Day 23 Core_Java Assignments

digvijaythakare2017@gmail.com

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

Code-

```
package com.epwipro.day23;


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

public class DatabaseConnectionManager {

    // Private static instance variable (Lazy Initialization)
    private static DatabaseConnectionManager instance;

    // Private connection variable
    private Connection connection;

    // Database URL, username and password
    private final String url = "jdbc:mysql://localhost:3306/testdb";
    private final String username = "root";
    private final String password = "root";

    // Private constructor to prevent instantiation
    private DatabaseConnectionManager() throws SQLException {
        try {
            // Load MySQL JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            this.connection = DriverManager.getConnection(url, username,
password);
        } catch (ClassNotFoundException ex) {
            throw new SQLException(ex);
        }
    }

    // Public static method to get the instance of the class
    public static DatabaseConnectionManager getInstance() throws
SQLException {
        if (instance == null) {
            synchronized (DatabaseConnectionManager.class) {
                if (instance == null) {
                    instance = new DatabaseConnectionManager();
                }
            }
        }
        return instance;
    }

    // Public method to get the connection
    public Connection getConnection() {
        return connection;
    }

    // Method to close the connection
    public void closeConnection() {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    // Main method for testing the singleton
    public static void main(String[] args) {
        try {
            DatabaseConnectionManager manager =
DatabaseConnectionManager.getInstance();
            Connection conn = manager.getConnection();
            if (conn != null && !conn.isClosed()) {
                System.out.println("Successfully connected to the database.");
                // Perform database operations if needed
            } else {
                System.out.println("Failed to connect to the database.");
            }
            // Close the connection
            manager.closeConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Output-

```
Successfully connected to the database.
```

Task 2: Factory Method Create a ShapeFactory class that encapsulates the object creation logic of different Shape objects like Circle, Square, and Rectangle."

Code 1- Shape interface

```java
package com.epwipro.day23;

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

Code 2-Circle class

```java
package com.epwipro.day23;

public class Circle implements Shape {

	@Override
	public void draw() {
		System.out.println("Drawing a Circle");
	}
}
```

Code 3 Square class

```java
package com.epwipro.day23;

public class Square implements Shape {

	@Override
	public void draw() {
		System.out.println("Drawing a Square");
	}
}
```

Code 4- Rectangle class

```
package com.epwipro.day23;

public class Rectangle implements Shape {

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

Code 5-Shape Factory class-

```
package com.epwipro.day23;

public class ShapeFactory {

        // Method to get an instance of Shape based on the given shape type
        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;
        }
}
```

Code 6 Factory Pattern Demo

```
package com.epwipro.day23;

public class FactoryPatternDemo {

        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.

Code

```
package com.epwipro.day23;

import java.util.Scanner;


interface SensitiveObject {
 String getSecretKey(String password);
}


class RealSensitiveObject implements SensitiveObject {
 private String secretKey;

 RealSensitiveObject(String secretKey) {
     this.secretKey = secretKey;
 }
```

```java
 @Override
 public String getSecretKey(String password) {
     if ("correct_password".equals(password)) {
         return secretKey;
     } else {
         return "Access denied. Incorrect password.";
     }
 }
}


class ProxySensitiveObject implements SensitiveObject {
 private RealSensitiveObject realObject;

 ProxySensitiveObject(String secretKey) {
     this.realObject = new RealSensitiveObject(secretKey);
 }

 @Override
 public String getSecretKey(String password) {

     return realObject.getSecretKey(password);
 }
}


public class Main {
 public static void main(String[] args) {
     Scanner scanner = new Scanner(System.in);


     SensitiveObject proxy = new ProxySensitiveObject("super_secret_key");


     System.out.print("Enter password: ");
     String password = scanner.nextLine();


     String result = proxy.getSecretKey(password);
     System.out.println(result);

     scanner.close();
 }
}
```

Output-1

```
Enter password: correct_password
super_secret_key
```

Output-2

```
Enter password: correct
Access denied. Incorrect password
```

Task 4: Strategy Develop a Context class that can use different SortingStrategy algorithms interchangeably to sort a collection of numbers

Solution-1 Interface Sorting strategy

```
package com.epwipro.day23;

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

2.

```
package com.epwipro.day23;

class BubbleSortStrategy implements SortingStrategy {
	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;
				}
			}
		}
```

```
        }
}

class MergeSortStrategy implements SortingStrategy {
        public void sort(int[] numbers) {
                // Implement merge sort algorithm
                mergeSort(numbers, 0, numbers.length - 1);
        }

        private void mergeSort(int[] numbers, int left, int right) {
                if (left < right) {
                        int mid = (left + right) / 2;
                        mergeSort(numbers, left, mid);
                        mergeSort(numbers, mid + 1, right);
                        merge(numbers, left, mid, right);
                }
        }

        private void merge(int[] numbers, int left, int mid, int right) {
                int n1 = mid - left + 1;
                int n2 = right - mid;

                int[] L = new int[n1];
                int[] R = new int[n2];

                for (int i = 0; i < n1; i++) {
                        L[i] = numbers[left + i];
                }
                for (int j = 0; j < n2; j++) {
                        R[j] = numbers[mid + 1 + j];
                }

                int i = 0, j = 0;
                int k = left;

                while (i < n1 && j < n2) {
                        if (L[i] <= R[j]) {
                                numbers[k++] = L[i++];
                        } else {
                                numbers[k++] = R[j++];
                        }
                }

                while (i < n1) {
                        numbers[k++] = L[i++];
                }

                while (j < n2) {
                        numbers[k++] = R[j++];
                }
        }
}
```

3.

```
package com.epwipro.day23;


class Context {
 private SortingStrategy strategy;

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

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

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

4.Main class

```
package com.epwipro.day23;

public class MainStrategy {
        public static void main(String[] args) {
                int[] numbers = { 5, 1, 4, 2, 8 };

                SortingStrategy bubbleSortStrategy = new BubbleSortStrategy();
                Context context = new Context(bubbleSortStrategy);

                context.performSort(numbers);
                System.out.println("Sorted array using Bubble Sort:");
                printArray(numbers);

                SortingStrategy mergeSortStrategy = new MergeSortStrategy();
                context.setStrategy(mergeSortStrategy);

                context.performSort(numbers);
                System.out.println("Sorted array using Merge Sort:");
        printArray(numbers);
    }

    private static void printArray(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

Output-

```
Sorted array using Bubble Sort:
1 2 4 5 8
Sorted array using Merge Sort:
1 2 4 5 8
```