Day 23_Assignments:

Task 1: Singleton

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

Code: -

```
package com.example;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DatabaseConnection {

	private static volatile DatabaseConnection instance;

	private Connection connection;

	private DatabaseConnection() {

		try {

			// Load the JDBC driver


	Class.forName("com.mysql.cj.jdbc.Driver");

			// Establish the connection
```

```
                this.connection =
                        DriverManager.getConnection

    ("jdbc:mysql://localhost:3306/jdbc1", "root",
                                "root@123");
            } catch (ClassNotFoundException |
SQLException e) {
                e.printStackTrace();
                // Handle exceptions
            }
    }
    public static DatabaseConnection getInstance() {
        if (instance == null) {
            synchronized (DatabaseConnection.class)
{
                if (instance == null) {
                    instance = new
DatabaseConnection();
                }
            }
        }
```

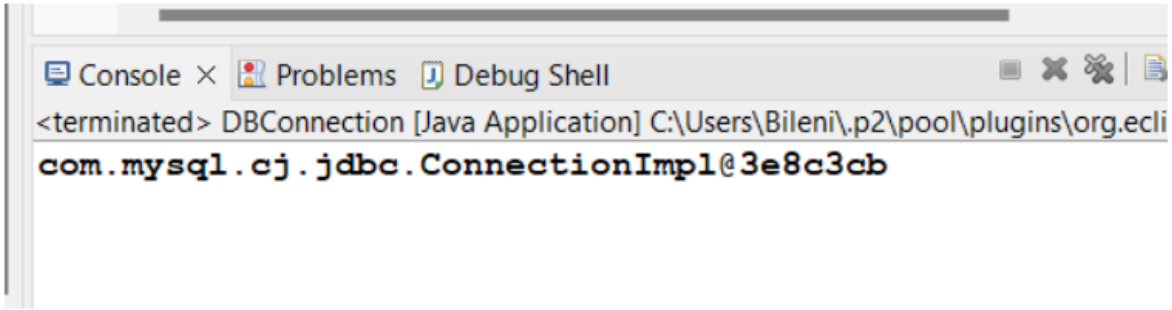
```
        return instance;
    }
    public Connection getConnection() {
        return connection;
    }
}
```

Usage Example: -

```
package com.example;

import java.sql.Connection;
import java.sql.SQLException;

public class Main {
    public static void main(String[] args) {
        // Get the Singleton instance
        DatabaseConnection dbConnection =
DatabaseConnection.getInstance();

        // Retrieve the connection
        Connection connection =
dbConnection.getConnection();

        try {
            // Perform database operations
            var statement = connection.createStatement();
            var resultSet = statement.executeQuery("SELECT
sqlite_version();");

            if (resultSet.next()) {
                System.out.println("SQLite Version: " +
resultSet.getString(1));
            }

            resultSet.close();
```

```
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            // Close the connection
            dbConnection.getConnection();
        }
    }
}
```

Output: -



Task 2: Factory Method

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

Define this shape interface: -

```
package com.example;

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

2. Implement Concrete Shape Classes.

```
package com.example;
public class Circle implements Shape {
```

```
    @Override
    public void draw() {
        System.out.println("Drawing a Circle.");
    }
}
package com.example;
public class Square implements Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a Square.");
    }
}
package com.example;
public class Rectangle implements Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a Rectangle.");
    }
}
```

3. Create the ShapeFactory Class

```
package com.example;
//ShapeFactory.java
public class ShapeFactory {
 // Use getShape method to get the type of shape object
 public Shape getShape(String shapeType) {
     if (shapeType == null) {
         return null;
     }
     switch (shapeType.toUpperCase()) {
         case "CIRCLE":
             return new Circle();
         case "SQUARE":
             return new Square();
         case "RECTANGLE":
             return new Rectangle();
         default:
             return null;
     }
 }
}
```

4. Usage Example

```
package com.example;
//Main.java
public class Main1 {
 public static void main(String[] args) {
     ShapeFactory shapeFactory = new ShapeFactory();

     // Get an object of Circle and call its draw method
     Shape shape1 = shapeFactory.getShape("CIRCLE");
     shape1.draw();

     // Get an object of Square and call its draw method
     Shape shape2 = shapeFactory.getShape("SQUARE");
     shape2.draw();

     // Get an object of Rectangle and call its draw
method
     Shape shape3 = shapeFactory.getShape("RECTANGLE");
     shape3.draw();
 }
}
```

Output: -

```
Drawing a Circle.
Drawing a Square.
Drawing a Rectangle.
```

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.

Class: -

```
package com.example;
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.");
        }
    }

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

```java
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: -

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

Task 4: Strategy

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

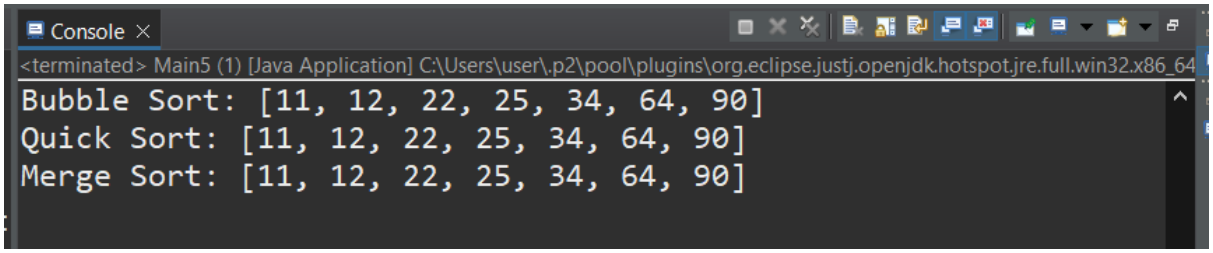
```java
package com.example;

interface SortingStrategy {
    void sort(int[] numbers);
 // BubbleSortStrategy.java
    public 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] and numbers[j +
1]
                        int temp = numbers[j];
                        numbers[j] = numbers[j + 1];
                        numbers[j + 1] = temp;
                    }
                }
            }
        }
    }
 // QuickSortStrategy.java
    public class QuickSortStrategy implements
SortingStrategy {
        @Override
        public void sort(int[] numbers) {
            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++;
                    // Swap arr[i] and arr[j]
                    int temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                }
            }
            // Swap arr[i+1] and arr[high]
            int temp = arr[i + 1];
            arr[i + 1] = arr[high];
            arr[high] = temp;
            return i + 1;
        }
    }
 // MergeSortStrategy.java
    public class MergeSortStrategy implements
SortingStrategy {
        @Override
        public void sort(int[] numbers) {
            if (numbers.length > 1) {
                int mid = numbers.length / 2;

                // Split left part
                int[] left = new int[mid];
                System.arraycopy(numbers, 0, left, 0,
mid);

                // Split right part
                int[] right = new int[numbers.length -
mid];
                System.arraycopy(numbers, mid, right, 0,
numbers.length - mid);

                sort(left);
                sort(right);

                // Merge left and right parts
                merge(numbers, left, right);
```

```
            }
        }

        private void merge(int[] result, int[] left, int[]
right) {
            int i = 0, j = 0, k = 0;
            while (i < left.length && j < right.length) {
                if (left[i] <= right[j]) {
                    result[k++] = left[i++];
                } else {
                    result[k++] = right[j++];
                }
            }
            while (i < left.length) {
                result[k++] = left[i++];
            }
            while (j < right.length) {
                result[k++] = right[j++];
            }
        }
    }
 // SortContext.java
    public class SortContext {
        private SortingStrategy strategy;

        // Set the strategy at runtime
        public void setStrategy(SortingStrategy strategy)
{
            this.strategy = strategy;
        }

        // Sort using the strategy
        public void sort(int[] numbers) {
            if (strategy == null) {
                throw new IllegalStateException("Sorting
strategy not set");
            }
            strategy.sort(numbers);
        }
    }

}
```

```java
package com.example;

import java.util.Arrays;

import com.example.SortingStrategy.BubbleSortStrategy;
import com.example.SortingStrategy.MergeSortStrategy;
import com.example.SortingStrategy.QuickSortStrategy;
import com.example.SortingStrategy.SortContext;

public class Main5 {
    public static void main(String[] args) {
        SortContext context = new SortContext();

        int[] numbers = {64, 34, 25, 12, 22, 11, 90};

        // Sort using Bubble Sort
        context.setStrategy(new BubbleSortStrategy());
        context.sort(numbers);
        System.out.println("Bubble Sort: " +
Arrays.toString(numbers));

        // Sort using Quick Sort
        numbers = new int[]{64, 34, 25, 12, 22, 11, 90};
// Reset array
        context.setStrategy(new QuickSortStrategy());
        context.sort(numbers);
        System.out.println("Quick Sort: " +
Arrays.toString(numbers));

        // Sort using Merge Sort
        numbers = new int[]{64, 34, 25, 12, 22, 11, 90};
// Reset array
        context.setStrategy(new MergeSortStrategy());
        context.sort(numbers);
        System.out.println("Merge Sort: " +
Arrays.toString(numbers));
    }
}
```

Output: -


Console
<terminated> Main5 (1) [Java Application] C:\Users\user\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64
Bubble Sort: [11, 12, 22, 25, 34, 64, 90]
Quick Sort: [11, 12, 22, 25, 34, 64, 90]
Merge Sort: [11, 12, 22, 25, 34, 64, 90]