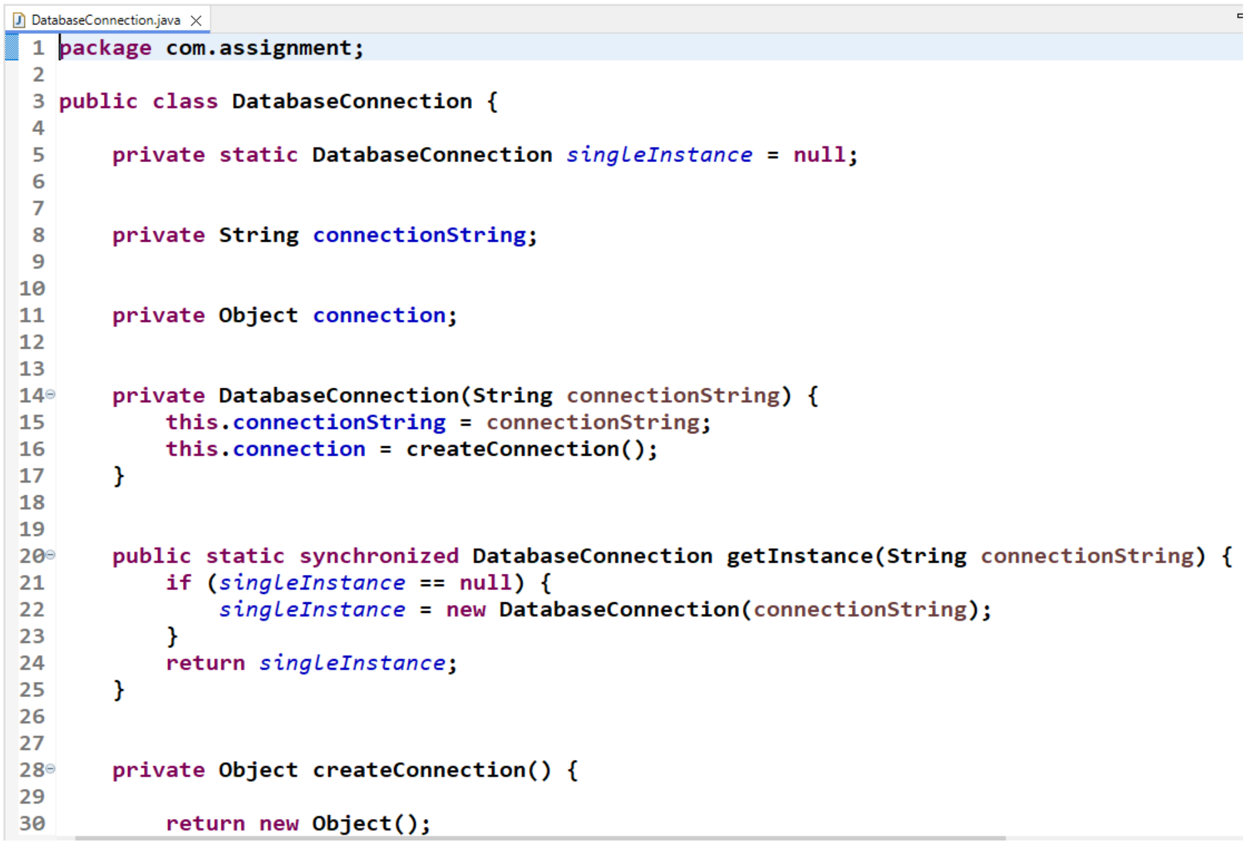
Day - 23

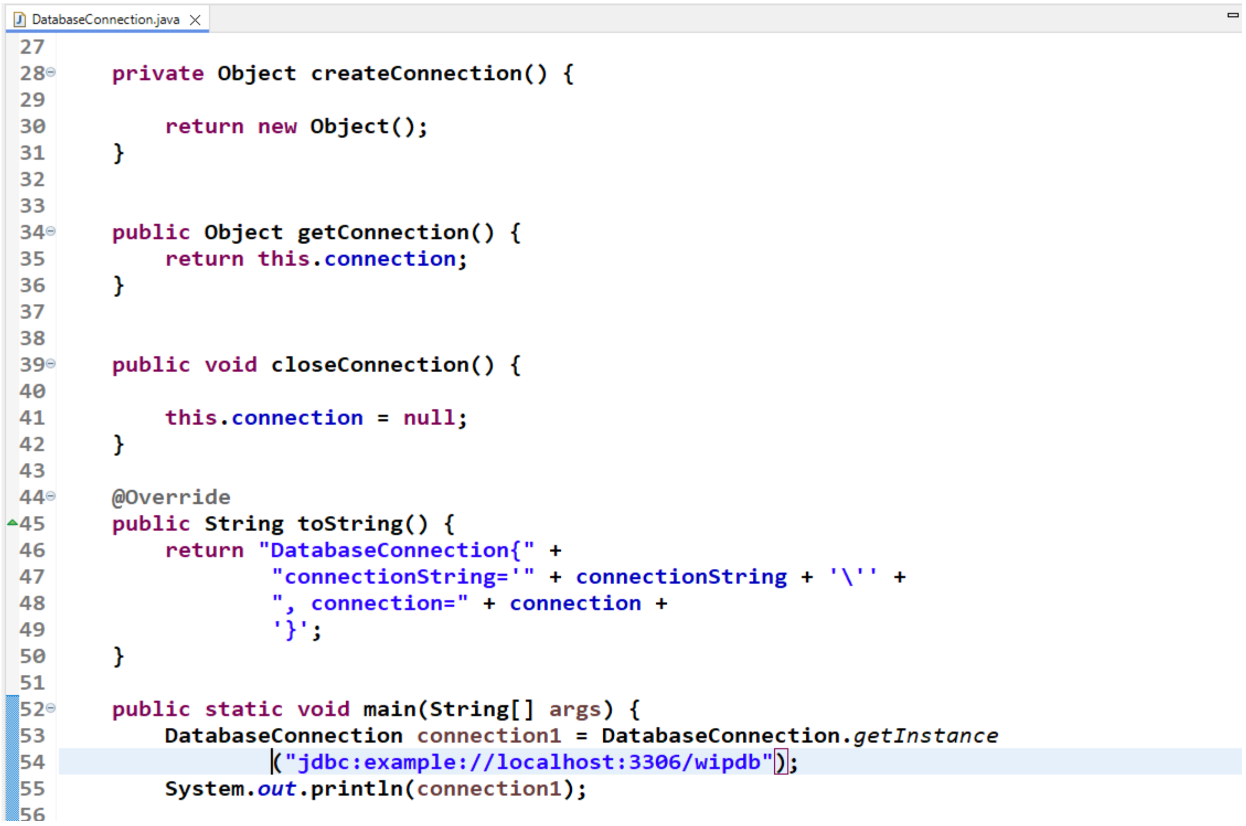
1] Task 1: Singleton
Implement a Singleton class that manages database connections. Ensure the class adheres strictly to the singleton pattern principles.

Solution :-
Code :-

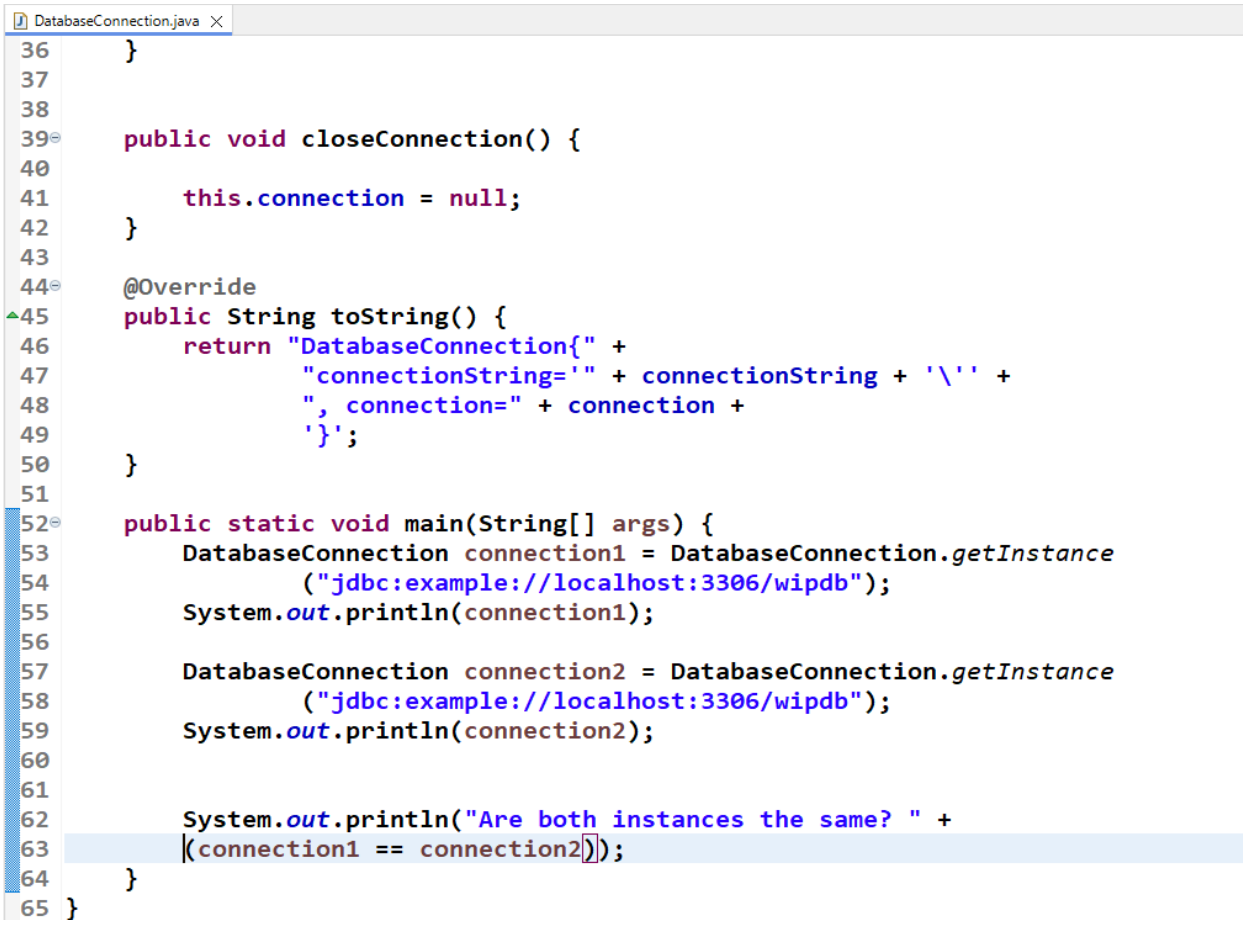


DatabaseConnection.java

```java
package com.assignment;

public class DatabaseConnection {

    private static DatabaseConnection singleInstance = null;


    private String connectionString;


    private Object connection;


    private DatabaseConnection(String connectionString) {
        this.connectionString = connectionString;
        this.connection = createConnection();
    }


    public static synchronized DatabaseConnection getInstance(String connectionString) {
        if (singleInstance == null) {
            singleInstance = new DatabaseConnection(connectionString);
        }
        return singleInstance;
    }


    private Object createConnection() {

        return new Object();
```

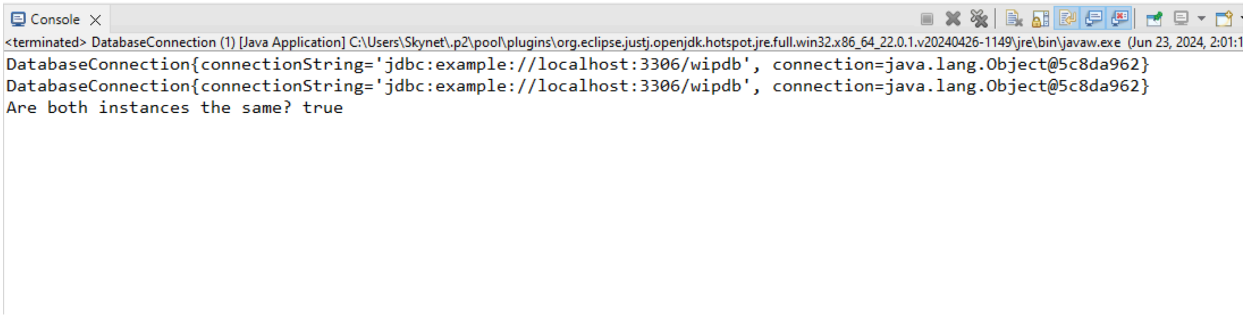

DatabaseConnection.java

```java

    private Object createConnection() {

        return new Object();
    }


    public Object getConnection() {
        return this.connection;
    }


    public void closeConnection() {

        this.connection = null;
    }

    @Override
    public String toString() {
        return "DatabaseConnection{" +
                "connectionString='" + connectionString + '\'' +
                ", connection=" + connection +
                '}';
    }

    public static void main(String[] args) {
        DatabaseConnection connection1 = DatabaseConnection.getInstance
                ("jdbc:example://localhost:3306/wipdb");
        System.out.println(connection1);

```



```java
    }


    public void closeConnection() {

        this.connection = null;
    }

    @Override
    public String toString() {
        return "DatabaseConnection{" +
                "connectionString='" + connectionString + '\'' +
                ", connection=" + connection +
                '}';
    }

    public static void main(String[] args) {
        DatabaseConnection connection1 = DatabaseConnection.getInstance
                ("jdbc:example://localhost:3306/wipdb");
        System.out.println(connection1);

        DatabaseConnection connection2 = DatabaseConnection.getInstance
                ("jdbc:example://localhost:3306/wipdb");
        System.out.println(connection2);


        System.out.println("Are both instances the same? " +
        (connection1 == connection2));
    }
}
```

Output :-



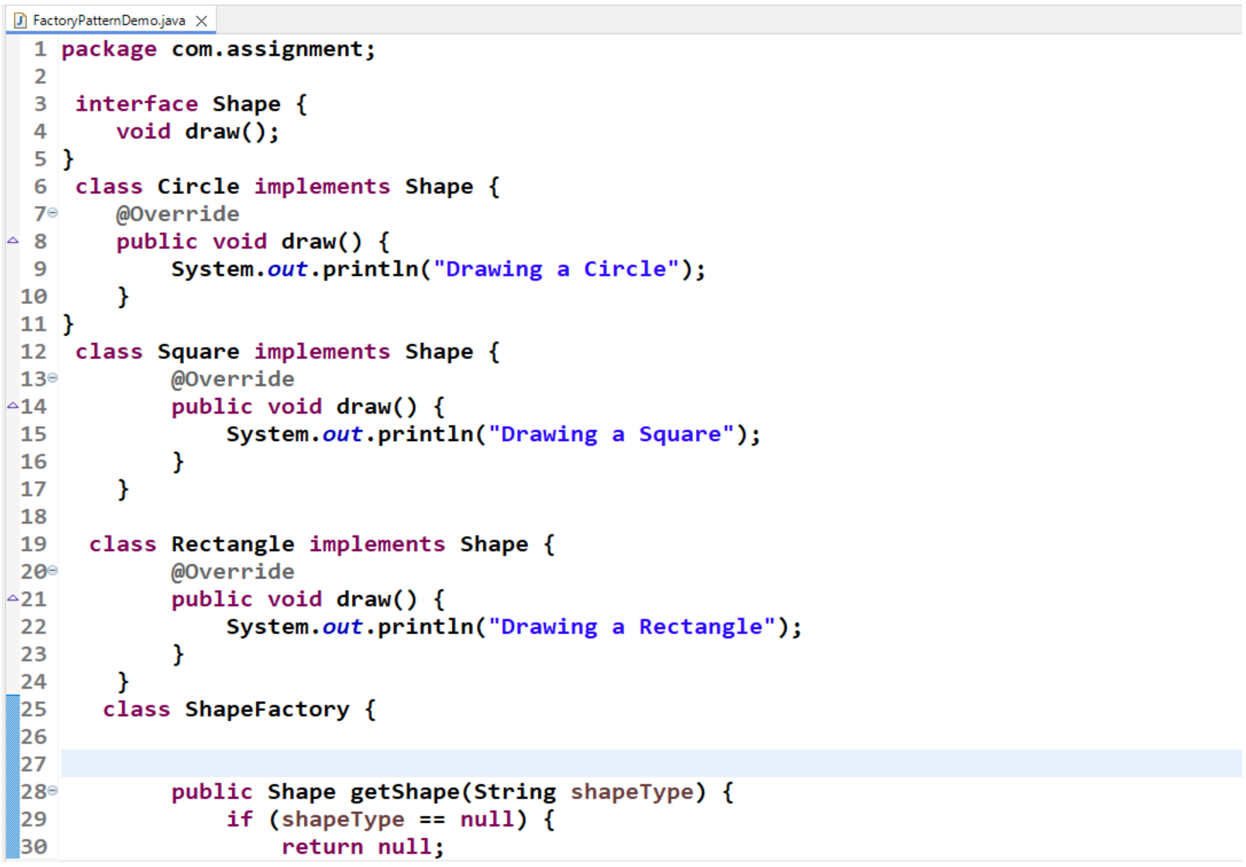
```
<terminated> DatabaseConnection (1) [Java Application] C:\Users\Skynet\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.1.v20240426-1149\jre\bin\javaw.exe (Jun 23, 2024, 2:01:1
DatabaseConnection{connectionString='jdbc:example://localhost:3306/wipdb', connection=java.lang.Object@5c8da962}
DatabaseConnection{connectionString='jdbc:example://localhost:3306/wipdb', connection=java.lang.Object@5c8da962}
Are both instances the same? true
```

2] Task 2: Factory Method

Create a ShapeFactory class that encapsulates the object creation logic of different Shape objects like Circle, Square, and Rectangle.

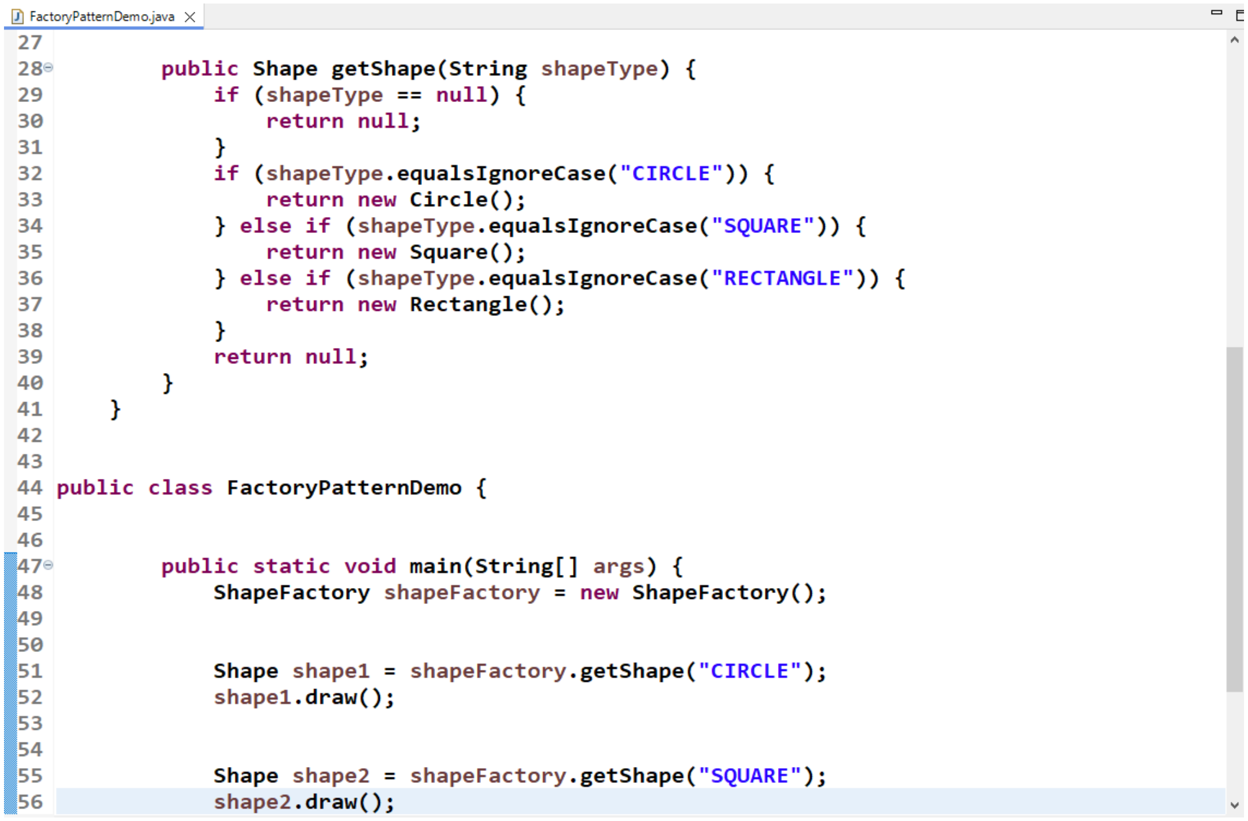
Solution :-

Code :-

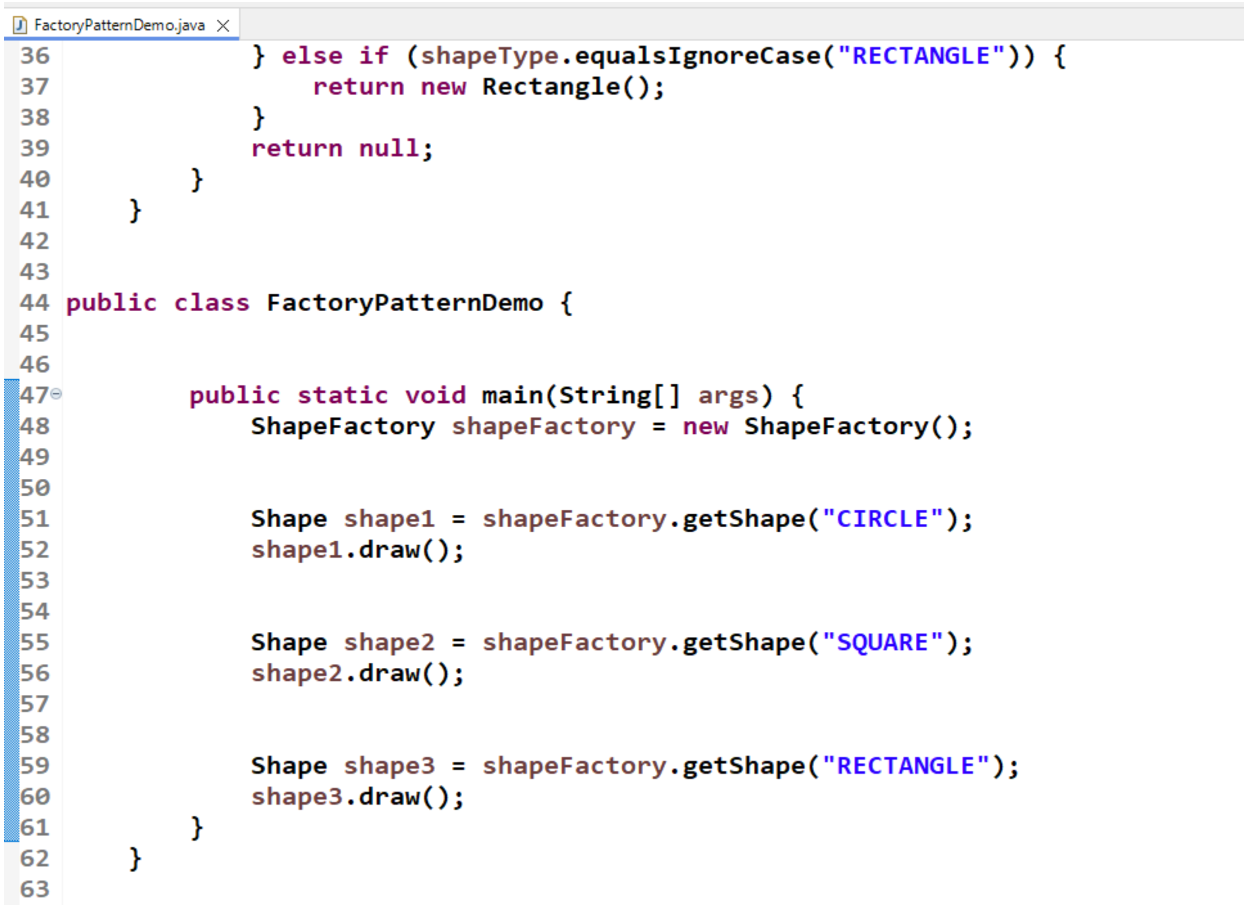


FactoryPatternDemo.java

```java
package com.assignment;

 interface Shape {
    void draw();
}
 class Circle implements Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a Circle");
    }
}
 class Square implements Shape {
        @Override
        public void draw() {
            System.out.println("Drawing a Square");
        }
    }

  class Rectangle implements Shape {
        @Override
        public void draw() {
            System.out.println("Drawing a Rectangle");
        }
    }
   class ShapeFactory {


        public Shape getShape(String shapeType) {
            if (shapeType == null) {
                return null;
```

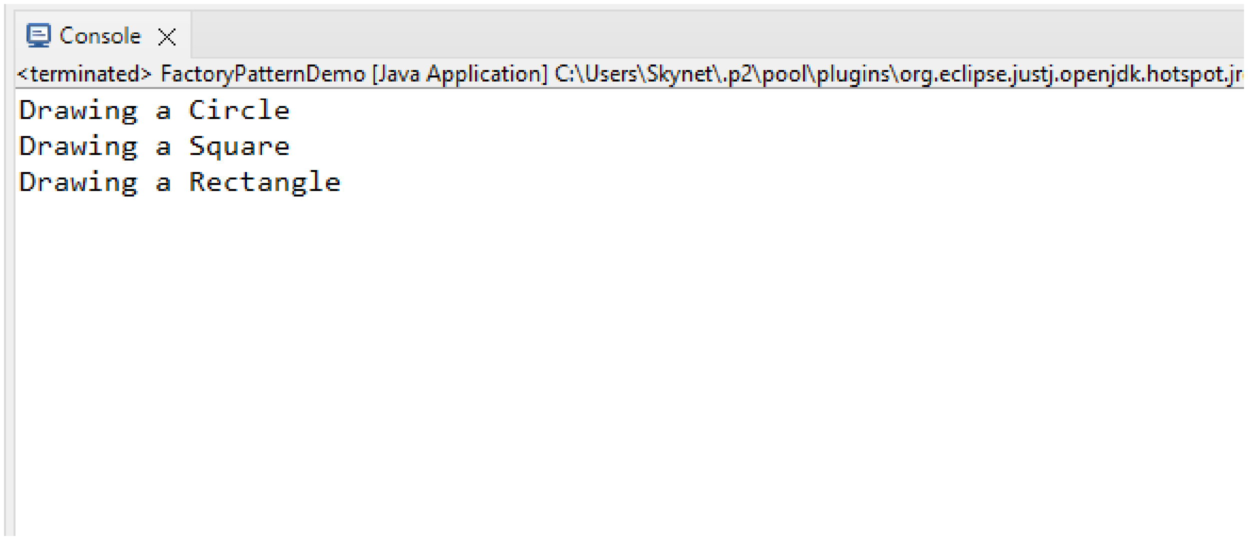


```java

        public Shape getShape(String shapeType) {
            if (shapeType == null) {
                return null;
            }
            if (shapeType.equalsIgnoreCase("CIRCLE")) {
                return new Circle();
            } else if (shapeType.equalsIgnoreCase("SQUARE")) {
                return new Square();
            } else if (shapeType.equalsIgnoreCase("RECTANGLE")) {
                return new Rectangle();
            }
            return null;
        }
    }


public class FactoryPatternDemo {


        public static void main(String[] args) {
            ShapeFactory shapeFactory = new ShapeFactory();


            Shape shape1 = shapeFactory.getShape("CIRCLE");
            shape1.draw();


            Shape shape2 = shapeFactory.getShape("SQUARE");
            shape2.draw();
```


FactoryPatternDemo.java

```java
            } else if (shapeType.equalsIgnoreCase("RECTANGLE")) {
                return new Rectangle();
            }
            return null;
        }
    }


public class FactoryPatternDemo {


        public static void main(String[] args) {
            ShapeFactory shapeFactory = new ShapeFactory();


            Shape shape1 = shapeFactory.getShape("CIRCLE");
            shape1.draw();


            Shape shape2 = shapeFactory.getShape("SQUARE");
            shape2.draw();


            Shape shape3 = shapeFactory.getShape("RECTANGLE");
            shape3.draw();
        }
    }

```

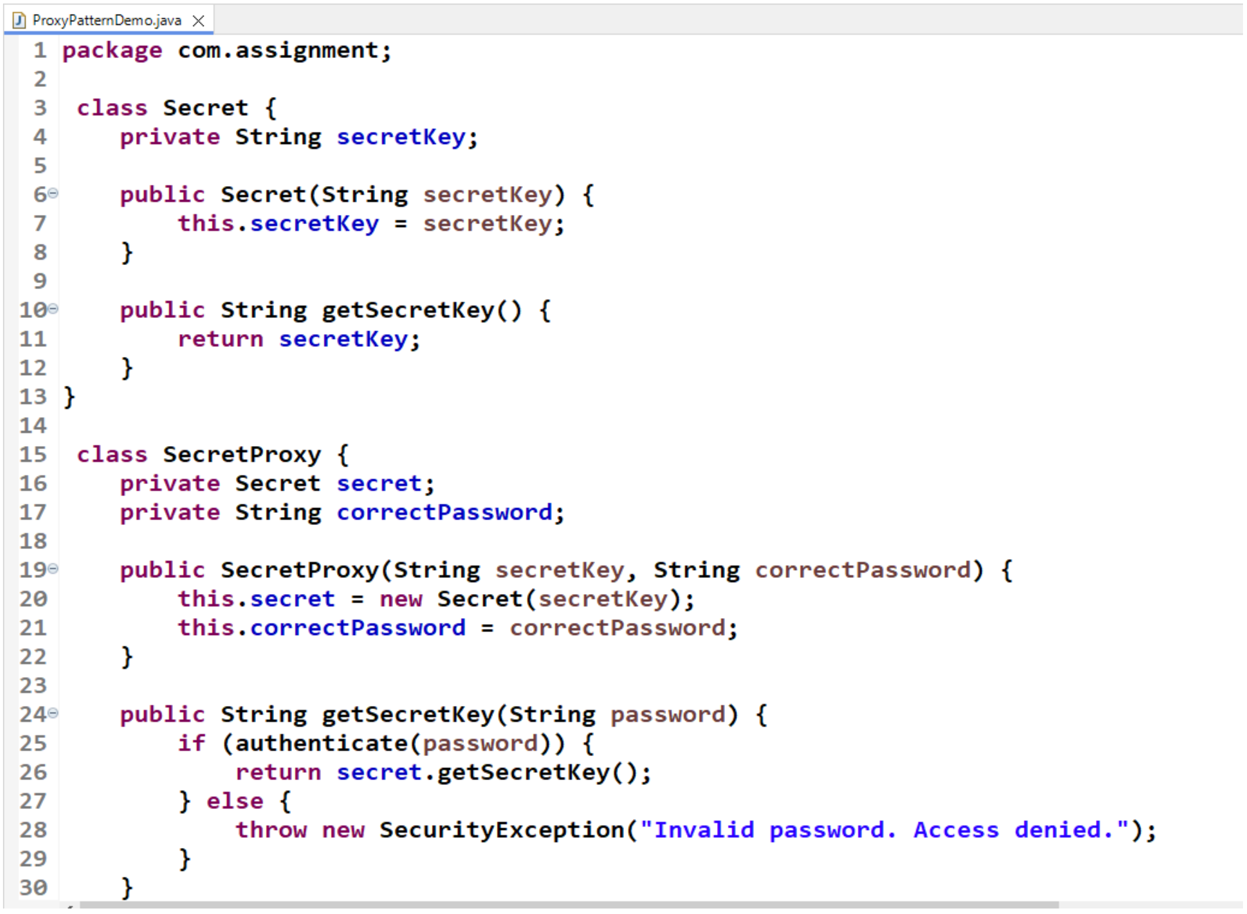
Output :-


Console

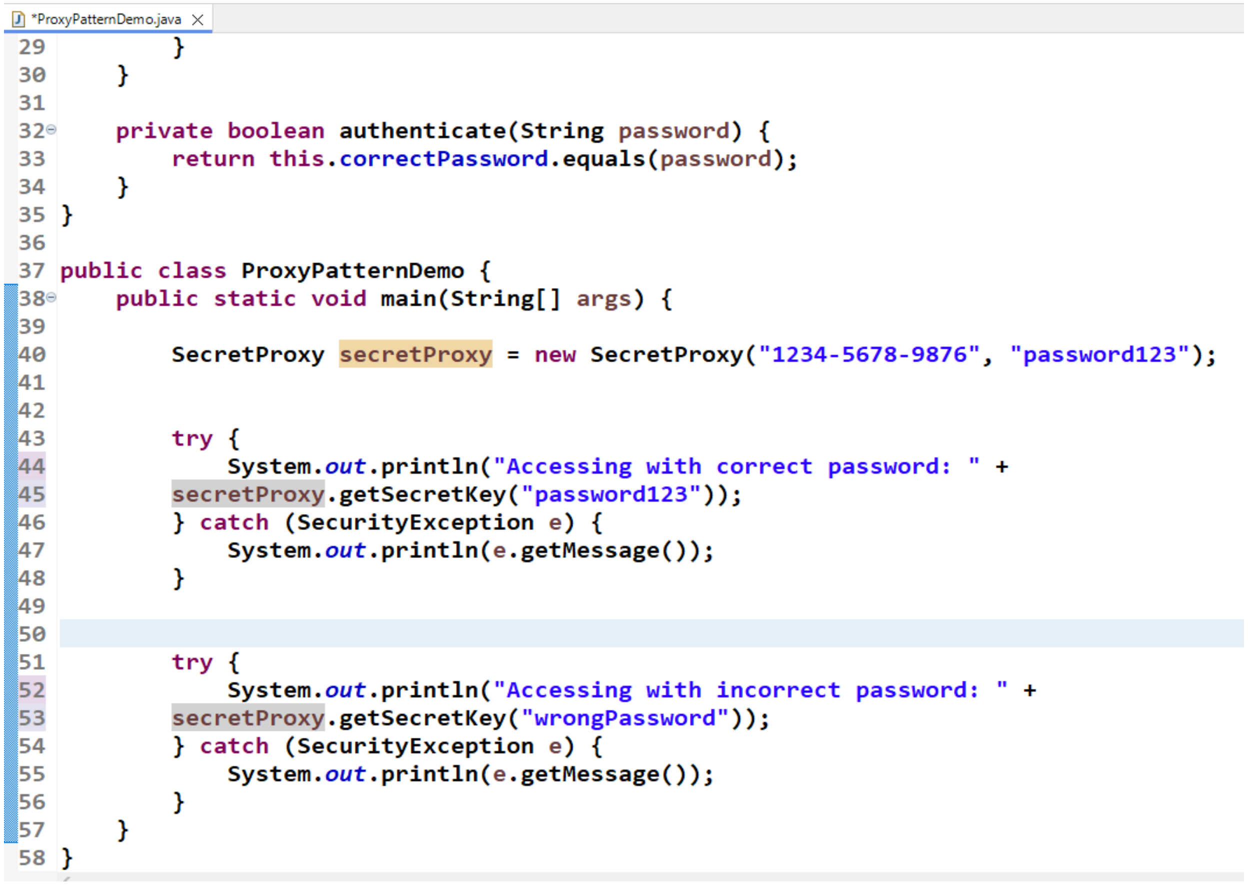
```
<terminated> FactoryPatternDemo [Java Application] C:\Users\Skynet\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jr
Drawing a Circle
Drawing a Square
Drawing a Rectangle
```

3] Task 3: Proxy
Create a proxy class for accessing a sensitive object that contains a secret key. The proxy should only allow access to the secret key if a correct password is provided.

Solution :-
Code : -

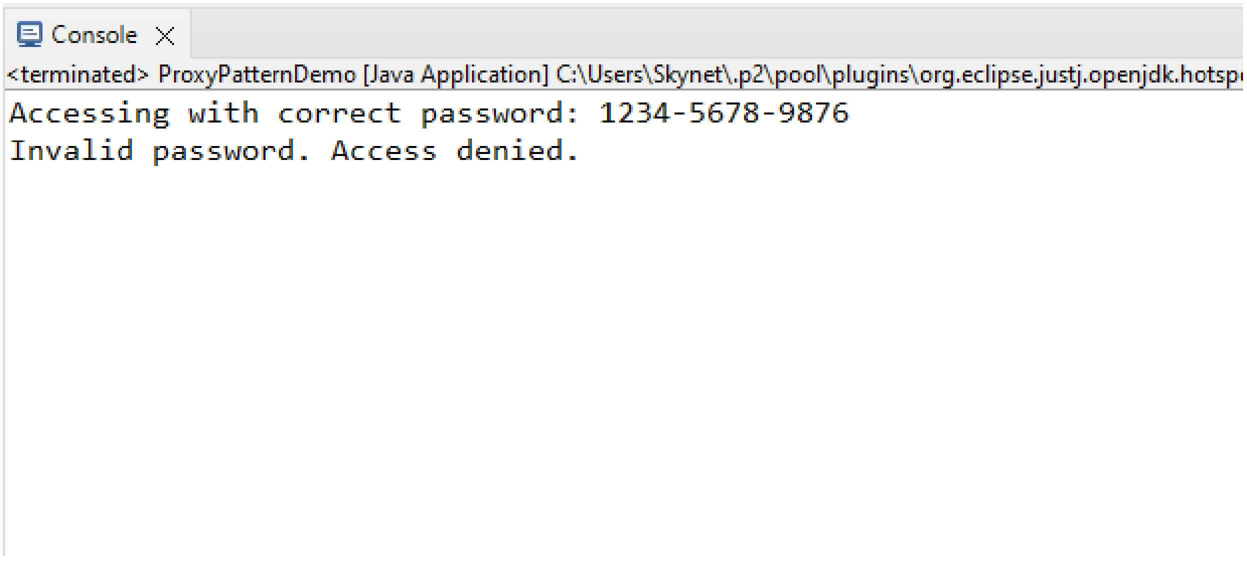


```java
package com.assignment;

 class Secret {
    private String secretKey;

    public Secret(String secretKey) {
        this.secretKey = secretKey;
    }

    public String getSecretKey() {
        return secretKey;
    }
}

 class SecretProxy {
    private Secret secret;
    private String correctPassword;

    public SecretProxy(String secretKey, String correctPassword) {
        this.secret = new Secret(secretKey);
        this.correctPassword = correctPassword;
    }

    public String getSecretKey(String password) {
        if (authenticate(password)) {
            return secret.getSecretKey();
        } else {
            throw new SecurityException("Invalid password. Access denied.");
        }
    }
```


*ProxyPatternDemo.java

```
            }
        }

        private boolean authenticate(String password) {
            return this.correctPassword.equals(password);
        }
}

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

        SecretProxy secretProxy = new SecretProxy("1234-5678-9876", "password123");


        try {
            System.out.println("Accessing with correct password: " +
        secretProxy.getSecretKey("password123"));
        } catch (SecurityException e) {
            System.out.println(e.getMessage());
        }


        try {
            System.out.println("Accessing with incorrect password: " +
        secretProxy.getSecretKey("wrongPassword"));
        } catch (SecurityException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Output :-

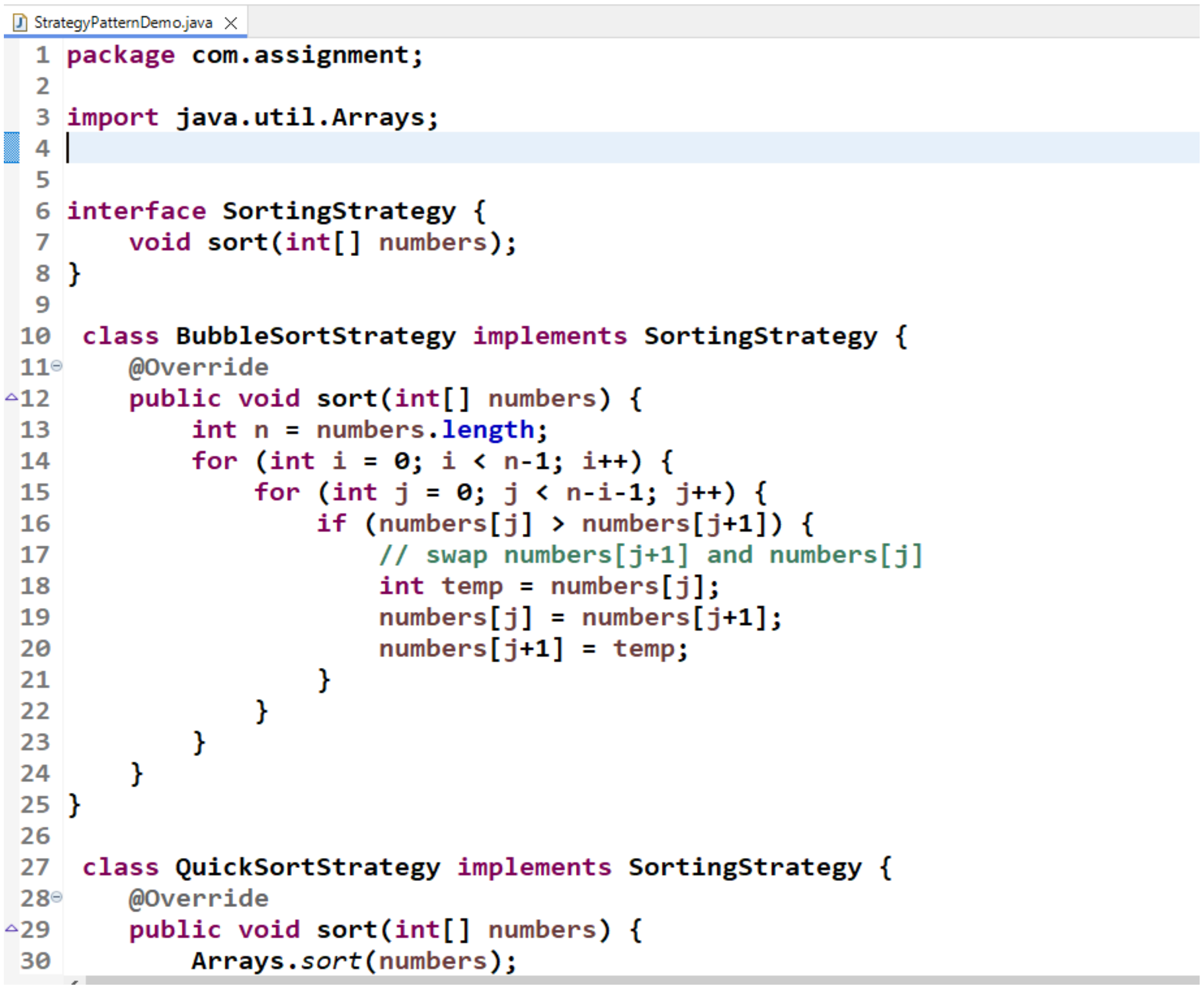

Console

```
<terminated> ProxyPatternDemo [Java Application] C:\Users\Skynet\.p2\pool\plugins\org.eclipse.justj.openjdk.hotsp
Accessing with correct password: 1234-5678-9876
Invalid password. Access denied.
```

4] Task 4: Strategy

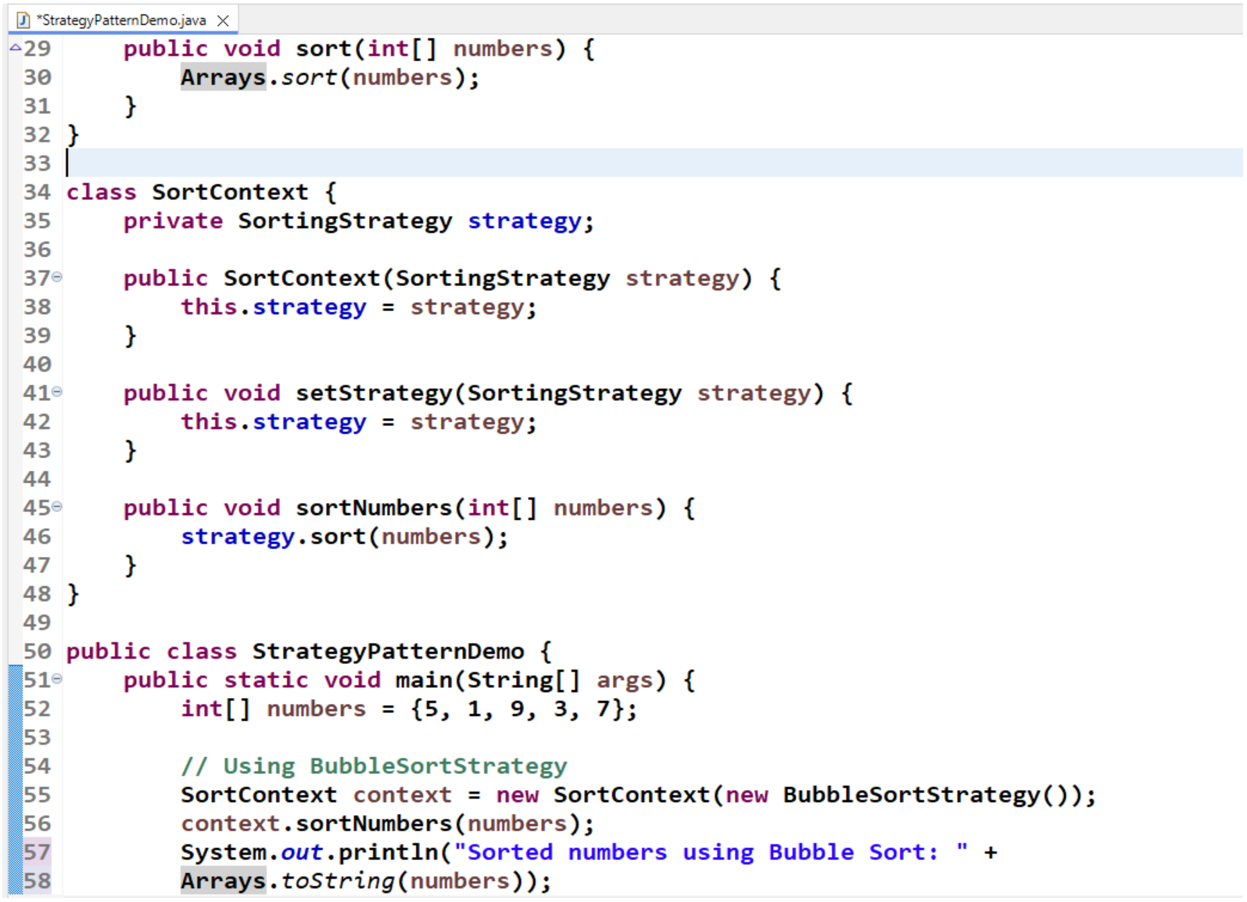
Develop a Context class that can use different SortingStrategy algorithms interchangeably to sort a collection of numbers

Solution :-
Code :-

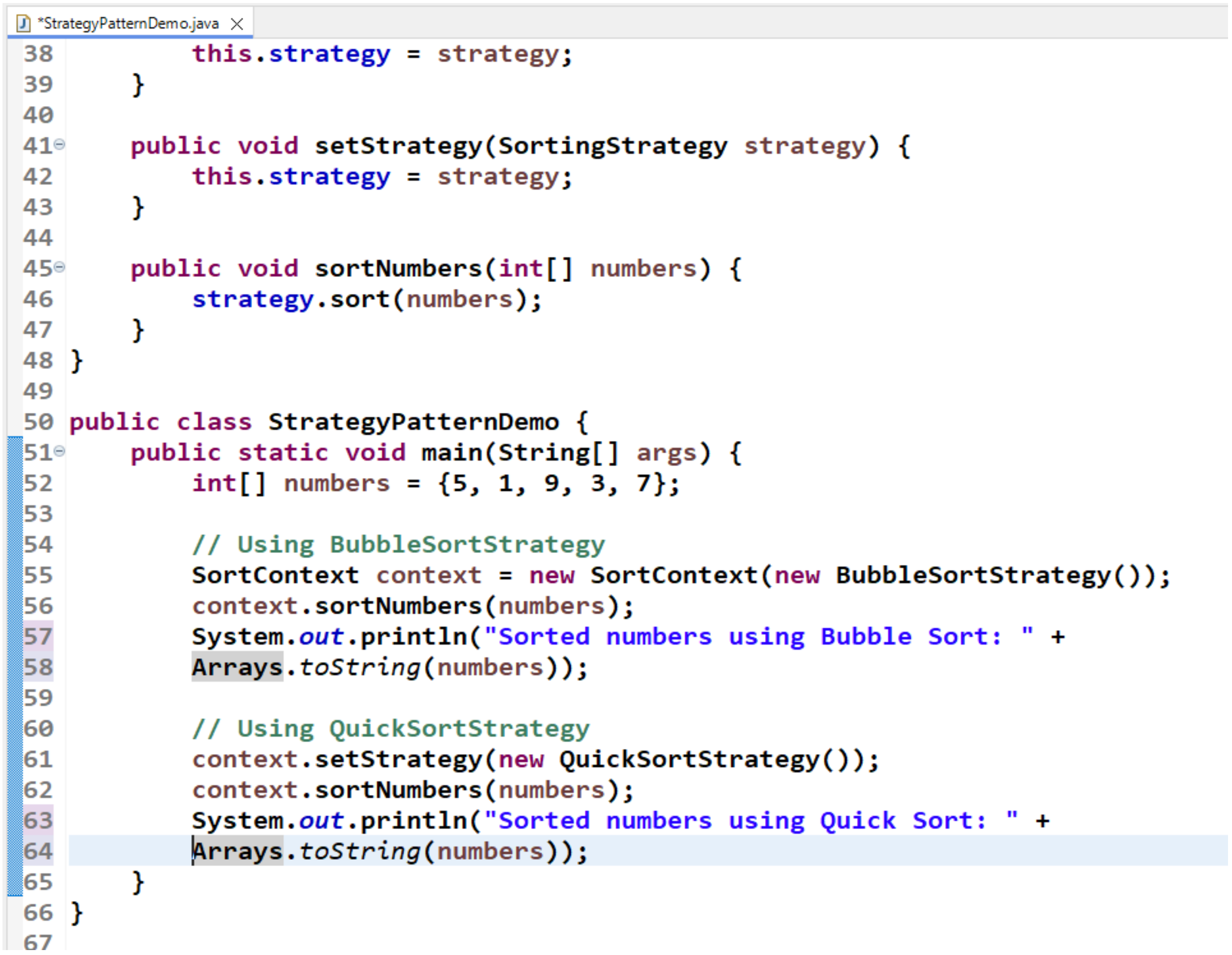


StrategyPatternDemo.java

```java
package com.assignment;

import java.util.Arrays;


interface SortingStrategy {
    void sort(int[] numbers);
}

 class BubbleSortStrategy implements SortingStrategy {
    @Override
    public void sort(int[] numbers) {
        int n = numbers.length;
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n-i-1; j++) {
                if (numbers[j] > numbers[j+1]) {
                    // swap numbers[j+1] and numbers[j]
                    int temp = numbers[j];
                    numbers[j] = numbers[j+1];
                    numbers[j+1] = temp;
                }
            }
        }
    }
}

 class QuickSortStrategy implements SortingStrategy {
    @Override
    public void sort(int[] numbers) {
        Arrays.sort(numbers);
```


*StrategyPatternDemo.java

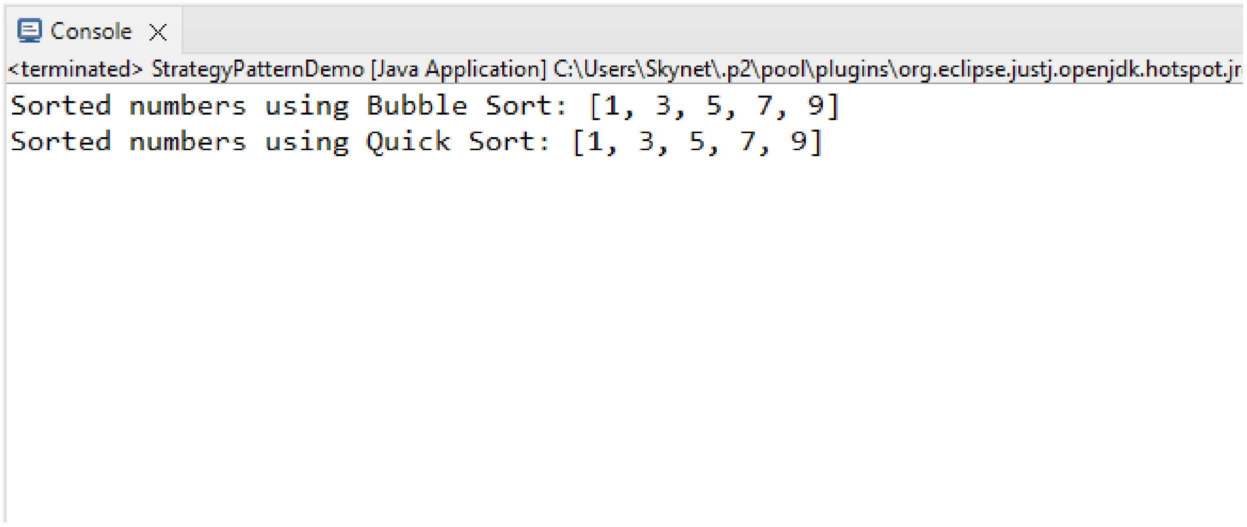
```java
    public void sort(int[] numbers) {
        Arrays.sort(numbers);
    }
}

class SortContext {
    private SortingStrategy strategy;

    public SortContext(SortingStrategy strategy) {
        this.strategy = strategy;
    }

    public void setStrategy(SortingStrategy strategy) {
        this.strategy = strategy;
    }

    public void sortNumbers(int[] numbers) {
        strategy.sort(numbers);
    }
}

public class StrategyPatternDemo {
    public static void main(String[] args) {
        int[] numbers = {5, 1, 9, 3, 7};

        // Using BubbleSortStrategy
        SortContext context = new SortContext(new BubbleSortStrategy());
        context.sortNumbers(numbers);
        System.out.println("Sorted numbers using Bubble Sort: " +
        Arrays.toString(numbers));
```



*StrategyPatternDemo.java

```java
        this.strategy = strategy;
    }

    public void setStrategy(SortingStrategy strategy) {
        this.strategy = strategy;
    }

    public void sortNumbers(int[] numbers) {
        strategy.sort(numbers);
    }
}

public class StrategyPatternDemo {
    public static void main(String[] args) {
        int[] numbers = {5, 1, 9, 3, 7};

        // Using BubbleSortStrategy
        SortContext context = new SortContext(new BubbleSortStrategy());
        context.sortNumbers(numbers);
        System.out.println("Sorted numbers using Bubble Sort: " +
        Arrays.toString(numbers));

        // Using QuickSortStrategy
        context.setStrategy(new QuickSortStrategy());
        context.sortNumbers(numbers);
        System.out.println("Sorted numbers using Quick Sort: " +
        Arrays.toString(numbers));
    }
}
```

Output :-



Console

<terminated> StrategyPatternDemo [Java Application] C:\Users\Skynet\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jr

```
Sorted numbers using Bubble Sort: [1, 3, 5, 7, 9]
Sorted numbers using Quick Sort: [1, 3, 5, 7, 9]
```