Técnicas de Diseño
75.10

Trabajo Práctico Nº2:

Framework para tests unitarios

Grupo:

Emiliano Suárez
Federico Rodriguez
Leandro Miguenz

Introducción

El framework FWK provee un conjunto de funciones que permiten al usuario escribir pruebas unitarias y obtener sus resultados de forma automática. Está construido sin utilizar reflexión, RTTI ni anotaciones, por lo que, a diferencia de otros frameworks de pruebas, no se requiere seguir convenciones de nombres en los tests ni marcarlos de alguna forma, sino que el usuario debe explicitar qué métodos en particular desea correr como pruebas, listándolos dentro de un main para su posterior ejecución por parte del framework.

Modo de uso

En primer lugar, la clase del usuario en donde están contenidos los métodos de prueba debe extender la clase Test del framework. Esto le brinda acceso a los métodos de comparación de valores (Assert...) definidos en dicha clase, que le brindan al framework la información necesaria para poder mostrar los resultados de las pruebas.

Concretamente, la clase de pruebas del usuario debe:
- importar la clase cuyo comportamiento se quiere probar
- importar la clase Test
- extender la clase Test
- tener un método main donde se invoquen (directa o indirectamente) todos los tests que quiera ejecutar el usuario.
- contener los tests definidos por el usuario (deben ser métodos estáticos, ya que el main es estático)

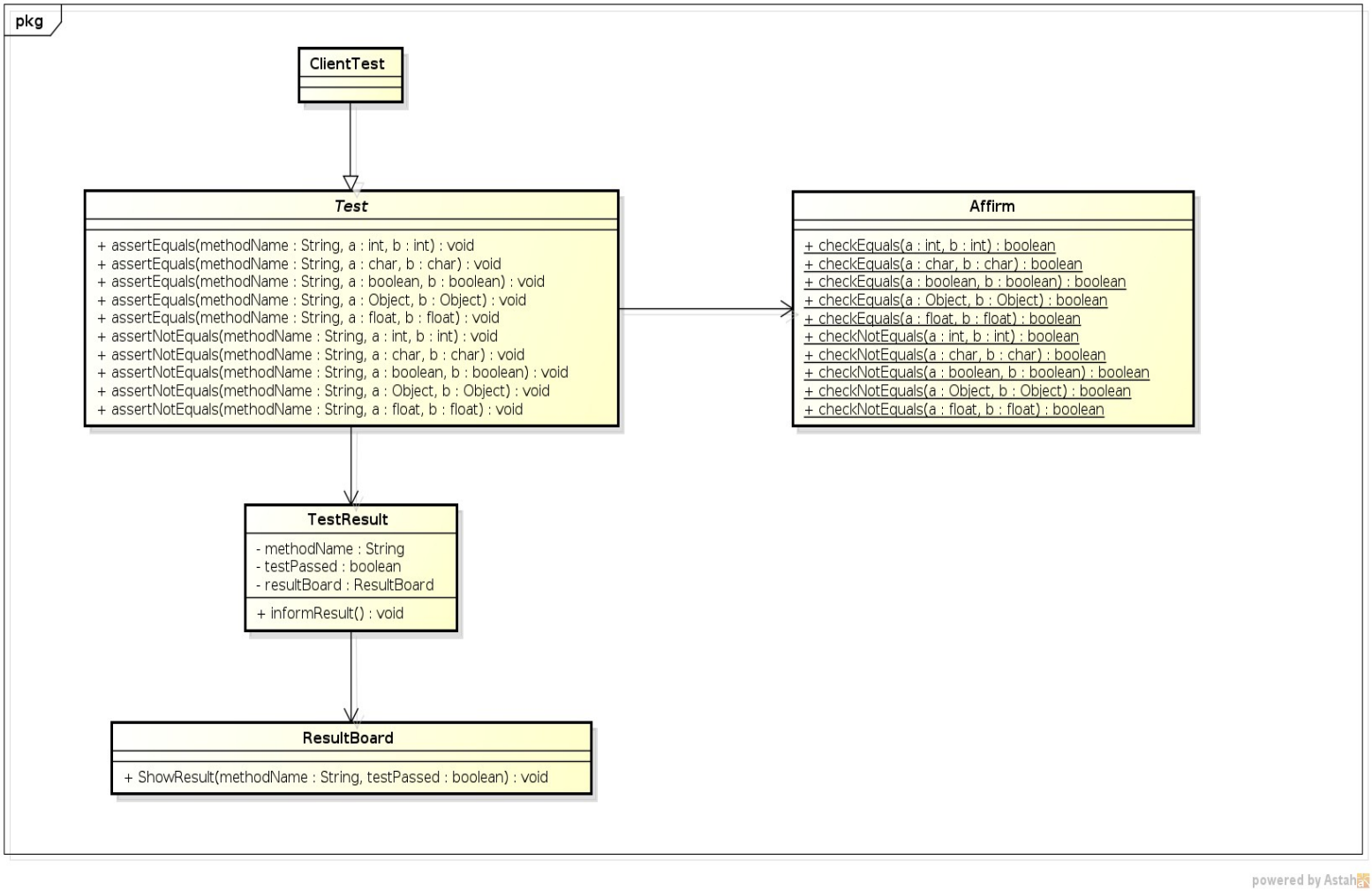
Un ejemplo de la estructura de la clase podría ser el siguiente:

```
import main.java.ClaseASerTesteada;
import main.java.Test;

public class ClaseASerTesteadaTest extends Test {

    // El main es el "disparador" de la corrida.
    public static void main(String[] args) throws Exception {
        run();
    }

    // El método run no es necesario, podrían ponerse los tests
directamente en el main.
    public static void run() {
        miTest();
    }

    // Declaracion del test
    public static void miTest() {
        assertEquals("miTest", valor1, valor2);
    }
}
```

Los métodos assertEquals/assertNotEquals comparan los valores recibidos en el segundo y tercer parámetro, y en base al resultado el framework determina si el test pasó con éxito o no. Para poder identificar el test que pasó o falló, el usuario debería pasar un mensaje adecuado como primer parámetro del assert (el nombre del test por ejemplo).

En caso de necesitar un setUp() o un tearDown() (no necesariamente deben llamarse así), el usuario puede definirlos e invocarlos desde run(), por ejemplo. Sin embargo, cabe aclarar que, a diferencia de lo que pasa en otros frameworks como jUnit, el setUp no se ejecuta automáticamente antes de cada test. Por esta razón el usuario debe prestar atención al orden en el que el main ejecuta los tests, o ejecutar el setUp antes de cada uno, para asegurarse de que cada prueba corra sobre el mismo escenario.

Estructura del framework



Como se mencionó antes, el punto de conexión entre el codigo cliente y el framework es la clase Test, de la cual la clase con las pruebas del usuario debe heredar para poder usar los métodos de comparación definidos en dicha clase.

Los métodos básicos de comparación están definidos en la clase Affirm. Lo que hace la clase Test es tomar esos métodos y envolverlos en otros, que a la comparación de los valores le agregan la funcionalidad provista por la clase TestResult para registrar los resultados de la prueba, y mostrarlos a través de ResultBoard.

(Ver el javadoc adjunto para información específica sobre los métodos de cada clase).