Entornos de desarrollo

Act.02_UT04. Diseño y realización de pruebas. JUnit

Francisco José García Cutillas | 1FPGS_DAM

Índice

Ejercicio 1

Para realizar esta primera parte trabajaremos con el proyecto ActividadJUnit que está añadido a la Tarea.

**En esta actividad queremos realizar pruebas sobre el método isMayorDeEdad()
de la clase Persona.**

Esta clase tiene una variable de tipo entero denominada edad que si su valor es mayor que 18 y menor que 120 devolverá true y en otro caso devolverá false.

Se pide para la realización de esta actividad

1) **Realizar las pruebas de manera individual, es decir, una prueba por cada ejecución de la prueba de test, que pruebe los diferentes casos de prueba que se obtienen a través del método de clases de equivalencia y del método de análisis de valores límite. A partir de las pruebas, ¿tiene errores el código? En caso de ser así (que no tiene por qué tenerlos) corrige los mismos.**

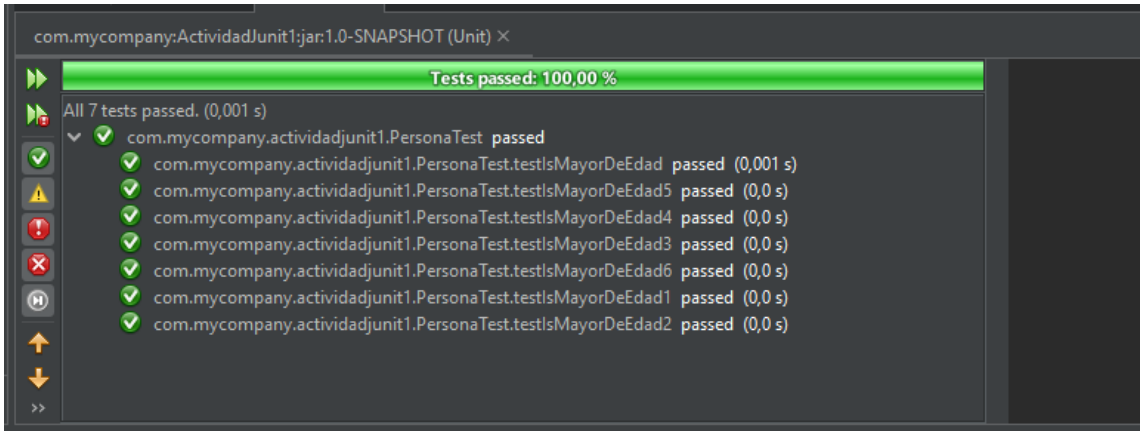
   En este caso consideramos las siguientes salidas:
   - S1 = True.
   - S2 = False.

Condición de entrada	Clases de equivalencia	Clases válidas	COD	Clases no válidas	COD
Edad	Rango	18 >= Edad <= 120	V1	Edad < 18	NV1
				Edad > 120	NV2

Caso de prueba	Clases de equivalencia	Condición de entrada	Resultado esperado
		Edad	
CP1	V1	26	S1
CP2	NV1	15	S2
CP3	NV2	125	S2
Valores límite			
CP4	V1a	18	S1
CP5	V1b	120	S1
CP6	NV1a	17	S2
CP7	NV2b	121	S2

```java
package com.mycompany.actividadjunit1;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

/**
 *
 * @author Fran
 */
public class PersonaTest {

    public PersonaTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }

    /**
     * Test of isMayorDeEdad method, of class Persona.
     */
    @Test
    public void testIsMayorDeEdad() {

        Persona instance = new Persona( edad: 26);
        boolean expResult = true;
        boolean result = instance.isMayorDeEdad();
        assertEquals( expected: expResult,  actual: result);

    }
```

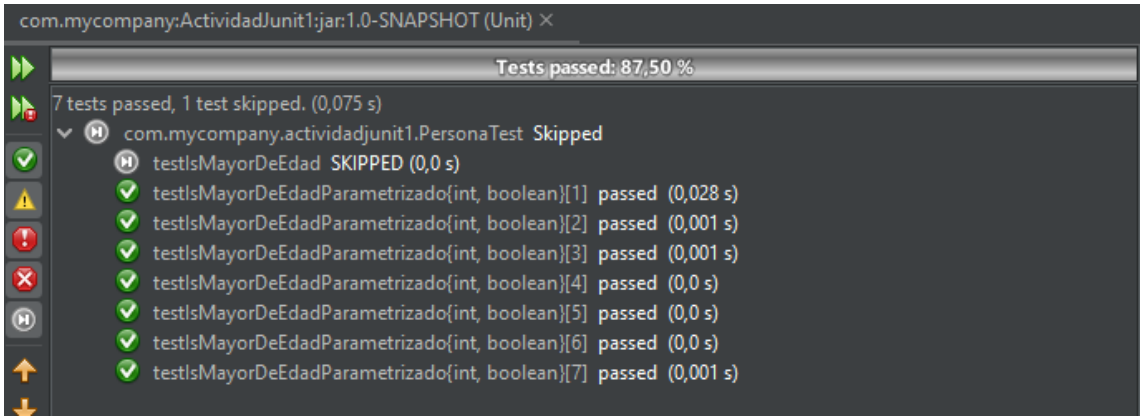
```java
    @Test
    public void testIsMayorDeEdad1() {

        Persona instance = new Persona( edad: 15);
        boolean expResult = false;
        boolean result = instance.isMayorDeEdad();
        assertEquals( expected: expResult,  actual: result);

    }

    @Test
    public void testIsMayorDeEdad2() {

        Persona instance = new Persona( edad: 125);
        boolean expResult = false;
        boolean result = instance.isMayorDeEdad();
        assertEquals( expected: expResult,  actual: result);

    }

    @Test
    public void testIsMayorDeEdad3() {

        Persona instance = new Persona( edad: 18);
        boolean expResult = true;
        boolean result = instance.isMayorDeEdad();
        assertEquals( expected: expResult,  actual: result);

    }

    @Test
    public void testIsMayorDeEdad4() {

        Persona instance = new Persona( edad: 120);
        boolean expResult = true;
        boolean result = instance.isMayorDeEdad();
        assertEquals( expected: expResult,  actual: result);

    }

    @Test
    public void testIsMayorDeEdad5() {

        Persona instance = new Persona( edad: 17);
        boolean expResult = false;
        boolean result = instance.isMayorDeEdad();
        assertEquals( expected: expResult,  actual: result);

    }

    @Test
    public void testIsMayorDeEdad6() {

        Persona instance = new Persona( edad: 121);
        boolean expResult = false;
        boolean result = instance.isMayorDeEdad();
        assertEquals( expected: expResult,  actual: result);

    }

}
```



En este caso, el código no posee errores ya que cumple con las exigencias que se piden. Debe dar true en el rango de edad entre 18 y 120, y false en cualquier valor fuera de dicho rango.
Se podría añadir la comprobación de que no introduzcan una edad negativa, con otra salida diferente “ER1. Introducida edad negativa”, pero esto no cambiaría nada en la salida true del método.

2) En lugar de realizar estas pruebas una a una, se quieren poder hacer todas de una vez. Realiza los cambios oportunos para realizar estas pruebas de manera parametrizada.

```
    @ParameterizedTest
    @CsvSource({

            "26, true",
            "15, false",
            "125, false",
            "18, true",
            "120, true",
            "17, false",
            "121, false"

        })

    public void testIsMayorDeEdadParametrizado(int _edad, boolean esperado) {

        Persona instance = new Persona( edad: _edad);
        boolean resultado = instance.isMayorDeEdad();
        assertEquals( expected: esperado,  actual: resultado);

    }

}
```



Ejercicio 2

Para realizar esta segunda parte trabajaremos con el proyecto ActividadJUnit2 que está añadido a la Tarea.

En esta actividad queremos realizar pruebas sobre el método registraEntradaEmpleado que recibe tres parámetros (es el ejemplo visto en la primera parte de la teoría).

Se definen 3 campos de entrada. La aplicación acepta los datos de esta manera:

- **Empleado. Número de 3 dígitos que no empiece por 0.**
- **Departamento. En blanco o número de 2 dígitos**
- **Oficio. 3 valores posibles: Analista, Diseñador o Programador**

Salida del método:

- **Parámetros correctos: 2500 si es analista, 1500 si es diseñador y 2000 si es programador**
- **Parámetros erróneos: -1 empleado no correcto, -2 departamento no correcto y -3 Oficio no correcto.**

Se pide para la realización de esta actividad

1) **Realizar las pruebas de manera individual, es decir, una prueba por cada ejecución de la prueba de test, que pruebe los diferentes casos de prueba que se obtienen a través del método de clases de equivalencia y del método de análisis de valores límite. A partir de las pruebas, ¿tiene errores el código? En caso de ser así (que no tiene por qué tenerlos) corrige los mismos (se recomienda depurar).**

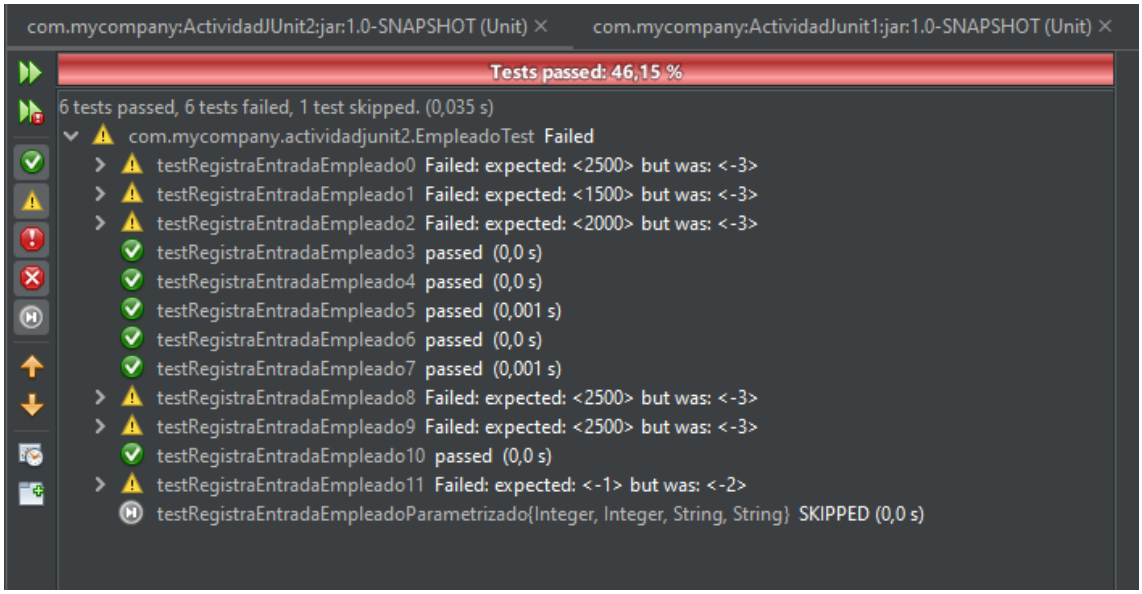
Condición entrada	Clases equivalencia	Clases válidas	COD	Clases no válidas	COD
Empleado	Rango	100 >= Empleado <= 999	V1	Empleado < 100	NV1
				Empleado > 999	NV2
Departamento	Rango	10 >= departamento <= 99	V2	Departamento < 10	NV3
	Lógica	En blanco	V3	Departamento > 99	NV4
Oficio	Miembro de un conjunto	Analista	V4	Cualquier valor que no pertenezca al conjunto	NV5
		Diseñador	V5		
		Programador	V6		

Caso prueba	Clases equivalencia	Condición entrada			Resultado esperado
		Empleado	Departamento	Oficio	
CP1	V1, V2, V4	250	25	Analista	2500
CP2	V1, V3, V5	453		Diseñador	1500
CP3	V1, V2, V6	115	17	Programador	2000
CP4	NV1, V2, V5	71	86	Diseñador	-1
CP5	NV2, V3, NV6	1020		Programador	-1
CP6	V1, NV3, V4	327	2	Analista	-2
CP7	V1, NV4, V5	721	127	Diseñador	-2
CP8	V1, V3, NV6	555		Otro	-3
Valores límite					
CP9	V1a, V2a, V4	100	10	Analista	2500
CP10	V1b, V2b, V5	999	99	Diseñador	1500
CP11	NV1a, NV3a, V6	99	9	Programador	-1
CP12	NV2b, NV4b, V4	1000	100	Analista	-1

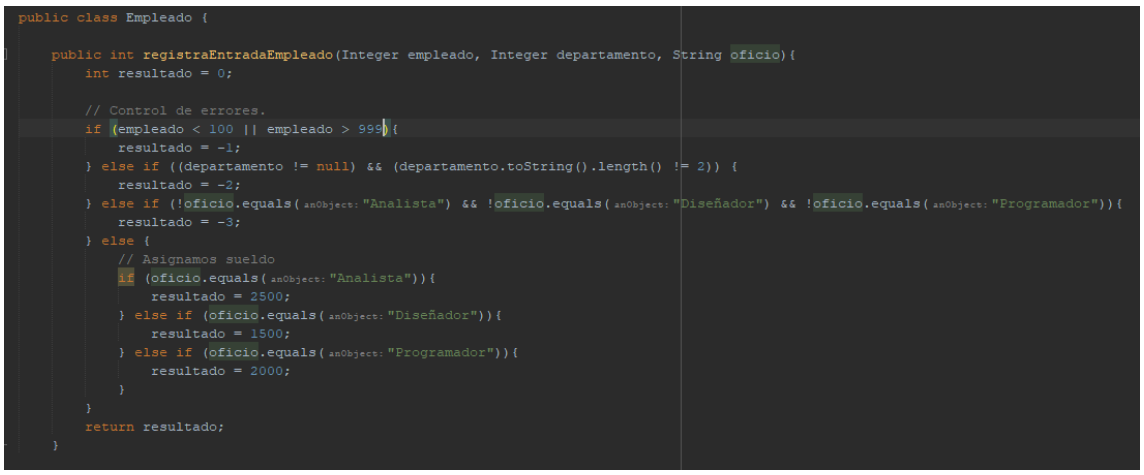
```java
package com.mycompany.actividadjunit2;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Disabled;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;

/**
 *
 * @author Fran
 */
public class EmpleadoTest {

    public EmpleadoTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
    }

    @AfterEach
    public void tearDown() {
    }

    /**
     * Test of registraEntradaEmpleado method, of class Empleado.
     */
    // @Disabled
    @Test
    public void testRegistraEntradaEmpleado() {

        Integer empleado = 250;
        Integer departamento = 25;
        String oficio = "Analista";
        Empleado instance = new Empleado();
        int expResult = 2500;
        int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
        assertEquals( expected: expResult,  actual: result);

    }
    //  @Disabled
    @Test
    public void testRegistraEntradaEmpleado1() {

        Integer empleado = 453;
        Integer departamento = null;
        String oficio = "Diseñador";
        Empleado instance = new Empleado();
        int expResult = 1500;
        int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
        assertEquals( expected: expResult,  actual: result);

    }
```

```
72      // @Disabled
73       @Test
74       public void testRegistraEntradaEmpleado2() {
75
76           Integer empleado = 115;
77           Integer departamento = 17;
78           String oficio = "Programador";
79           Empleado instance = new Empleado();
80           int expResult = 2000;
81           int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
82           assertEquals( expected: expResult,  actual: result);
83
84       }
85      // @Disabled
86       @Test
87       public void testRegistraEntradaEmpleado3() {
88
89           Integer empleado = 71;
90           Integer departamento = 86;
91           String oficio = "Diseñador";
92           Empleado instance = new Empleado();
93           int expResult = -1;
94           int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
95           assertEquals( expected: expResult,  actual: result);
96
97       }
98      // @Disabled
99       @Test
100      public void testRegistraEntradaEmpleado4() {
101
102          Integer empleado = 1020;
103          Integer departamento = null;
104          String oficio = "Programador";
105          Empleado instance = new Empleado();
106          int expResult = -1;
107          int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
108          assertEquals( expected: expResult,  actual: result);
109
110      }
111     // @Disabled
112      @Test
113      public void testRegistraEntradaEmpleado5() {
114
115          Integer empleado = 327;
116          Integer departamento = 2;
117          String oficio = "Analista";
118          Empleado instance = new Empleado();
119          int expResult = -2;
120          int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
121          assertEquals( expected: expResult,  actual: result);
122
123      }
124     // @Disabled
125      @Test
126      public void testRegistraEntradaEmpleado6() {
127
128          Integer empleado = 721;
129          Integer departamento = 127;
130          String oficio = "Diseñador";
131          Empleado instance = new Empleado();
132          int expResult = -2;
133          int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
134          assertEquals( expected: expResult,  actual: result);
```

```
    }
    // @Disabled
    @Test
    public void testRegistraEntradaEmpleado7() {

        Integer empleado = 555;
        Integer departamento = null;
        String oficio = "Otro";
        Empleado instance = new Empleado();
        int expResult = -3;
        int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
        assertEquals( expected: expResult,  actual: result);

    }
    // @Disabled
    @Test
    public void testRegistraEntradaEmpleado8() {

        Integer empleado = 100;
        Integer departamento = 10;
        String oficio = "Analista";
        Empleado instance = new Empleado();
        int expResult = 2500;
        int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
        assertEquals( expected: expResult,  actual: result);

    }
    // @Disabled
    @Test
    public void testRegistraEntradaEmpleado9() {

        Integer empleado = 999;
        Integer departamento = 10;
        String oficio = "Analista";
        Empleado instance = new Empleado();
        int expResult = 2500;
        int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
        assertEquals( expected: expResult,  actual: result);

    }
    //@Disabled
    @Test
    public void testRegistraEntradaEmpleado10() {

        Integer empleado = 99;
        Integer departamento = 9;
        String oficio = "Programador";
        Empleado instance = new Empleado();
        int expResult = -1;
        int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
        assertEquals( expected: expResult,  actual: result);

    }

    @Test
    //@Disabled
    public void testRegistraEntradaEmpleado11() {

        Integer empleado = 1000;
        Integer departamento = 100;
        String oficio = "Analista";
        Empleado instance = new Empleado();
        int expResult = -1;
        int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
        assertEquals( expected: expResult,  actual: result);

    }
}
```



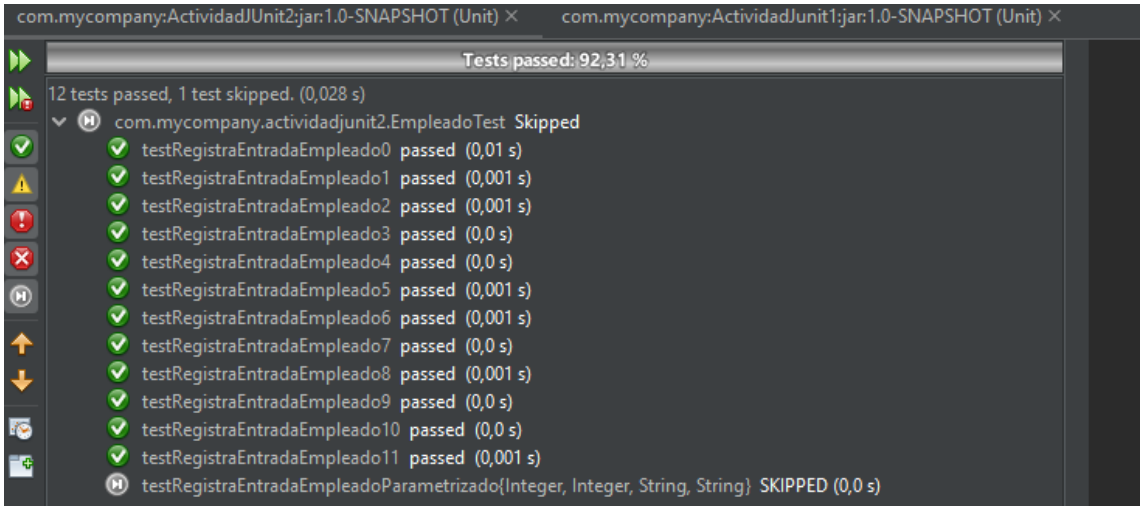
Con las pruebas individuales podemos observar que fallan las pruebas 0, 1, 2, 8, 9 y 11. Las cuales tras depurar el código y corregir los errores, lo dejamos de la siguiente manera:



```
public class Empleado {

    public int registraEntradaEmpleado(Integer empleado, Integer departamento, String oficio){
        int resultado = 0;

        // Control de errores.
        if (empleado < 100 || empleado > 999){
            resultado = -1;
        } else if ((departamento != null) && (departamento.toString().length() != 2)) {
            resultado = -2;
        } else if (!oficio.equals( anObject: "Analista") && !oficio.equals( anObject: "Diseñador") && !oficio.equals( anObject: "Programador")){
            resultado = -3;
        } else {
            // Asignamos sueldo
            if (oficio.equals( anObject: "Analista")){
                resultado = 2500;
            } else if (oficio.equals( anObject: "Diseñador")){
                resultado = 1500;
            } else if (oficio.equals( anObject: "Programador")){
                resultado = 2000;
            }
        }
        return resultado;
    }
```

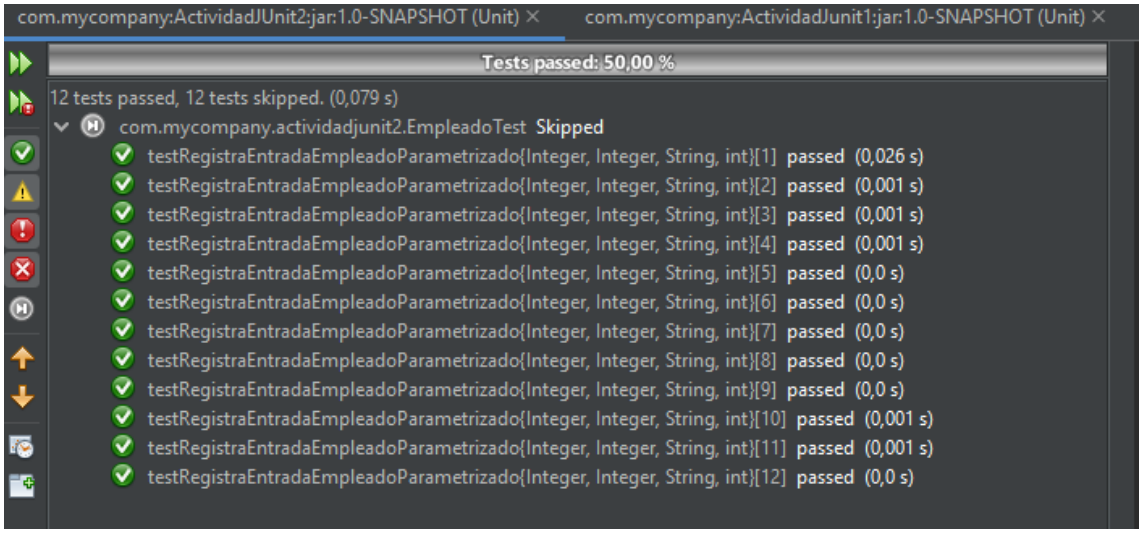
Se ha tenido que modificar la condición en la evaluación de oficio, cambiando los “||” por “&&”, para que así sólo se cumpla la condición en el caso de que oficio NO sea miembro de (Analista, Diseñador o Programador). También se ha modificado en la evaluación de empleado el valor límite por arriba, el cual hemos cambiado por “999” para que sea el límite del rango.

Las pruebas ahora quedarían de la siguiente manera:



2) En lugar de realizar estas pruebas una a una, se quieren poder hacer todas de una vez. Realiza los cambios oportunos para realizar estas pruebas de manera parametrizada.

```
    @ParameterizedTest
    @CsvSource ({
        "250, 25, Analista, 2500",
        "453, , Diseñador, 1500",
        "115, 17, Programador, 2000",
        "71, 86, Diseñador, -1",
        "1020, , Programador, -1",
        "327, 2, Analista, -2",
        "721, 127, Diseñador, -2",
        "555, , Otro, -3",
        "100, 10, Analista, 2500",
        "999, 99, Diseñador, 1500",
        "99, 9, Programador, -1",
        "1000, 100, Analista, -1"

    })

    public void testRegistraEntradaEmpleadoParametrizado(Integer _empleado,
            Integer _departamento, String _oficio, int esperado) {

        Integer empleado = _empleado;
        Integer departamento = _departamento;
        String oficio = _oficio;
        Empleado instance = new Empleado();

        int result = instance.registraEntradaEmpleado(empleado, departamento, oficio);
        assertEquals( expected: esperado,  actual: result);

    }

}
```


com.mycompany:ActividadJUnit2:jar:1.0-SNAPSHOT (Unit)
com.mycompany:ActividadJunit1:jar:1.0-SNAPSHOT (Unit)
Tests passed: 50,00 %
12 tests passed, 12 tests skipped. (0,079 s)
com.mycompany.actividadjunit2.EmpleadoTest Skipped
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[1] passed (0,026 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[2] passed (0,001 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[3] passed (0,001 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[4] passed (0,001 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[5] passed (0,0 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[6] passed (0,0 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[7] passed (0,0 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[8] passed (0,0 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[9] passed (0,0 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[10] passed (0,001 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[11] passed (0,001 s)
testRegistraEntradaEmpleadoParametrizado{Integer, Integer, String, int}[12] passed (0,0 s)