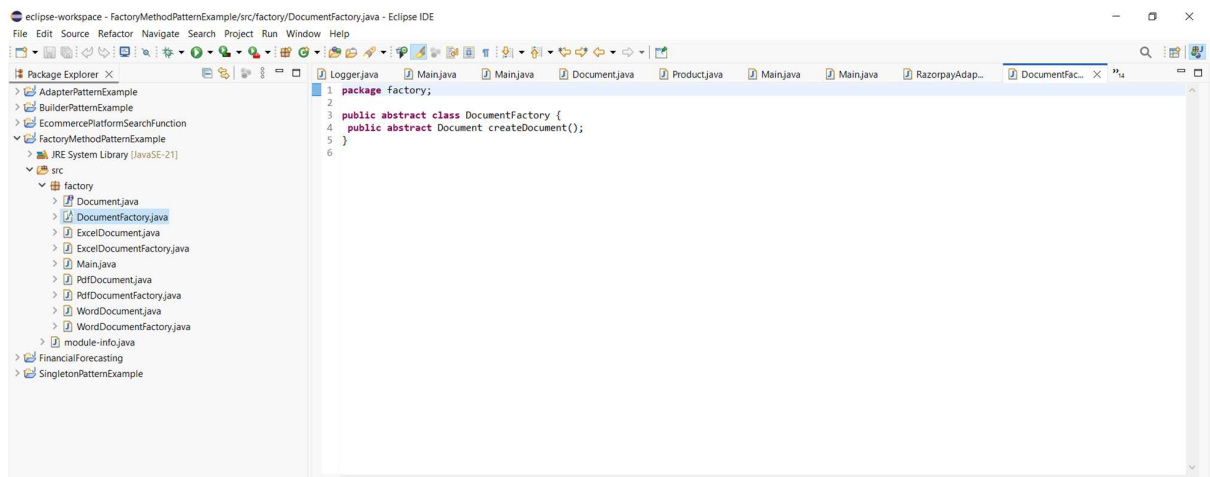
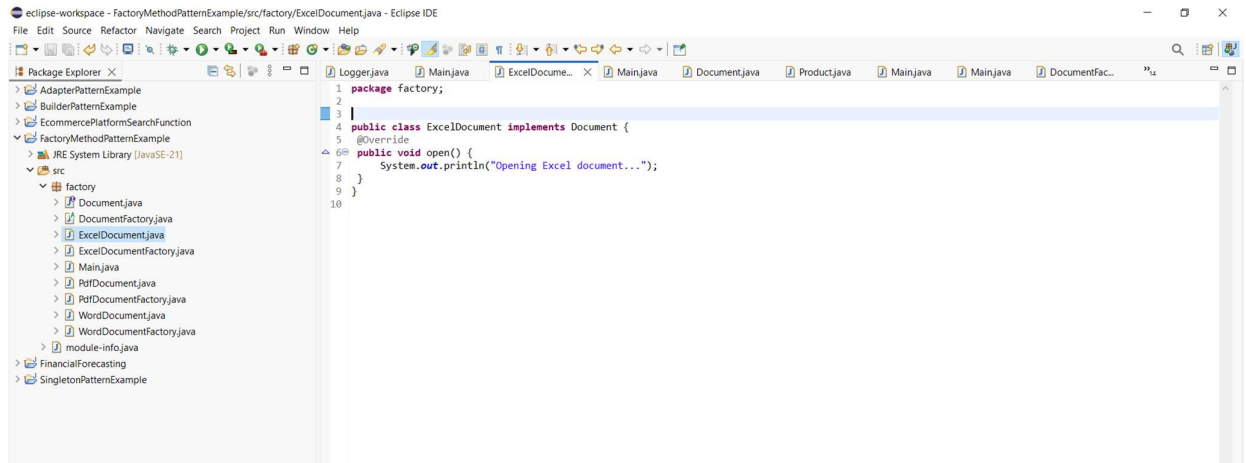
OUTPUT:

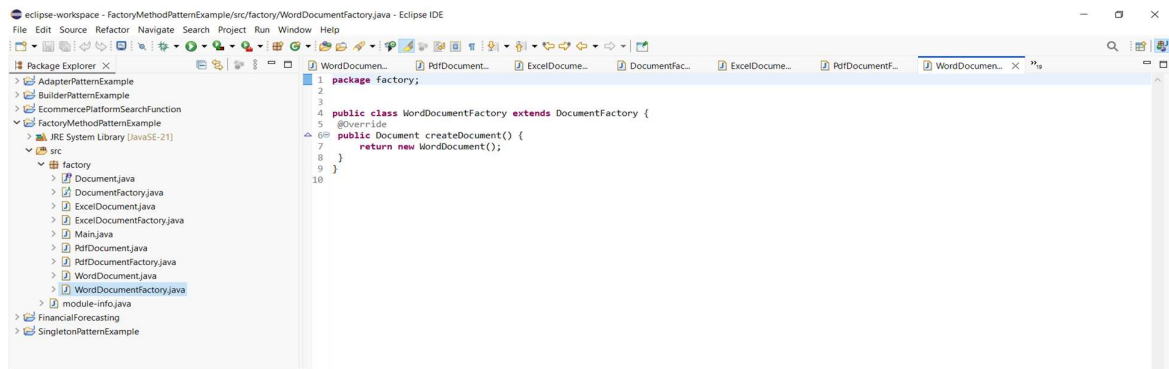


```
<terminated> Main [Java Application] C:\Users\Nikhitha\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.21.0.7.v20250502-0916\jre\bin\javaw.exe (22-Jun-2025, 9:36:25 pm - 9:36:26 pm elapsed: C
Logger initialized
Log: This is the first log.
Log: This is the second log.
Same instance? true
```

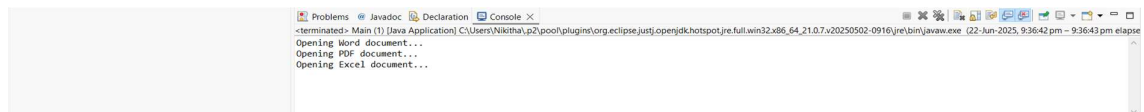
2) Implementing the Factory Method Pattern



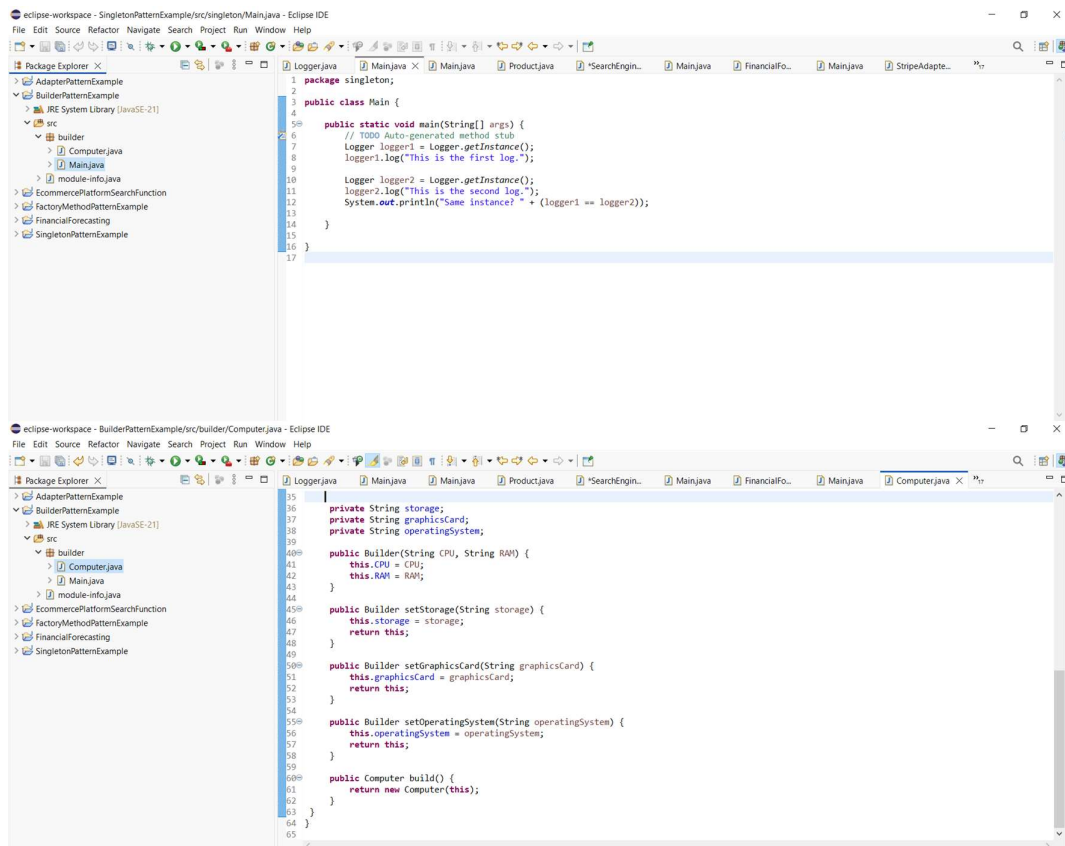




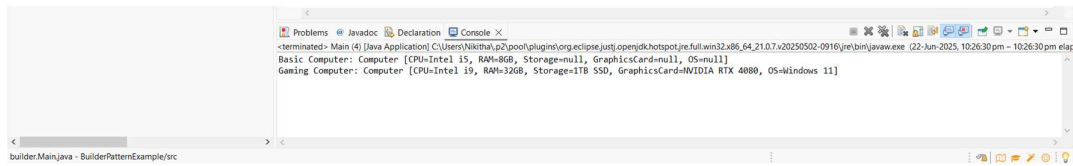
OUTPUT :



3) Implementing the Builder Pattern

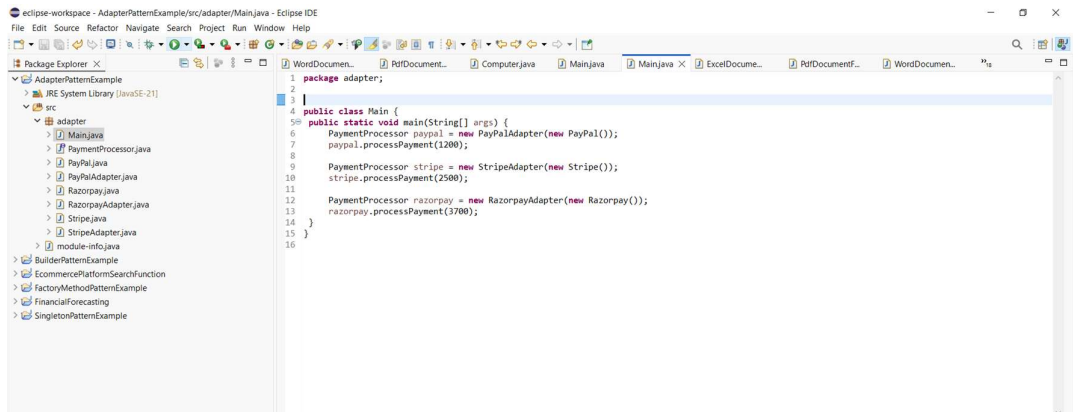


OUTPUT:



```
<terminated> Main (4) [Java Application] C:\Users\Nikhil.p\Zoo\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0910\jre\bin\java.exe (22-Jun-2025, 10:26:30 pm - 10:26:30 pm elapsed)
Basic Computer: Computer [CPU=Intel i5, RAM=8GB, Storage=null, GraphicsCard=null, OS=null]
Gaming Computer: Computer [CPU=Intel i9, RAM=32GB, Storage=1TB SSD, GraphicsCard=NVIDIA RTX 4080, OS=Windows 11]
```

4) Implementing the Adapter Pattern

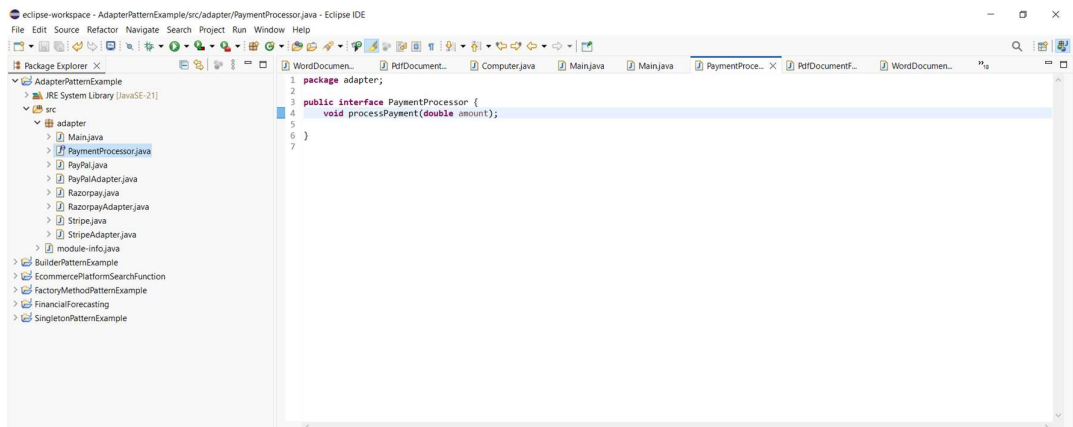


```
package adapter;

public class Main {
    public static void main(String[] args) {
        PaymentProcessor paypal = new PayPalAdapter(new PayPal());
        paypal.processPayment(1200);

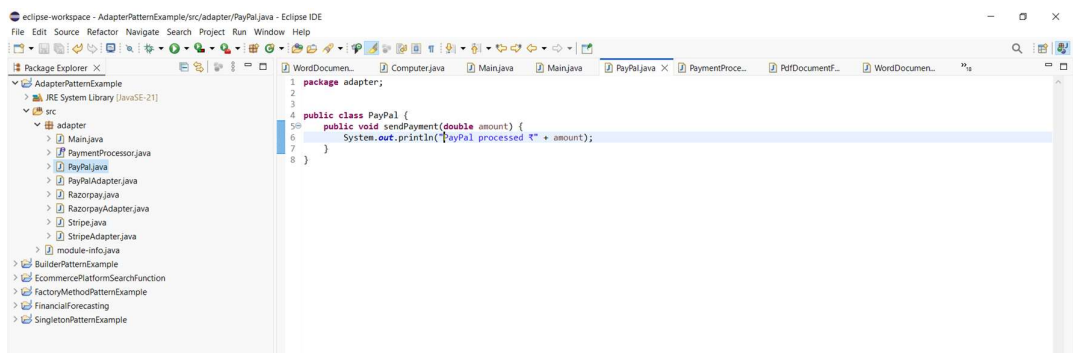
        PaymentProcessor stripe = new StripeAdapter(new Stripe());
        stripe.processPayment(2500);

        PaymentProcessor razorpay = new RazorpayAdapter(new Razorpay());
        razorpay.processPayment(3700);
    }
}
```



```
package adapter;

public interface PaymentProcessor {
    void processPayment(double amount);
}
```



```
package adapter;

public class PayPal {
    public void sendPayment(double amount) {
        System.out.println("PayPal processed ₹" + amount);
    }
}
```

eclipse-workspace - AdapterPatternExample/src/adapter/PayPalAdapter.java - Eclipse IDE

```
1 package adapter;
2
3
4 public class PayPalAdapter implements PaymentProcessor {
5     private PayPal paypal;
6
7     public PayPalAdapter(PayPal paypal) {
8         this.paypal = paypal;
9     }
10
11     @Override
12     public void processPayment(double amount) {
13         paypal.sendPayment(amount);
14     }
15 }
16
```

eclipse-workspace - AdapterPatternExample/src/adapter/Stripe.java - Eclipse IDE

```
1 package adapter;
2
3
4 public class Stripe {
5     public void makeStripePayment(double amount) {
6         System.out.println("Stripe charged $" + amount);
7     }
8 }
9
```

eclipse-workspace - AdapterPatternExample/src/adapter/StripeAdapter.java - Eclipse IDE

```
1 package adapter;
2
3 public class StripeAdapter implements PaymentProcessor {
4     private Stripe stripe;
5
6     public StripeAdapter(Stripe stripe) {
7         this.stripe = stripe;
8     }
9
10    @Override
11    public void processPayment(double amount) {
12        stripe.makeStripePayment(amount);
13    }
14 }
15
```

```

1 package adapter;
2
3
4 public class RazorpayAdapter implements PaymentProcessor {
5     private Razorpay razorpay;
6
7     public RazorpayAdapter(Razorpay razorpay) {
8         this.razorpay = razorpay;
9     }
10
11     @Override
12     public void processPayment(double amount) {
13         razorpay.completeTransaction(amount);
14     }
15 }
16

```

```

1 package adapter;
2
3
4 public class Razorpay {
5
6     public void completeTransaction(double amount) {
7         System.out.println("Razorpay transaction done for ₹" + amount);
8     }
9 }
10

```

OUTPUT:

```

<terminated> Main (5) [Java Application] C:\Users\Nikhil\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.21.0.7.v20250502-0916\jre\bin\javaw.exe (22-Jun-2025, 9:37:49 pm - 9:37:49 pm elapsed)
PayPal processed ₹1200.0
Stripe charged ₹2500.0
Razorpay transaction done for ₹3700.0

```

ALGORITHMS AND DATA STRUCTURES

Exercise 2: E-commerce Platform Search Function

Big O notation is used to describe the performance of an algorithm as the input size increases. It helps in understanding how the algorithm scales.

- Best Case: The minimum time required (e.g., item is at the beginning).
- Average Case: Expected time for typical inputs.
- Worst Case: Maximum time required (e.g., item not found or at the end).


```

1 package ecommerce;
2
3
4 public class Main {
5     public static void main(String[] args) {
6         Product[] products = {
7             new Product(101, "iPhone", "Electronics"),
8             new Product(102, "Shoes", "Fashion"),
9             new Product(103, "Watch", "Accessories"),
10            new Product(104, "Laptop", "Electronics"),
11            new Product(105, "Bag", "Fashion")
12        };
13
14        System.out.println("Linear Search:");
15        Product foundLinear = SearchEngine.linearSearch(products, "Watch");
16        System.out.println(foundLinear != null ? foundLinear : "Product not found");
17
18        System.out.println("\nBinary Search:");
19        Product foundBinary = SearchEngine.binarySearch(products, "Laptop");
20        System.out.println(foundBinary != null ? foundBinary : "Product not found");
21    }
22 }
23
24

```

```

1 package ecommerce;
2
3
4 public class Product {
5     int productId;
6     String productName;
7     String category;
8
9     public Product(int productId, String productName, String category) {
10        this.productId = productId;
11        this.productName = productName;
12        this.category = category;
13    }
14
15    @Override
16    public String toString() {
17        return productId + " - " + productName + " (" + category + ")";
18    }
19 }
20

```

```

1 package ecommerce;
2
3 import java.util.Arrays;
4 import java.util.Comparator;
5
6 public class SearchEngine {
7
8     public static Product linearSearch(Product[] products, String targetName) {
9         for (Product product : products) {
10            if (product.productName.equalsIgnoreCase(targetName)) {
11                return product;
12            }
13        }
14        return null;
15    }
16
17    // Binary search by product name
18    public static Product binarySearch(Product[] products, String targetName) {
19        Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));
20
21        int left = 0, right = products.length - 1;
22        while (left <= right) {
23            int mid = (left + right) / 2;
24            int comparison = products[mid].productName.compareToIgnoreCase(targetName);
25
26            if (comparison == 0) return products[mid];
27            else if (comparison < 0) left = mid + 1;
28            else right = mid - 1;
29        }
30        return null;
31    }
32 }

```

OUTPUT:

```

Linear Search:
103 - Watch (Accessories)

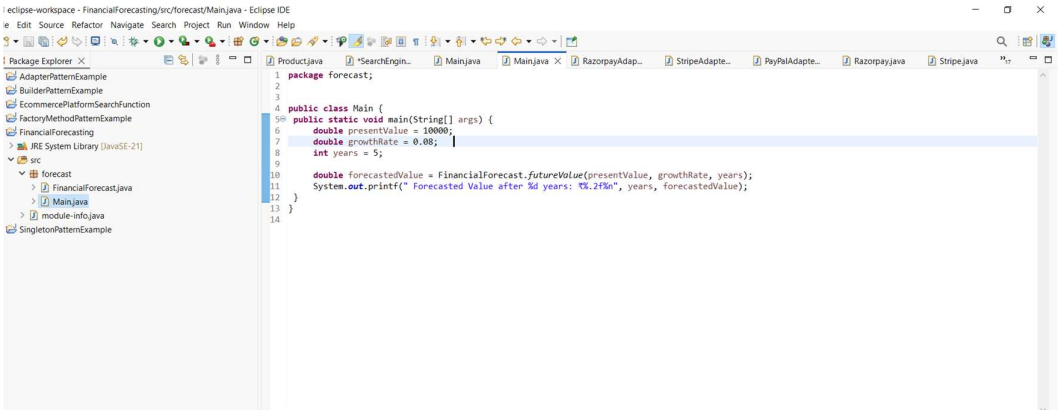
Binary Search:
104 - Laptop (Electronics)

```

Exercise 7: Financial Forecasting

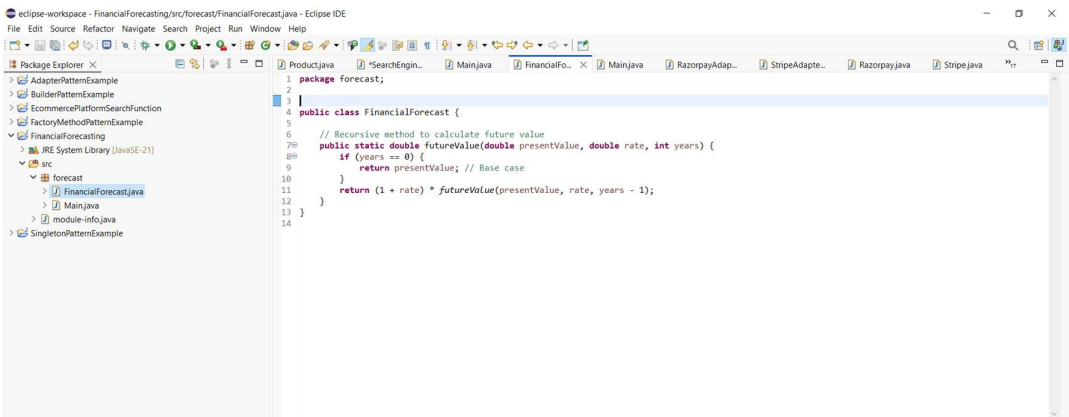
Recursion is a technique where a function calls itself to solve a problem. It is particularly useful in scenarios where a problem can be divided into subproblems of the same type.

Example: Predicting future value based on compound growth — where each year's value depends on the previous year's.



The screenshot shows the Eclipse IDE with the Package Explorer on the left and the Main.java file open in the editor. The Package Explorer shows a project named 'FinancialForecast' with a package 'forecast' containing 'FinancialForecast.java' and 'Main.java'. The Main.java file contains the following code:

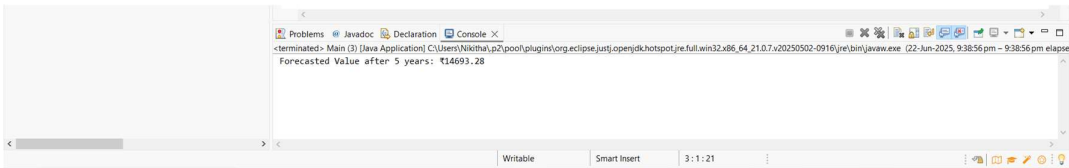
```
1 package forecast;
2
3
4 public class Main {
5     public static void main(String[] args) {
6         double presentValue = 10000;
7         double growthRate = 0.08;
8         int years = 5;
9
10        double forecastedValue = FinancialForecast.futureValue(presentValue, growthRate, years);
11        System.out.printf(" Forecasted Value after %d years: ₹%.2f\n", years, forecastedValue);
12    }
13 }
14
```



The screenshot shows the Eclipse IDE with the Package Explorer on the left and the FinancialForecast.java file open in the editor. The FinancialForecast.java file contains the following code:

```
1 package forecast;
2
3
4 public class FinancialForecast {
5     // Recursive method to calculate future value
6
7     public static double futureValue(double presentValue, double rate, int years) {
8         if (years == 0) {
9             return presentValue; // Base case
10        }
11        return (1 + rate) * futureValue(presentValue, rate, years - 1);
12    }
13 }
14
```

OUTPUT:



The screenshot shows the Eclipse IDE with the Console window open at the bottom. The console output displays the result of the program execution:

```
<terminated> Main (3) [Java Application] C:\Users\Nikhil.pz\poo\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64.21.0.7.v20230502-0916\jre\bin\java.exe (22-Jun-2025, 9:38:56 pm - 9:38:56 pm elapsed)
Forecasted Value after 5 years: ₹14693.28
```