JUnit: Marco de Pruebas para Java

JUnit es un marco de pruebas popular para Java. Permite a los desarrolladores escribir y ejecutar pruebas automatizadas repetibles, lo que ayuda a asegurar que el código se comporte como se espera. Aquí tienes una guía básica sobre cómo usar JUnit para pruebas:

1. Configurar JUnit en tu Proyecto

- **Maven**: Añade la dependencia de JUnit a tu archivo `pom.xml`.

```
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

- **Gradle**: Añade la dependencia de JUnit a tu archivo `build.gradle`.

```
testImplementation 'junit:junit:4.13.2'
```

2. Escribir una Clase de Prueba

Crea una nueva clase Java para tus pruebas. Es una buena práctica nombrar tu clase de prueba después de la clase que estás probando, con un sufijo “Test”.

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class MyClassTest {

    @Test
    public void testAddition() {
        MyClass myClass = new MyClass();
        int result = myClass.add(2, 3);
        assertEquals(5, result);
    }
}
```

3. Anotaciones

- `@Test`: Indica que el método es un método de prueba.

- `@Before`: Se ejecuta antes de cada método de prueba. Útil para la configuración.
- `@After`: Se ejecuta después de cada método de prueba. Útil para la limpieza.
- `@BeforeClass`: Se ejecuta una vez antes de cualquier método de prueba en la clase.
- `@AfterClass`: Se ejecuta una vez después de todos los métodos de prueba en la clase.

4. Aserciones

Las aserciones se utilizan para verificar que el código se comporte como se espera.

- `assertEquals(expected, actual)`: Verifica que dos valores sean iguales.
- `assertTrue(condition)`: Verifica que una condición sea verdadera.
- `assertFalse(condition)`: Verifica que una condición sea falsa.
- `assertNull(object)`: Verifica que un objeto sea nulo.
- `assertNotNull(object)`: Verifica que un objeto no sea nulo.

5. Ejecutar tus Pruebas

Puedes ejecutar tus pruebas utilizando las herramientas integradas de tu IDE o a través de la línea de comandos utilizando Maven o Gradle.

- **Maven**: `mvn test`
- **Gradle**: `gradle test`

Ejemplo

Aquí tienes un ejemplo simple de una clase y su clase de prueba correspondiente:

```java
// MyClass.java
public class MyClass {
    public int add(int a, int b) {
        return a + b;
    }
}

// MyClassTest.java
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class MyClassTest {
```

```
    @Test
    public void testAddition() {
        MyClass myClass = new MyClass();
        int result = myClass.add(2, 3);
        assertEquals(5, result);
    }
}
```

Consejos Adicionales

- **Independencia de las Pruebas**: Cada prueba debe ser independiente de las demás. Evita dependencias entre pruebas.
- **Cubrimiento de Pruebas**: Aspira a un alto cubrimiento de pruebas, pero enfócate en pruebas significativas en lugar de solo aumentar el porcentaje de cubrimiento.
- **Simulación**: Utiliza marcos de simulación como Mockito para aislar la clase bajo prueba de sus dependencias.

Siguiendo estos pasos, puedes usar JUnit de manera efectiva para escribir y ejecutar pruebas para tus aplicaciones Java.