**ASSIGNMENT 4**

**Design Patterns Implementation**

**Façade Pattern Junit Test Code:**

//ShapeMakerTest.java

**package** facadepattern;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

/\*\*

\* Unit tests for the ShapeMaker class.

\* This class tests the methods in the ShapeMaker class, ensuring that they

\* correctly call the draw method of the respective shape objects.

\*

\* Author: Nathaniel Ashie

\*/

**public** **class** ShapeMakerTest {

/\*\*

\* Test of the drawCircle method, in the ShapeMaker class.

\* This test ensures that the `drawCircle` method calls the `draw` method

\* of the Circle object and returns the correct result.

\*/

@Test

**public** **void** testDrawCircle() {

System.*out*.println("drawCircle"); // Print the method being tested

ShapeMaker instance = **new** ShapeMaker(); // Create an instance of ShapeMaker

String expResult = "Circle::draw()"; // Define the expected result

String result = instance.drawCircle(); // Call the method and capture the result

// Verify the result matches the expected output (Positive test case)

*assertEquals*(expResult, result);

*assertEquals*("Circle::draw()", instance.drawCircle());

// Verify the result does not match incorrect outputs (Negative test case)

*assertNotEquals*("Not Circle", result);

}

/\*\*

\* Test of the drawRectangle method, in the ShapeMaker class.

\* This test ensures that the `drawRectangle` method calls the `draw` method

\* of the Rectangle object and returns the correct result.

\*/

@Test

**public** **void** testDrawRectangle() {

System.*out*.println("drawRectangle"); // Print the method being tested

ShapeMaker instance = **new** ShapeMaker(); // Create an instance of ShapeMaker

String expResult = "Rectangle::draw()"; // Define the expected result

String result = instance.drawRectangle(); // Call the method and capture the result

// Verify the result matches the expected output (Positive test case)

*assertEquals*(expResult, result);

*assertEquals*("Rectangle::draw()", instance.drawRectangle());

// Verify the result does not match incorrect outputs (Negative test case)

*assertNotEquals*("Not Rectangle", result);

}

/\*\*

\* Test of the drawSquare method, in the ShapeMaker class.

\* This test ensures that the `drawSquare` method calls the `draw` method

\* of the Square object and returns the correct result.

\*/

@Test

**public** **void** testDrawSquare() {

System.*out*.println("drawSquare"); // Print the method being tested

ShapeMaker instance = **new** ShapeMaker(); // Create an instance of ShapeMaker

String expResult = "Square::draw()"; // Define the expected result

String result = instance.drawSquare(); // Call the method and capture the result

// Verify the result matches the expected output (Positive test case)

*assertEquals*(expResult, result);

*assertEquals*("Square::draw()", instance.drawSquare());

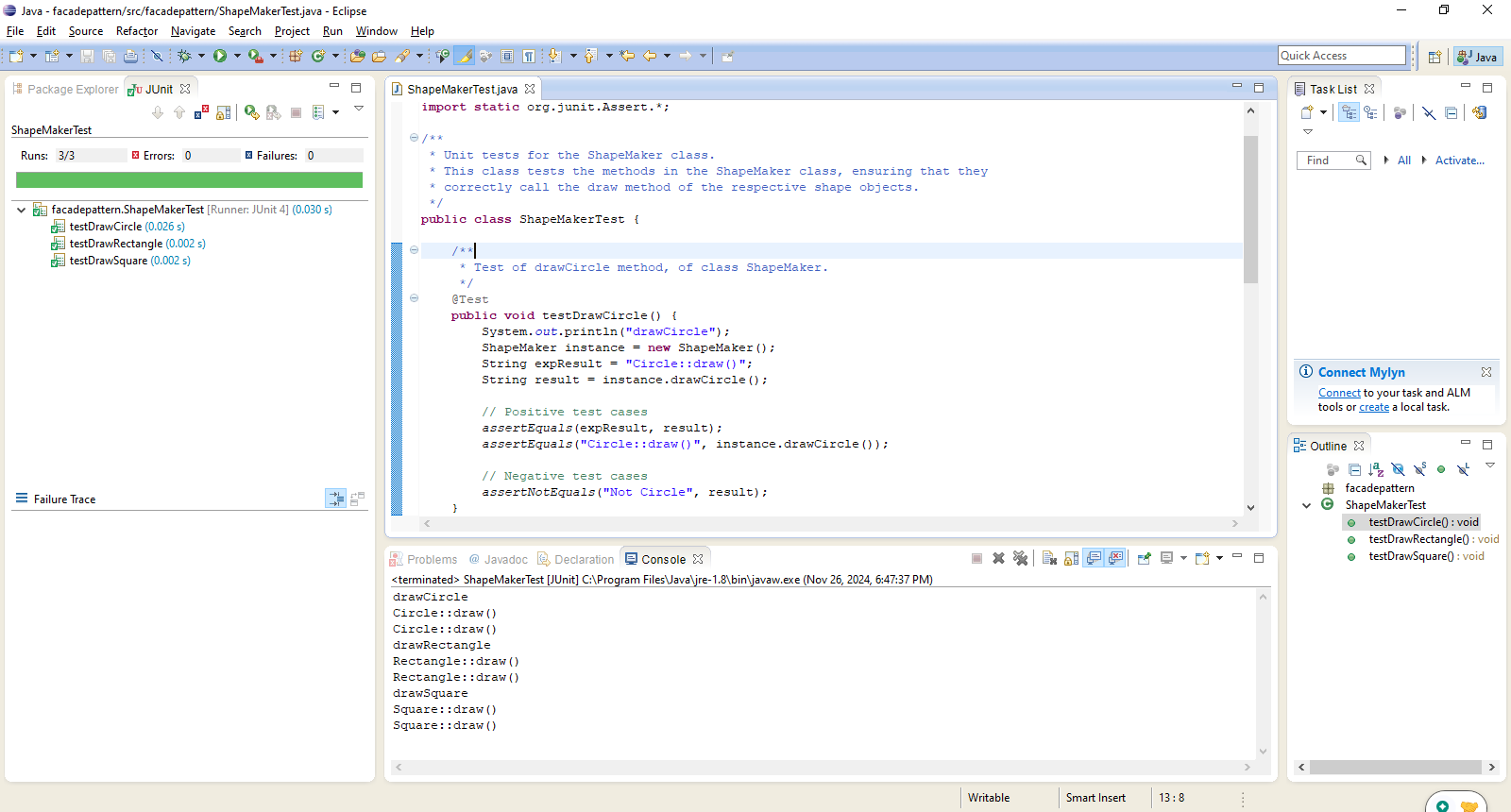
// Verify the result does not match incorrect outputs (Negative test case)

*assertNotEquals*("Not Square", result);

}

}

**Junit Test Output – Façade Pattern**



**Factory Pattern Junit Test Code:**

//ShapeFactoryTest.java

**package** factorypattern;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

/\*\*

\* Unit tests for the ShapeFactory class.

\* This class tests the getShape method of ShapeFactory, ensuring that it

\* correctly returns the appropriate Shape object and that the draw methods

\* behave as expected.

\*

\* **@author** Nathaniel Ashie

\* **@version** 1.0

\*/

**public** **class** ShapeFactoryTest {

/\*\*

\* Test of getShape method for a Circle.

\*/

@Test

**public** **void** testGetCircle() {

System.*out*.println("getCircle");

ShapeFactory shapeFactory = **new** ShapeFactory();

Shape shape = shapeFactory.getShape("CIRCLE");

// Ensure the factory returns a Circle object

*assertNotNull*(shape);

*assertTrue*(shape **instanceof** Circle);

// Capture the draw method output

shape.draw();

String expectedOutput = "Inside Circle::draw() method.";

// Since System.out.print() is used, this test assumes observation of output for verification.

// You can use libraries like System Lambda or OutputCaptureExtension in advanced tests.

}

/\*\*

\* Test of getShape method for a Rectangle.

\*/

@Test

**public** **void** testGetRectangle() {

System.*out*.println("getRectangle");

ShapeFactory shapeFactory = **new** ShapeFactory();

Shape shape = shapeFactory.getShape("RECTANGLE");

// Ensure the factory returns a Rectangle object

*assertNotNull*(shape);

*assertTrue*(shape **instanceof** Rectangle);

// Capture the draw method output

shape.draw();

String expectedOutput = "Inside Rectangle::draw() method.";

}

/\*\*

\* Test of getShape method for a Square.

\*/

@Test

**public** **void** testGetSquare() {

System.*out*.println("getSquare");

ShapeFactory shapeFactory = **new** ShapeFactory();

Shape shape = shapeFactory.getShape("SQUARE");

// Ensure the factory returns a Square object

*assertNotNull*(shape);

*assertTrue*(shape **instanceof** Square);

// Capture the draw method output

shape.draw();

String expectedOutput = "Inside Square::draw() method.";

}

/\*\*

\* Test of getShape method for an invalid input.

\*/

@Test

**public** **void** testInvalidShape() {

System.*out*.println("invalidShape");

ShapeFactory shapeFactory = **new** ShapeFactory();

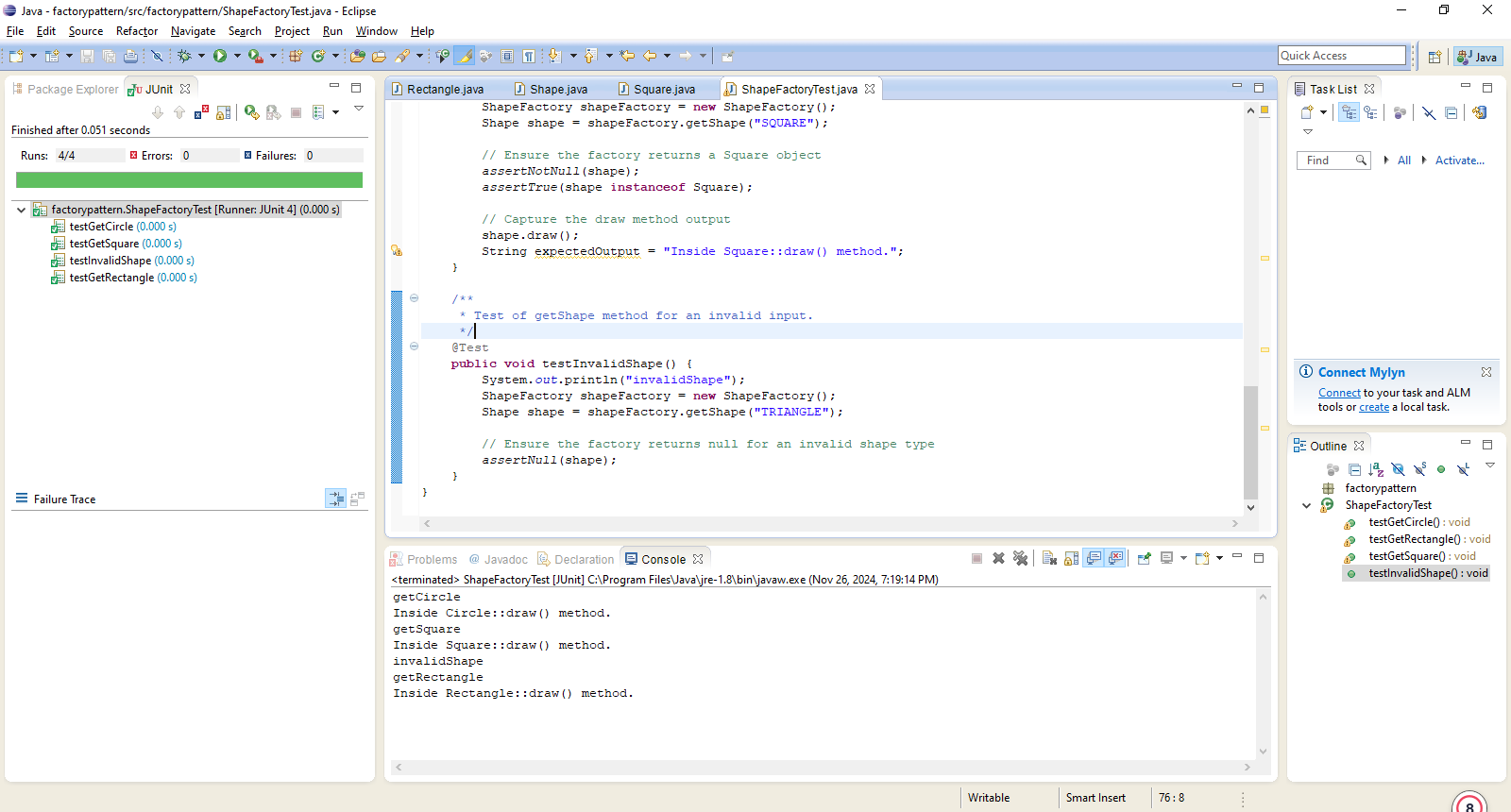
Shape shape = shapeFactory.getShape("TRIANGLE");

// Ensure the factory returns null for an invalid shape type

*assertNull*(shape);

}

**Junit Test Output – FactoryPattern**



**Singleton Pattern Junit Test Code:**

//SingletonPatternTest.java

**package** singletonpattern;

**import** org.junit.Test;

**import** **static** org.junit.Assert.\*;

/\*\*

\* Unit tests for Singleton classes (Configuration and SingleObject).

\* This class ensures that the singleton behavior is correctly implemented.

\*

\* Author: Nathaniel Ashie

\* Version: 1.0

\*/

**public** **class** SingletonPatternTest {

/\*\*

\* Test for the Configuration singleton class.

\* Verifies that:

\* - Configuration maintains a single instance (singleton behavior).

\* - Shared state between instances behaves as expected.

\*/

@Test

**public** **void** testConfigurationSingleton() {

System.*out*.println("Testing Configuration Singleton");

// Get two instances of the Configuration class

Configuration config1 = Configuration.*getInstance*();

Configuration config2 = Configuration.*getInstance*();

// Test if both instances are the same

*assertSame*(config1, config2);

// Set a value using one instance

String settingValue = "DatabaseURL: jdbc:mysql://localhost";

config1.setSetting(settingValue);

// Check if the other instance reflects the same value

*assertEquals*(settingValue, config2.getSetting());

}

/\*\*

\* Test for the SingleObject singleton class.

\* Verifies that:

\* - SingleObject maintains a single instance (singleton behavior).

\* - The showMessage method works as expected.

\*/

@Test

**public** **void** testSingleObjectSingleton() {

System.*out*.println("Testing SingleObject Singleton");

// Get two instances of the SingleObject class

SingleObject object1 = SingleObject.*getInstance*();

SingleObject object2 = SingleObject.*getInstance*();

// Test if both instances are the same

*assertSame*(object1, object2);

// Verify the showMessage method does not break functionality

object1.showMessage(); // This test assumes you manually verify the output.

}

/\*\*

\* Test for lazy initialization in the Configuration class.

\* Verifies that:

\* - The singleton instance is created only when getInstance() is called.

\*/

@Test

**public** **void** testConfigurationLazyInitialization() {

System.*out*.println("Testing Configuration Lazy Initialization");

// Ensure that Configuration instance is null before calling getInstance

**try** {

// Check if lazy initialization works correctly by asserting the instance is not null after calling getInstance.

Configuration config = Configuration.*getInstance*();

*assertNotNull*(config);

} **catch** (Exception e) {

*fail*("Lazy initialization failed: " + e.getMessage());

}

}

/\*\*

\* Test for eager initialization in the SingleObject class.

\* Verifies that:

\* - The singleton instance is created during class loading and is not null.

\*/

@Test

**public** **void** testSingleObjectEagerInitialization() {

System.*out*.println("Testing SingleObject Eager Initialization");

// Since eager initialization occurs during class loading, ensure the instance is not null

SingleObject object = SingleObject.*getInstance*();

*assertNotNull*(object);

}

}

**Junit Test Output – SingletonPattern**

