Assignment – 24

Prakash Manikanta Irrinki
Prakashnaidu9494@gmail.com

Task 1:

Singleton:

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

Program:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class SingletonDatabase {
	private static SingletonDatabase databaseobj = null;
	private Connection con = null;
	private SingletonDatabase() {
		try {
			con = DriverManager.getConnection
 ("jdbc:mysql://localhost:3306/practicedb", "root", "Sys@123");
		} catch (SQLException e) {
		}
	}
	public static synchronized SingletonDatabase getSingletonobj()
	{
		if (databaseobj == null) {
			databaseobj = new SingletonDatabase();
		}
		return databaseobj;
	}
	public Connection getConnection() {
		return con;
	}
	public void closeConnection() {
		try {
			if (con != null && !con.isClosed()) {
				con.close();
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}
}
public class SingletonMainClass {
	public static void main(String[] args) {
	SingletonDatabase sDatabase1=SingletonDatabase
			.getSingletonobj();
		System.out.println(sDatabase1.hashCode());
		SingletonDatabase sDatabase2 = SingletonDatabase
			.getSingletonobj();
		System.out.println(sDatabase2.hashCode());
	}

}
```

Output:

```
day23.SingletonDatabase@45c7e403
day23.SingletonDatabase@45c7e403
```

Task 2:

Factory Method:

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

Program:

```
public interface Shape {

      void draw();
}
public class Circle implements Shape {
      @Override
      public void draw() {
            System.out.println("Inside Circle :: Draw() method!");
      }
}
public class Square implements Shape {
      @Override
      public void draw() {
            System.out.println("Inside Square :: Draw() method!");
      }
}
public class Ractangle implements Shape {
      @Override
      public void draw() {
            System.out.println("Inside Ractangle :: Draw() method!");
      }
}
public class ShapeFactoryClass {
      public Shape getShape(String str) {
            if (str == null) {
                  return null;
            }
            if (str.equalsIgnoreCase("CIRCLE")) {
                  return new Circle();
            } else if (str.equalsIgnoreCase("SQUARE")) {
                  return new Square();
            } else if (str.equalsIgnoreCase("RACTANGLE")) {
                  return new Ractangle();
            }
            return null;
      }
}
public class ShapeFactoryMain {
      public static void main(String[] args) {
            ShapeFactoryClass sfc=new ShapeFactoryClass();
            Shape s1=sfc.getShape("CIRCLE");
            s1.draw();
            Shape s2=sfc.getShape("SQUARE");
```

```
		s2.draw();
		Shape s3=sfc.getShape("RACTANGLE");
		s3.draw();
	}
}
```

Output:

```
Inside Circle :: Draw() method!
Inside Square :: Draw() method!
Inside Ractangle :: Draw() method!
```

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.

Program:

```
public interface ProxyInterface {

	public void runCommand(String cmd)throws Exception;
}
public class ProxyImpClass implements ProxyInterface {

	@Override
	public void runCommand(String cmd) throws Exception {
		System.out.println(cmd + " Command executed!");
	}

}

public class ProxyClass implements ProxyInterface {

	private boolean isAdmin;
	private ProxyImpClass executor;

	public ProxyClass(String userid, String password) {
		if ("Prakash".equals(userid) &&
"Naidu@1623".equals(password))
			isAdmin = true;
		executor = new ProxyImpClass();
	}

	@Override
	public void runCommand(String cmd) throws Exception {

		if (isAdmin) {
			executor.runCommand(cmd);
		} else {
			throw new Exception("Not allowed to execte the
					commands only allowed Admin");
		}
	}

}
```

```
public class ProxyMainClass {

	public static void main(String[] args) {
		ProxyInterface pe = new ProxyClass("Prakash", 
"Naidu@1623");

		try {
			pe.runCommand("dir");
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

}
```

Output:

```
dir Command executed!
```

Task 4:

Strategy:

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

Program:

```
public interface StrategyInterface {

	void sort(int[] num);
}
public class StrategyBubbleSortImp implements StrategyInterface {

	@Override
	public void sort(int[] num) {
		int n = num.length;
	    for (int i = 0; i < n - 1; i++) {
	        for (int j = 0; j < n - i - 1; j++) {
	            if (num[j] > num[j + 1]) {
	                int temp = num[j];
	                num[j] = num[j + 1];
	                num[j + 1] = temp;
	            }
	        }
	    }
	}

}
public class StrategyInsertionSortImp implements StrategyInterface {

	@Override
	public void sort(int[] num) {
		int n = num.length;
	    for (int i = 1; i < n; ++i) {
	         int key = num[i];
	        int j = i - 1;
```

```
            while (j >= 0 && num[j] > key) {
                num[j + 1] = num[j];
                j = j - 1;
            }
            num[j + 1] = key;
        }
    }

}
public class StrategyClass {

    private StrategyInterface strategy;

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

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

}
public class StrategyMainClass {

    public static void main(String[] args) {
        int[] numbers = { 9, 2, 4, 8, 1, 7, 3, 5 };
        StrategyClass strategyClass = new StrategyClass();
        strategyClass.setStrategy(new StrategyBubbleSortImp());
        strategyClass.performSort(numbers);
        System.out.println("Sorted array using BubbleSort:");
        printArray(numbers);
        strategyClass.setStrategy(new
                                 StrategyInsertionSortImp());
        strategyClass.performSort(numbers);

        System.out.println("Sorted array using InsertionSort:");
        printArray(numbers);
    }

    private static void printArray(int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

}
```

Output:

```
Sorted array using BubbleSort:
1 2 3 4 5 7 8 9
Sorted array using InsertionSort:
1 2 3 4 5 7 8 9
```