Clase 8

Testing

Unit Tests / Tests de Unidad / JUnit

- Framework para realizar pruebas unitarias.
- Pruebas simples a bloques de código de clases o métodos.
- Detección temprana de errores.
- Repetir pruebas luego de aplicar cambios.
- Pruebas automatizadas escribiendo código.
- Se agrega como una dependencia de maven.

JUnit - Asserts

- **assertEquals**: verifica si dos valores son iguales.
- **assertNotEquals**: verifica si dos valores no son iguales.
- **assertTrue**: verifica si una condición es verdadera.
- **assertFalse**: verifica si una condición es falsa.
- **assertNull**: verifica si un valor es nulo.
- **assertNotNull**: verifica si un valor no es nulo.

JUnit - Asserts

- **assertSame**: verifica si dos valores referencian al mismo objeto.
- **assertNotSame**: verifica si dos valores no referencian al mismo objeto.
- **assertArrayEquals:** para verificar si dos arreglos son iguales.
- **assertThrows:** para verificar si se lanza una excepción
- **assertTimeout:** para verificar si un bloque de código se ejecuta en un tiempo determinado.

Agregando JUnit

```
<properties>
   <maven.compiler.source>17</maven.compiler.source>
   <maven.compiler.target>17</maven.compiler.target>
   <junit.version>4.13.2</junit.version>
</properties>

<dependencies>
   <dependency>
       <groupId>junit</groupId>
       <artifactId>junit</artifactId>
       <version>${junit.version}</version>
       <scope>test</scope>
   </dependency>
</dependencies>
```

Clase a probar

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Circulo {
   protected String nombre;
   protected double radio;

   public double calcularPerimetro() {
       double resultado = Math.PI*2*this.radio;
       return resultado;
   }

   public double calcularSuperficie() {
       double resultado = Math.PI*Math.pow(this.radio,2);
       return resultado;
   }

   public void excepcionSiCero(int valor) throws EjemploException{
       if(valor==0) {
           throw new EjemploException("Ejemplo");
       }
   }
```

Clase de prueba

```
public class CirculoTest {
    Circulo circulo1;
    Circulo circulo2;

    @Before
    public void preparacion() {
        this.circulo1 = new Circulo("circulo", 10);
        this.circulo2 = new Circulo("otro circulo",14);
    }

    @Test
    public void testCalcularPerimetro() {
        assertEquals(62.83185307179586, this.circulo1.calcularPerimetro(),0.0001);
        assertEquals(87.96459430051421, this.circulo2.calcularPerimetro(),0.0001);
    }

    @Test
    public void testCalcularSuperficie() {
        assertEquals(314.1592653589793, this.circulo1.calcularSuperficie(),0.0001);
        assertEquals(615.7521601035994, this.circulo2.calcularSuperficie(),0.0001);
    }
```

Clase de prueba

```
    @Test
    public void testToString() {
        assertEquals("Circulo(nombre=circulo, radio=10.0)", this.circulo1.toString());
        assertEquals("Circulo(nombre=otro circulo, radio=14.0)", this.circulo2.toString());
    }

    @Test(expected = EjemploException.class)
    public void testExcepcion1() throws EjemploException{
        this.circulo1.excepcionSiCero(0);
    }

    @Test
    public void testExcepcion2() {
        try {
            this.circulo1.excepcionSiCero(0);
        } catch (EjemploException e) {
            return;
        }
        fail("Excepcion tirada");
    }
}
```

Mock - Mockito

Definición

Framework que permite simular objetos en pruebas de unidad.

```
<properties>
   <mockito.version>5.3.1</mockito.version>
</properties>

<dependencies>
   <dependency>
       <groupId>org.mockito</groupId>
       <artifactId>mockito-core</artifactId>
       <version>${mockito.version}</version>
       <scope>test</scope>
   </dependency>
</dependencies>
```

Mock - Mockito

```
public class ServicioEjemplo {
    public int suma(int a, int b) {
        return a+b;
    }

    public Circulo devolverCirculo(String nombre, int radio) {
        Circulo c = new Circulo();
        c.setNombre(nombre);
        c.setRadio(radio);
        return c;
    }
}
```

Mock - Mockito

¿Cómo lo trabajo?

- Maqueta de la clase
- Se define cuáles métodos se van a simular
- En base a ciertos parámetros podemos devolver ciertos objetos resultado predefinidos

¿Cuándo usarlo?

- El llamado a servicios es complejo de realizar.
- Cuando una clase todavía no ha sido desarrollada.
- El resultado del llamado es siempre el mismo (mismos parámetros)

Mock - Mockito

```
public class ServicioEjemplo {
    public int suma(int a, int b) {
        return a+b;
    }

    public Circulo devolverCirculo(String nombre, int radio) {
        Circulo c = new Circulo();
        c.setNombre(nombre);
        c.setRadio(radio);
        return c;
    }
}
```

```
public class MockitoTest {
    @Test
    public void pruebaServicioTest1 () {
        ServicioEjemplo servicioMock = mock(ServicioEjemplo.class);
        when(servicioMock.suma(4,4)).thenReturn(8);
        assertEquals(8, servicioMock.suma(4,4));
        verify(servicioMock).suma(4,4);
    }
```

Mock - Mockito

¿Cómo lo trabajo?

- Maqueta de la clase
- Se define cuáles métodos se van a simular
- En base a ciertos parámetros podemos devolver ciertos objetos resultado predefinidos

¿Cuándo usarlo?

- El llamado a servicios es complejo de realizar.
- Cuando una clase todavía no ha sido desarrollada.
- El resultado del llamado es siempre el mismo (mismos parámetros)