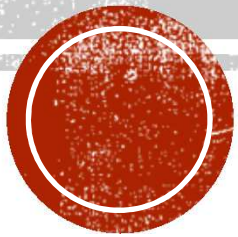
JUNIT

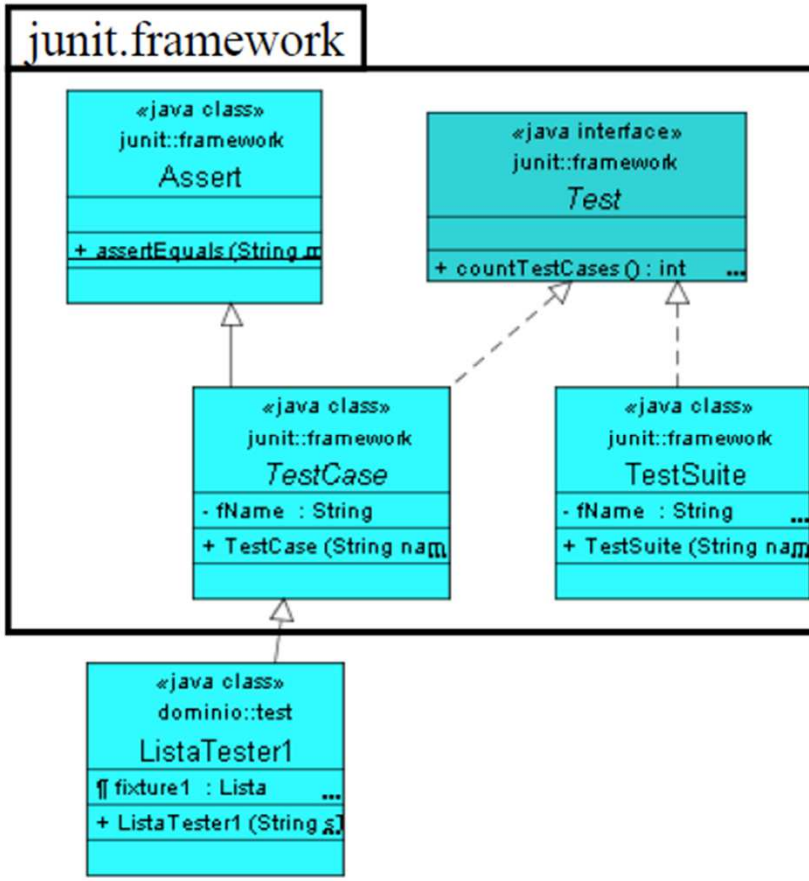


FRAMEWORK JUnit

- JUnit es un "framework" para automatizar las pruebas de programas Java
  - Escrito por Erich Gamma y Kent Beck
  - Open Source, disponible en http://www.junit.org
- Adecuado para el Desarrollo dirigido por las pruebas (Test-driven development)
- Consta de un conjunto de clases que el programador puede utilizar para construir sus casos de prueba y ejecutarlos automàticament.
- Los casos de prueba son realmente programas Java. Quedan archivados y pueden ser reejecutados tantas veces como sea necesario.
- Nos permite construir "árboles de casos de prueba" (suites)
- API: http://junit.sourceforge.net/javadoc/org/junit/Assert.html

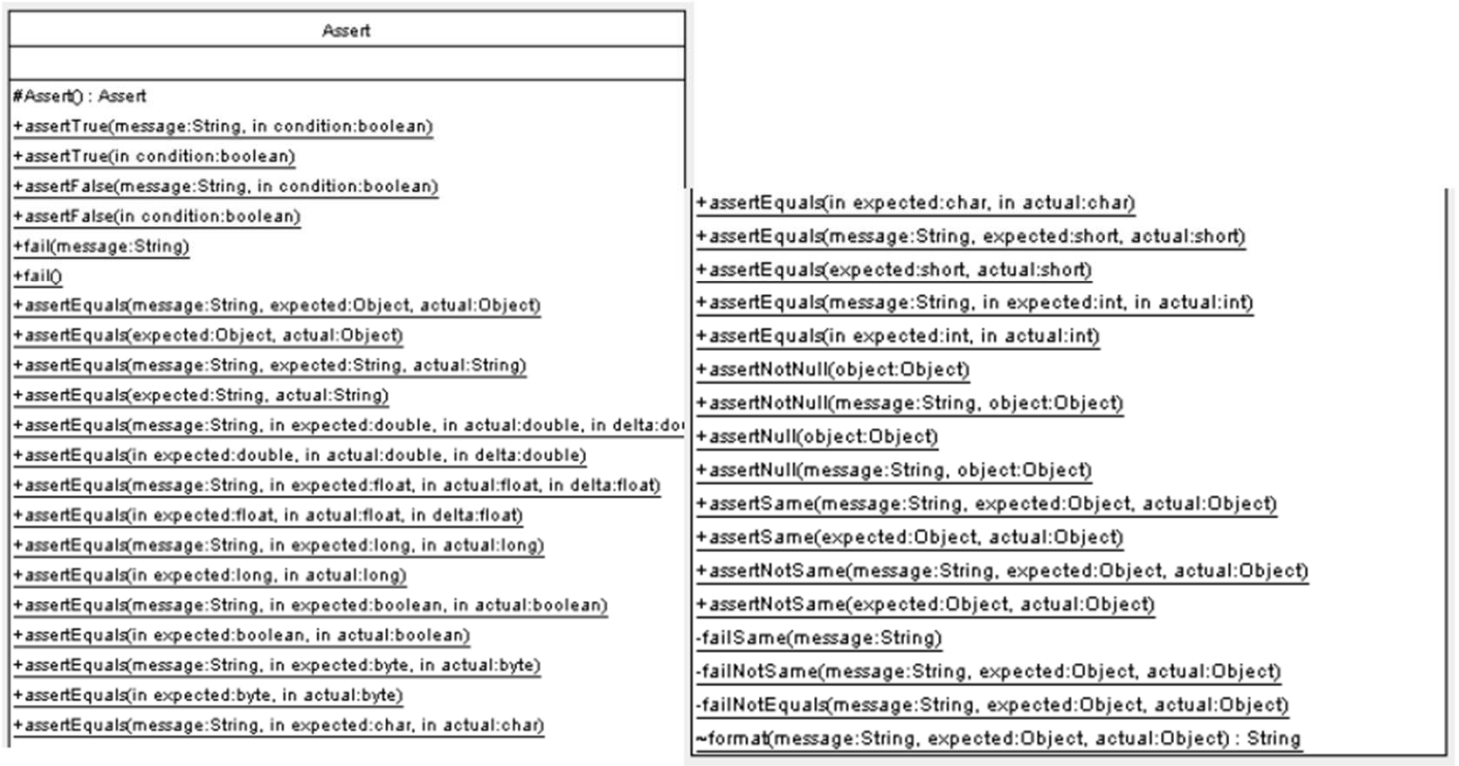
CLASES FUNDAMENTALES

- ¿Dónde está el código anterior?
- En una clase ListaTester , creada exprofeso para realizar las pruebas de Lista
- ListaTester especializa a la clase TestCase definida en JUnit
- En TestCase está definido el método assertEquals antes mencionado, y muchos otros más



Assert

Clases fundamentales: Assert



EJEMPLO

- Lo normal es hacer una clase de prueba por cada clase a probar o bien, una clase de prueba por cada conjunto de pruebas que esté relacionado de alguna manera.
- Tenemos dos clases; *Suma* y *Resta*, así que haremos dos clases de prueba *TestSuma* y *TestResta*. No es necesario llamar a estas clases con Test***, pero sí es conveniente seguir algún tipo de criterio.
- Nos ayuda a identificar rápidamente las clases que son de *test* automático

EJEMPLO:

- Se usan anotaciones de java en vez de tener que heredar de determinadas clases o cumplir determinada convención de nombres.
  - Ahora hay que ponerles una anotación *@Test*.

```
@Test
public void aVerSiSumaBien() {
    ...
}
```

- Cualquier clase puede ser de *test*.
- No hay ninguna obligación en el nombre de la clase de *test*, pero si es conveniente que nosotros sigamos algún tipo de nomenclatura, como hacer que todas las clases de *test* empiecen por *Test****.
- Los métodos son métodos estáticos de la clase *org.junit.Assert*, por lo que podemos importar dicha clase y usar sus métodos:

```
import org.junit.Assert
...
Assert.assertEquals("1+1 deberian ser 2", 2.0, resultado, 1e-6);
```

- o bien podemos hacer un *static import* para usar los métodos directamente

```
import static org.junit.Assert.*;
...
assertEquals("1+1 deberian ser 2", 2.0, resultado, 1e-6);
```

EJEMPLO:

- Para la *TestSuite*, nos basta una clase cualquiera, incluso aunque no tenga métodos. Basta con anotarla con *@RunWith(Suite.class)* para que *JUnit* sepa que debe ejecutar esa clase como una *TestSuite*, y anotarla también con *@SuiteClasses({ TestResta.class, TestSuma.class })* pasando las clases de *test*. Es decir, algo como esto

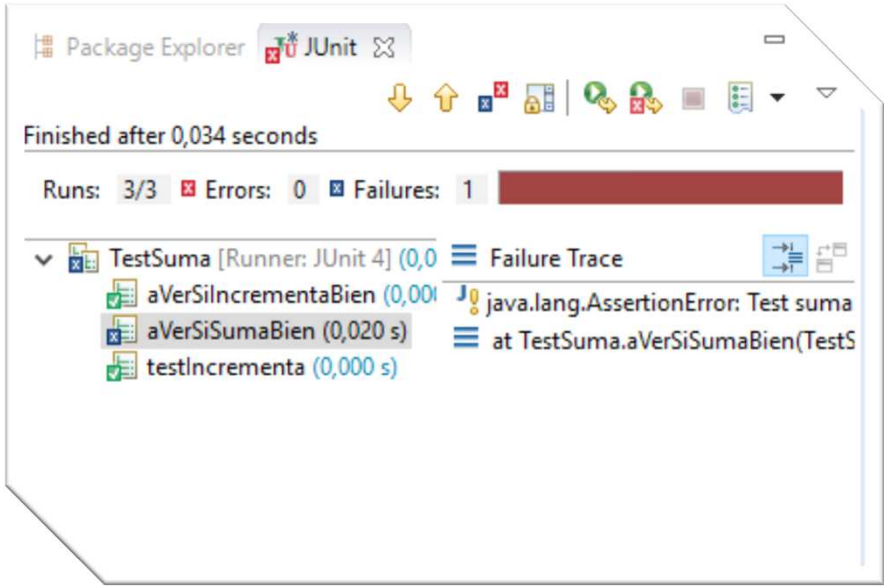
```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses( { TestResta.class, TestSuma.class })
public class AllTest {
}
```

EXEMPLE:

```
lass Suma {
ic double getSuma(double a, double b) {
// Se multiplica en vez de sumar a posta, para que los test fallen
return a * b;


ic double incrementa(double a) {
return a + 1;
```




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

import org.junit.Before;
import org.junit.Test;

public class TestSuma {
   private Suma suma;

    @Test
    public void testIncrementa() {
           double resultado = suma.incrementa(1.0);
           assertEquals("Al incrementar 1 deberia dar 2", 2.0, resultado, 1
    }
    @Test
    public void aVerSiIncrementaBien() {
            assertEquals("Test incrementa", 2.0, suma.incrementa(1.0), 1e-6
    }

    @Test
     public void aVerSiSumaBien() {
           assertEquals("Test suma", 2.0, suma.getSuma(1.0, 1.0), 1e-6);
     }

    @Before
     public void paraEjecutarAntes() throws Exception {
           suma = new Suma();
     }

}
```

EXEMPLO 2:

```
public class Suma {

   private int num1;

  private int num2;

  public Suma(int n1, int n2) {
     num1 = n1;
     num2 = n2;
  }
  public int sumar() {
     int resultado = num1 + num2;
     return resultado;
  }
}
```

Lo siguiente es crear la clase que nos servirá para probar la clase Suma. Queremos saber si la suma se hace correctamente en tres casos: sumando dos números positivos, sumando dos números negativos y sumando un número positivo y un número negativo. El código será el siguiente:

```
public class SumaTest {

  @Test
  public void sumaPositivos() {
     System.out.println("Sumando dos números positivos ...");
      Suma S = new Suma(2, 3);
      assertTrue(S.sumar() == 5);
  }

  @Test
  public void sumaNegativos() {
      System.out.println("Sumando dos números negativos ...");
      Suma S = new Suma(-2, -3);
      assertTrue(S.sumar() == -5);
  }

  @Test
  public void sumaPositivoNegativo() {
      System.out.println("Sumando un número positivo y un número negativo ...");
      Suma S = new Suma(2, -3);
      assertTrue(S.sumar() == -1);
  }
}
```

MÉTODOS JUNIT:

Método assertxxx() de JUnit	Qué comprueba
ssertTrue(expresión)	comprueba que expresión evalúe a true
ssertFalse(expresión)	comprueba que expresión evalúe a false
ssertEquals(esperado,real)	comprueba que esperado sea igual a real
ssertNull(objeto)	comprueba que objeto sea null
ssertNotNull(objeto)	comprueba que objeto no sea null
ssertSame(objeto_esperado,objeto_real)	comprueba que objeto_esperado y objeto_real sean el mismo objeto
ssertNotSame(objeto_esperado,objeto_real)	comprueba que objeto_esperado no sea el mismo objeto que objeto_rea
ail()	hace que el test termine con fallo

ANOTACIONS

Anotación	Comportamiento
`@Before`	El método se ejecutará antes de cada prueba (antes de ejecutar cada uno de los método marcados con `@Test`). Será útil para inicializar los datos de entrada y de salida esperada que se vayan a utilizar en las pruebas.
`@After`	Se ejecuta después de cada *test*. Nos servirá para liberar recursos que se hubiesen inicializado en el método marcado con `@Before`.
`@BeforeClass`	Se ejecuta una sola vez antes de ejecutar todos los *tests* de la clase. Se utilizarán para crear estructuras de datos y componentes que vayan a ser necesarios para todas las pruebas. Los métodos marcados con esta anotación deben ser estáticos.
`@AfterClass`	Se ejecuta una única vez después de todos los *tests* de la clase. Nos servirá para liberar los recursos inicializados en el método marcado con `@BeforeClass`, y al igual que este último, sólo se puede aplicar a métodos estáticos.