**Day 23 – August 2nd**

**TASK 1 – TAGS Annotation**

import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.Tag;  
import org.junit.jupiter.api.Tags;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
  
class Day23\_TagsCase {  
  
 @Test  
 @Tag("firstPriority")  
 void testMethod01() {  
 *assertEquals*(2, 1 + 1); // example test  
 }  
  
 @Test  
 @Tag("firstPriority")  
 void runTestcase02() {  
 *assertEquals*(5, 2 + 3);  
 }  
  
 @Test  
 @Tag("fastTag")  
 void testMethod03() {  
 *assertEquals*("Hello", "Hel" + "lo");  
 }  
  
 @Test  
 @Tag("slowTag")  
 void runTestcase04() {  
 *assertEquals*(9, 3 \* 3);  
 }  
}

**Output**

**4 test cases passed**

import org.junit.jupiter.api.Tag;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
  
public class Day23\_IncludeExcludeTags {  
  
 @Test  
 @Tag("firstPriority")  
 void testMethod01() {  
 *assertEquals*(2, 1 + 1);  
 }  
  
 @Test  
 @Tag("firstPriority")  
 void runTestcase02() {  
 *assertEquals*(5, 2 + 3);  
 }  
  
 @Test  
 @Tag("fastTag")  
 void testMethod03() {  
 *assertEquals*("Hello", "Hel" + "lo");  
 }  
  
 @Test  
 @Tag("slowTag")  
 void runTestcase04() {  
 *assertEquals*(9, 3 \* 3);  
 }  
}

**Including these lines in POM.XML**

<build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>3.1.2</version>  
 <configuration>  
 <properties>  
 <includeTags>firstPriority,fastTag</includeTags>  
 <excludeTags>slowTag</excludeTags>  
 </properties>  
 </configuration>  
 </plugin>  
 </plugins>  
</build>

**Output**

**Only first priority and fast tags test cases get pass and slow tag test case get failed.**

**TASK 2 – Factory method – Example PIZZA**

interface Pizza  
{  
 void prepare();  
 void bake();  
 void cut();  
 void box();  
}  
class PepperoniPizza implements Pizza  
{  
 @Override  
 public void prepare() {  
 System.*out*.println("Preparing Pepperoni Pizza...");  
 }  
  
 @Override  
 public void bake() {  
 System.*out*.println("Baking Pepperoni Pizza...");  
 }  
  
 @Override  
 public void cut() {  
 System.*out*.println("Cutting Pepperoni Pizza...");  
 }  
  
 @Override  
 public void box() {  
 System.*out*.println("Boxing Pepperoni Pizza...");  
 }  
}  
interface PizzaFactory {  
 Pizza createPizza(); // Factory Method  
}  
class PepperoniPizzaFactory implements PizzaFactory  
{  
 @Override  
 public Pizza createPizza() {  
 return new PepperoniPizza(); // Factory Method implementation  
 }  
}  
public class Day23\_FactoryMethod  
{  
 public static void main(String[] args) {  
 PizzaFactory factory = new PepperoniPizzaFactory();  
 Pizza pizza = factory.createPizza();  
  
 pizza.prepare();  
 pizza.bake();  
 pizza.cut();  
 pizza.box();  
 }  
}

**Output**

Preparing Pepperoni Pizza...

Baking Pepperoni Pizza...

Cutting Pepperoni Pizza...

Boxing Pepperoni Pizza...

**Notes**

**Factory Method:** used to define an **interface or abstract class** for creating an object, **but let the subclasses decide** which class to instantiate.

Instead of instantiating the object directly with new, you delegate that responsibility to a factory method (createPizza() in this case).

Pizza pizza = new PepperoniPizza(); // tightly coupled to PepperoniPizza

With Factory Method:

PizzaFactory factory = new PepperoniPizzaFactory();

Pizza pizza = factory.createPizza(); // loosely coupled

we can easily switch to CheesePizzaFactory, VegPizzaFactory, etc., without changing client code.

**TASK 3 – Singleton Method**

class SingletonDemo {  
 private static SingletonDemo *instance*; // lodinids.. //pass/ pin no  
  
 private SingletonDemo() {  
 System.*out*.println("initiating the singleton");  
 }  
 public static SingletonDemo getInstance() {  
 if (*instance* == null) {  
 *instance* = new SingletonDemo();  
 System.*out*.println("in get instance");  
 }  
 return *instance*;  
 }  
  
 public static void doHere() {  
 System.*out*.println("doing here some thing");  
 }  
}  
 public class Day23\_Singelton {  
 public static void main(String[] args) {  
 SingletonDemo.*getInstance*().*doHere*();  
  
  
// SingletonDemo obj = new SingletonDemo();  
// obj.doHere();  
// SingletonDemo obj2 = new SingletonDemo();  
// obj2.doHere();  
 }  
}

**Output**

initiating the singleton

in get instance

doing here some thing

**Notes:**

The **Singleton Pattern** is a **creational design pattern** used when you want to **ensure that only one instance of a class exists in the application** and provide a **global point of access** to that instance.

**When to Use Singleton:**

* Database connections
* Logging frameworks
* Configuration managers
* Cache managers
* Thread pools

**HOME TASK1**

**Write a test case for the below java file**

**public class Junit4Test {**

**public int compare(int n1, int n2) {**

**if (n1 > n2) return 1;**

**return -1;**

**}**

**}**

import org.junit.Test;  
import static org.junit.Assert.*assertEquals*;  
  
public class Day23\_HomeTaskTask1 {  
  
 @Test  
 public void testCompare\_FirstGreater() {  
 Day23\_HomeTask1 obj = new Day23\_HomeTask1();  
 int result = obj.compare(10, 5);  
 *assertEquals*(1, result); // 10 > 5  
 }  
  
 @Test  
 public void testCompare\_SecondGreater() {  
 Day23\_HomeTask1 obj = new Day23\_HomeTask1();  
 int result = obj.compare(3, 8);  
 *assertEquals*(-1, result); // 3 < 8  
 }  
  
 @Test  
 public void testCompare\_BothEqual() {  
 Day23\_HomeTask1 obj = new Day23\_HomeTask1();  
 int result = obj.compare(6, 6);  
 *assertEquals*(-1, result); // 6 == 6, but logic returns -1  
 }  
}

public class Day23\_HomeTask1  
{  
 public int compare(int n1, int n2)  
 {  
 if(n1>n2)  
 {  
 return 1;  
 }  
 return -1;  
 }  
}

**Output: All 3 test cases passed**

**HOME TASK2**

public class Day23\_HomeTask2  
{  
 public int compare(int n1, int n2) {  
 if (n1 > n2) return 1;  
 else if (n1 < n2) return -1;  
 return 0;  
 }  
}

import org.junit.Test;  
import static org.junit.Assert.*assertEquals*;  
  
public class Day23\_HomeTaskTask2 {  
  
 @Test  
 public void testCompare\_FirstGreater() {  
 Day23\_HomeTask2 obj = new Day23\_HomeTask2();  
 int result = obj.compare(10, 5);  
 *assertEquals*(1, result); // 10 > 5  
 }  
  
 @Test  
 public void testCompare\_SecondGreater() {  
 Day23\_HomeTask2 obj = new Day23\_HomeTask2();  
 int result = obj.compare(3, 8);  
 *assertEquals*(-1, result); // 3 < 8  
 }  
  
 @Test  
 public void testCompare\_BothEqual() {  
 Day23\_HomeTask2 obj = new Day23\_HomeTask2();  
 int result = obj.compare(6, 6);  
 *assertEquals*(-1, result); // 6 == 6, but logic returns -1  
 }  
}

**HOME TASK3**

**Include a test case and run the test suite.**

**Test Suite:**

**- its a bundle of unit test cases and run them**

**all together.**

**@Suite**

**@RunWith**

**TestSuite Demo**

**import org.junit.runner.RunWith;**

**import org.junit.runners.Suite;**

**@RunWith(Suite.class)**

**@Suite.SuiteClasses( {**

**JunitTest01.class,**

**JunitTest02.class,**

**JunitTest03.class**

**})**

public class Day23\_HomeTask3  
{  
 private String message;  
 public Day23\_HomeTask3(String message)  
 {  
 this.message=message;  
 }  
 public String Printmsg()  
 {  
 return message;  
 }  
}

import static org.junit.Assert.*assertEquals*;  
import org.junit.Test;  
  
public class Day23\_HomeTask3Test1  
{  
 String msg = "Hi Everyone";  
 Day23\_HomeTask3 obj = new Day23\_HomeTask3(msg);  
 @Test  
 public void msgTest()  
 {  
 System.*out*.println("Message Inside msg.Test");  
 *assertEquals*(msg, obj.Printmsg());  
  
 }  
}

public class Day23\_HomeTask3Test2 {  
  
 String msg = "running test02";  
 Day23\_HomeTask3 Obj1 = new Day23\_HomeTask3(msg);  
  
 @Test  
 public void msgTest() {  
 System.*out*.println("Inside JunitTest02.msgTest()");  
 *assertEquals*(msg, Obj1.Printmsg());  
 }  
}

public class Day23\_HomeTask3Test3 {  
  
 String msg = "running test03";  
 Day23\_HomeTask3 Obj2 = new Day23\_HomeTask3(msg);  
  
 @Test  
 public void msgTest() {  
 System.*out*.println("Inside JunitTest03.msgTest()");  
 *assertEquals*(msg, Obj2.Printmsg());  
 }  
}

import org.junit.runner.RunWith;  
import org.junit.runners.Suite;  
  
@RunWith(Suite.class)  
@Suite.SuiteClasses({Day23\_HomeTask3Test1.class, Day23\_HomeTask3Test2.class, Day23\_HomeTask3Test3.class})  
  
public class Day23\_HomeTask3TestSuite  
{  
  
}

**HOME TASK 4 – Use Test Runner for the above code**

public class Day23\_HomeTask3  
{  
 private String message;  
 public Day23\_HomeTask3(String message)  
 {  
 this.message=message;  
 }  
 public String Printmsg()  
 {  
 return message;  
 }  
}

import static org.junit.Assert.*assertEquals*;  
import org.junit.Test;  
  
public class Day23\_HomeTask3Test1  
{  
 String msg = "Hi Everyone";  
 Day23\_HomeTask3 obj = new Day23\_HomeTask3(msg);  
 @Test  
 public void msgTest()  
 {  
 System.*out*.println("Message Inside msg.Test");  
 *assertEquals*(msg, obj.Printmsg());  
  
 }  
}

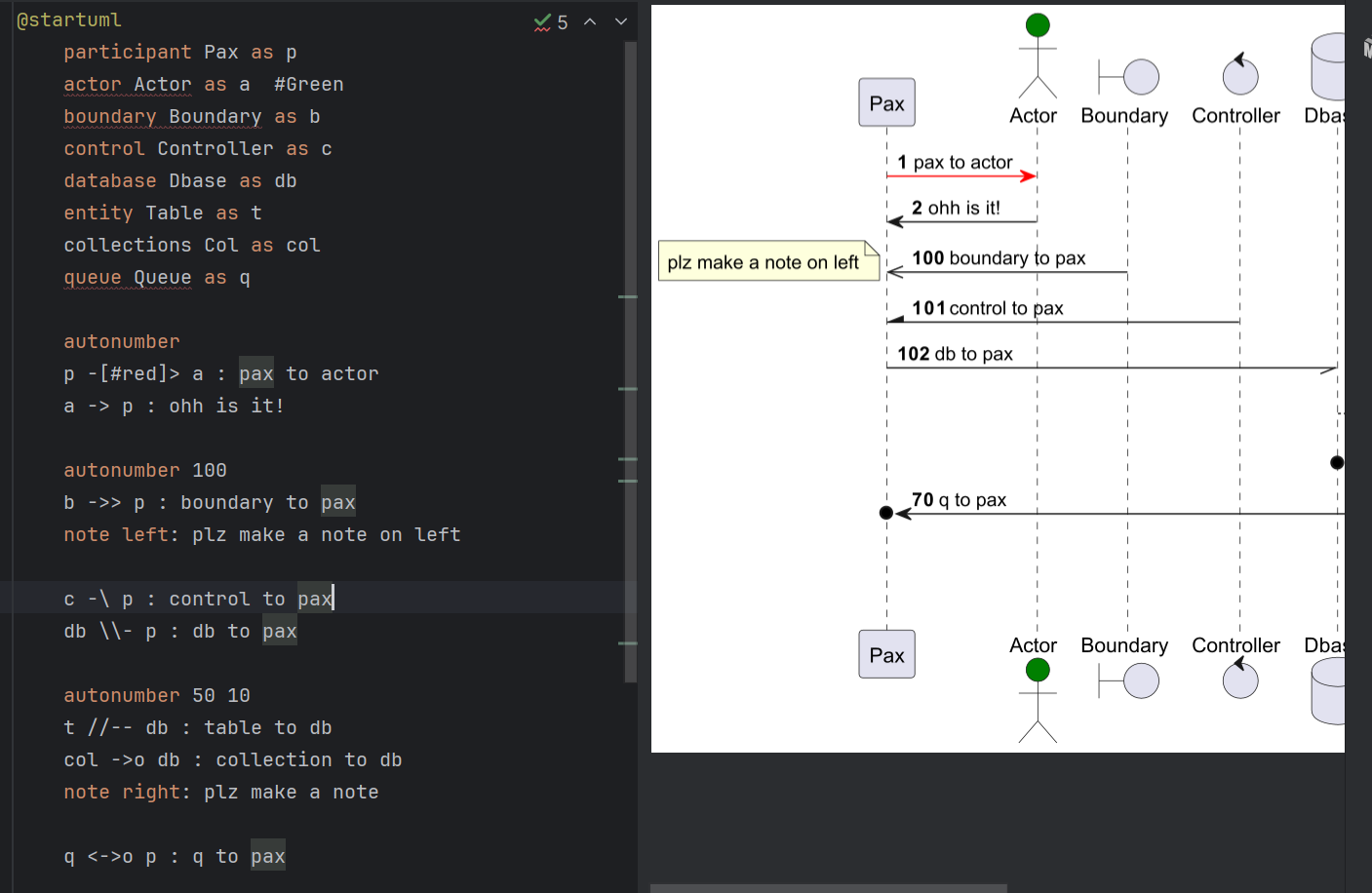
public class Day23\_HomeTask3Test2 {  
  
 String msg = "running test02";  
 Day23\_HomeTask3 Obj1 = new Day23\_HomeTask3(msg);  
  
 @Test  
 public void msgTest() {  
 System.*out*.println("Inside JunitTest02.msgTest()");  
 *assertEquals*(msg, Obj1.Printmsg());  
 }  
}

public class Day23\_HomeTask3Test3 {  
  
 String msg = "running test03";  
 Day23\_HomeTask3 Obj2 = new Day23\_HomeTask3(msg);  
  
 @Test  
 public void msgTest() {  
 System.*out*.println("Inside JunitTest03.msgTest()");  
 *assertEquals*(msg, Obj2.Printmsg());  
 }  
}

import org.junit.runner.RunWith;  
import org.junit.runners.Suite;  
  
@RunWith(Suite.class)  
@Suite.SuiteClasses({Day23\_HomeTask3Test1.class, Day23\_HomeTask3Test2.class, Day23\_HomeTask3Test3.class})  
  
public class Day23\_HomeTask3TestSuite  
{  
  
}

import org.junit.runner.JUnitCore;  
import org.junit.runner.Result;  
import org.junit.runner.notification.Failure;  
  
public class Day23\_HomeTask3TestRunner {  
 public static void main(String[] args) {  
 Result result = JUnitCore.*runClasses*(Day23\_HomeTask3TestSuite.class);  
  
 for (Failure failure : result.getFailures()) {  
 System.*out*.println(failure.toString());  
 }  
  
 System.*out*.println("All tests successful? " + result.wasSuccessful());  
 }  
}

**HOME TASK5 – PlantUML Code**

****

