Day 23:

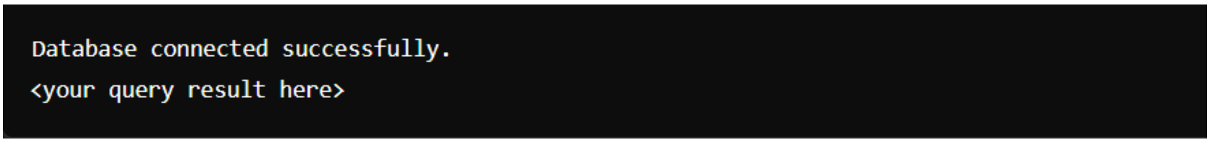
Task 1: Singleton

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

```
package com.wipro;



        import java.sql.Connection;

        import java.sql.DriverManager;

        import java.sql.SQLException;


        public class DatabaseManager {


           private static final DatabaseManager instance = new
DatabaseManager();



           private static final String URL = "jdbc:mysql://localhost:3306/kumar";

           private static final String USERNAME = "root";

           private static final String PASSWORD = "root";



           private Connection connection;



           private DatabaseManager() {
               try {

                   connection = DriverManager.getConnection(URL, USERNAME, PASSWORD);
                   System.out.println("Database connected successfully.");
               } catch (SQLException e) {
                   e.printStackTrace();
                   throw new RuntimeException("Failed to connect to the database", e);
               }
           }


           public static DatabaseManager getInstance() {
               return instance;
           }


           public Connection getConnection() {
               return connection;
           }


        }


package com.wipro;
```

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

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

        DatabaseManager dbManager = DatabaseManager.getInstance();


        Connection connection = dbManager.getConnection();

        try {

            Statement statement = connection.createStatement();


            ResultSet resultSet = statement.executeQuery("SELECT * FROM
yourtable");


            while (resultSet.next()) {

                System.out.println(resultSet.getString(1));
            }
            resultSet.close();
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {

                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

OUTPUT:



```
Database connected successfully.
<your query result here>
```

Task 2: Factory Method

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

```
package kk;

public class Main {
```

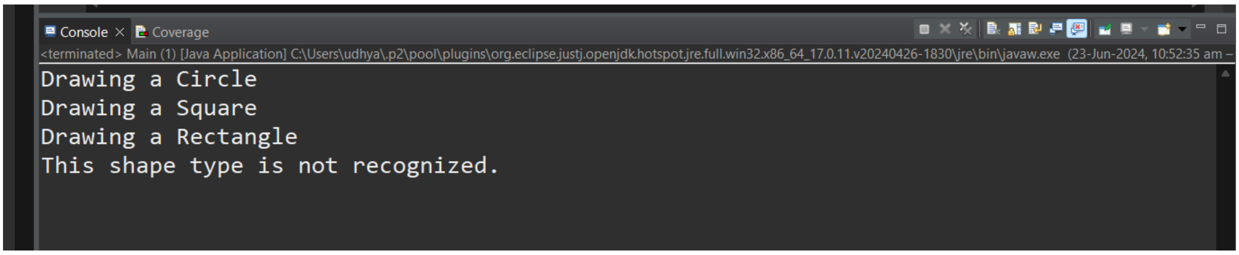
```java
    interface Shape {
  void draw();
}


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


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


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

```
}



static class ShapeFactory {

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



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

```
if (shape1 != null) {
    shape1.draw();
} else {
    System.out.println("This shape type is not recognized.");
}

Shape shape2 = shapeFactory.getShape("SQUARE");
if (shape2 != null) {
    shape2.draw();
} else {
    System.out.println("This shape type is not recognized.");
}

Shape shape3 = shapeFactory.getShape("RECTANGLE");
if (shape3 != null) {
    shape3.draw();
} else {
    System.out.println("This shape type is not recognized.");
}

Shape shape4 = shapeFactory.getShape("TRIANGLE");
if (shape4 != null) {
    shape4.draw();
```

```
        } else {
            System.out.println("This shape type is not recognized.");
        }
    }
}
```

OUTPUT:



```
<terminated> Main (1) [Java Application] C:\Users\udhya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe  (23-Jun-2024, 10:52:35 am –
Drawing a Circle
Drawing a Square
Drawing a Rectangle
This shape type is not recognized.
```

Task 3: Proxy

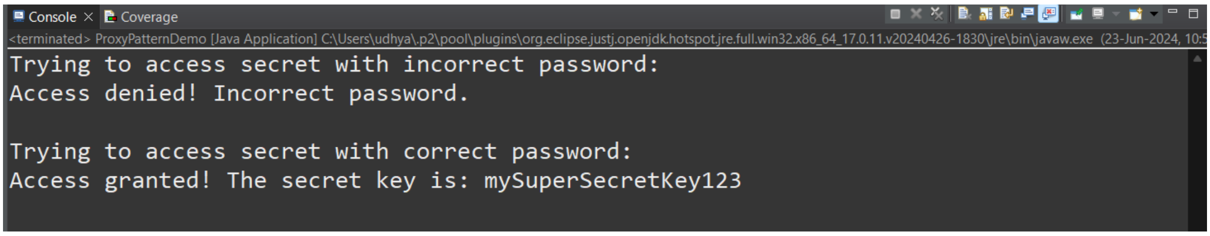
Create a proxy class for accessing a sensitive object that contains a secret key. The proxy should only allow access to the secret key if a correct password is provided.

```
package kk;


public class ProxyPatternDemo {



    interface SensitiveObject {
        void accessSecret(String password);
    }



    static class RealSensitiveObject implements SensitiveObject {
```

```
    private String secretKey = "mySuperSecretKey123";

    @Override
    public void accessSecret(String password) {
        if (password.equals("correctPassword")) {
            System.out.println("Access granted! The secret key is: " + secretKey);
        } else {
            System.out.println("Access denied! Incorrect password.");
        }
    }
}


static class SensitiveObjectProxy implements SensitiveObject {
    private RealSensitiveObject realObject = new RealSensitiveObject();

    @Override
    public void accessSecret(String password) {
        if (password.equals("correctPassword")) {
            realObject.accessSecret(password);
        } else {
            System.out.println("Access denied! Incorrect password.");
        }
    }
}
```

```
    public static void main(String[] args) {
        SensitiveObject proxyObject = new SensitiveObjectProxy();



        System.out.println("Trying to access secret with incorrect password:");
        proxyObject.accessSecret("wrongPassword");



        System.out.println("\nTrying to access secret with correct password:");
        proxyObject.accessSecret("correctPassword");
    }
}
```

OUTPUT:



```
<terminated> ProxyPatternDemo [Java Application] C:\Users\udhya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe  (23-Jun-2024, 10:5
Trying to access secret with incorrect password:
Access denied! Incorrect password.

Trying to access secret with correct password:
Access granted! The secret key is: mySuperSecretKey123
```

Task 4: Strategy

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

```
package kk;


import java.util.Arrays;


public class StrategyPatternDemo {
```

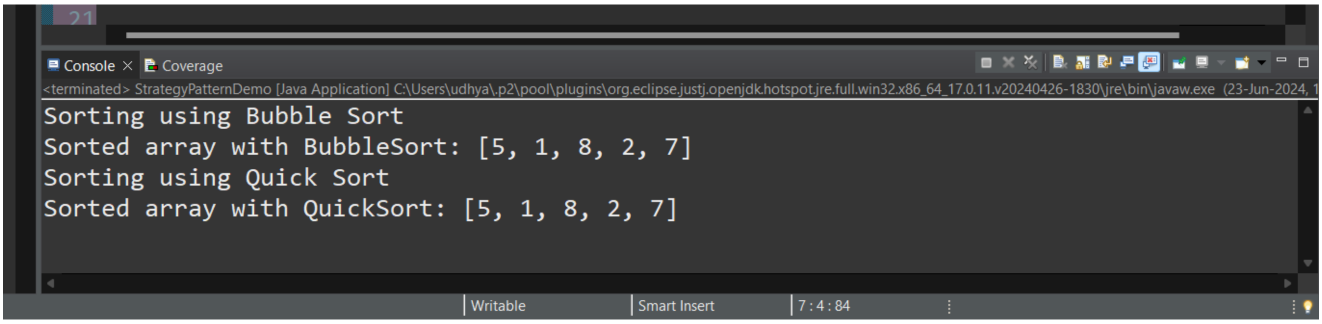
```
interface SortingStrategy {
    void sort(int[] numbers);
}


static class BubbleSortStrategy implements SortingStrategy {
    @Override
    public void sort(int[] numbers) {
        System.out.println("Sorting using Bubble Sort");
        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]) {

                    int temp = numbers[j];
                    numbers[j] = numbers[j + 1];
                    numbers[j + 1] = temp;
                }
            }
        }
    }
}


static class QuickSortStrategy implements SortingStrategy {
```

```
@Override
public void sort(int[] numbers) {
    System.out.println("Sorting using Quick Sort");
    quickSort(numbers, 0, numbers.length - 1);
}

private void quickSort(int[] arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

private int partition(int[] arr, int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;

            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
```

```
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }
}


static class Context {
    private SortingStrategy sortingStrategy;

    public void setSortingStrategy(SortingStrategy sortingStrategy) {
        this.sortingStrategy = sortingStrategy;
    }

    public void sortNumbers(int[] numbers) {
        if (sortingStrategy != null) {
            sortingStrategy.sort(numbers);
        } else {
            System.out.println("Please set a sorting strategy first.");
        }
    }
}
```

```
    public static void main(String[] args) {
        Context context = new Context();



        int[] numbers = {5, 1, 8, 2, 7};



        context.setSortingStrategy(new BubbleSortStrategy());
        context.sortNumbers(numbers.clone());
        System.out.println("Sorted array with BubbleSort: " +
Arrays.toString(numbers));



        context.setSortingStrategy(new QuickSortStrategy());
        context.sortNumbers(numbers.clone());
        System.out.println("Sorted array with QuickSort: " +
Arrays.toString(numbers));
    }
}
```

OUTPUT:



<terminated> StrategyPatternDemo [Java Application] C:\Users\udhya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.11.v20240426-1830\jre\bin\javaw.exe (23-Jun-2024, 1

```
Sorting using Bubble Sort
Sorted array with BubbleSort: [5, 1, 8, 2, 7]
Sorting using Quick Sort
Sorted array with QuickSort: [5, 1, 8, 2, 7]
```