Day 31 Assignment

Name: Mehul Anjikhane Email: mehulanjikhane13@gmail.com

Task 1: Singleton

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

SingletonDatabase Class:

```
package assignments;

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

public class SingletonDatabase {
	private static SingletonDatabase instance;
	private Connection connection;
	private String url = "jdbc:mysql://localhost:3306/database1";
	private String username = "root";
	private String password = "root";

	private SingletonDatabase() throws SQLException {
		try {
			DriverManager.registerDriver(new Driver());
			this.connection = DriverManager.getConnection(url, username, 
password);
		} catch (ClassNotFoundException | SQLException e) {
			throw new SQLException(e);
		}
	}

	public Connection getConnection() {
		return connection;
	}

	public static SingletonDatabase getInstance() throws SQLException {
		if (instance == null) {
			synchronized (SingletonDatabase.class) {
				if (instance == null) {
					instance = new SingletonDatabase();
				}
			}
		}
		return instance;
	}
}
```

Main Class:

```
package assignments;

import java.sql.SQLException;

public class Mains {
	public static void main(String[] args) {
		try {
			SingletonDatabase db1 = SingletonDatabase.getInstance();
			SingletonDatabase db2 = SingletonDatabase.getInstance();

			System.out.println("Are both instances the same? " + (db1 ==
db2));
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}
}
```

Output:

```
Are both instances the same? True
```

Task 2: Factory Method

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

```
package assignments;

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;
		}
		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 ShapeMain {
	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:

```
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.

```
package assignments;

class SensitiveData {
	private String secretKey = "12345";

	public String getSecretKey() {
		return secretKey;
	}
}

class SensitiveDataProxy {
	private SensitiveData sensitiveData;
	private String password;
	public SensitiveDataProxy(String password) {
		this.sensitiveData = new SensitiveData();
		this.password = password;
	}

	public String getSecretKey(String password) {
		if (this.password.equals(password)) {
			return sensitiveData.getSecretKey();
		} else {
			return "Access Denied!";
		}
	}
}

public class ProxyMain {
	public static void main(String[] args) {
		SensitiveDataProxy proxy = new SensitiveDataProxy("correct_password");

		System.out.println(proxy.getSecretKey("wrong_password")); // Access 
Denied!
		System.out.println(proxy.getSecretKey("correct_password")); // 12345
	}
}
```

Output:

```
Access Denied!
12345
```

Task 4: Strategy

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

```
package  assignments;

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]) {
					int temp = numbers[j];
					numbers[j] = numbers[j + 1];
					numbers[j + 1] = temp;
				}
}
```

```java
                }
        }
}

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++;

                                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;
        }
}

class Context {
        private SortingStrategy strategy;

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

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

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

```java
public class SortingStrategyMain {
	public static void main(String[] args) {
		int[] numbers = { 3, 5, 1, 4, 2 };
		System.out.println("Actual Array: " + Arrays.toString(numbers));

		Context context = new Context(new BubbleSortStrategy());
		context.sort(numbers);
		System.out.println("Bubble Sorted: " + Arrays.toString(numbers));

		int[] numbers2 = { 3, 5, 1, 4, 2 };
		context.setStrategy(new QuickSortStrategy());
		context.sort(numbers2);
		System.out.println("Quick Sorted: " + Arrays.toString(numbers2));
	}
}
```

Output:

```
Actual Array: [3, 5, 1, 4, 2]
Bubble Sorted: [1, 2, 3, 4, 5]
Quick Sorted: [1, 2, 3, 4, 5]
```