Técnicas de Diseño
(75.10)

Trabajo Práctico
Nº 2

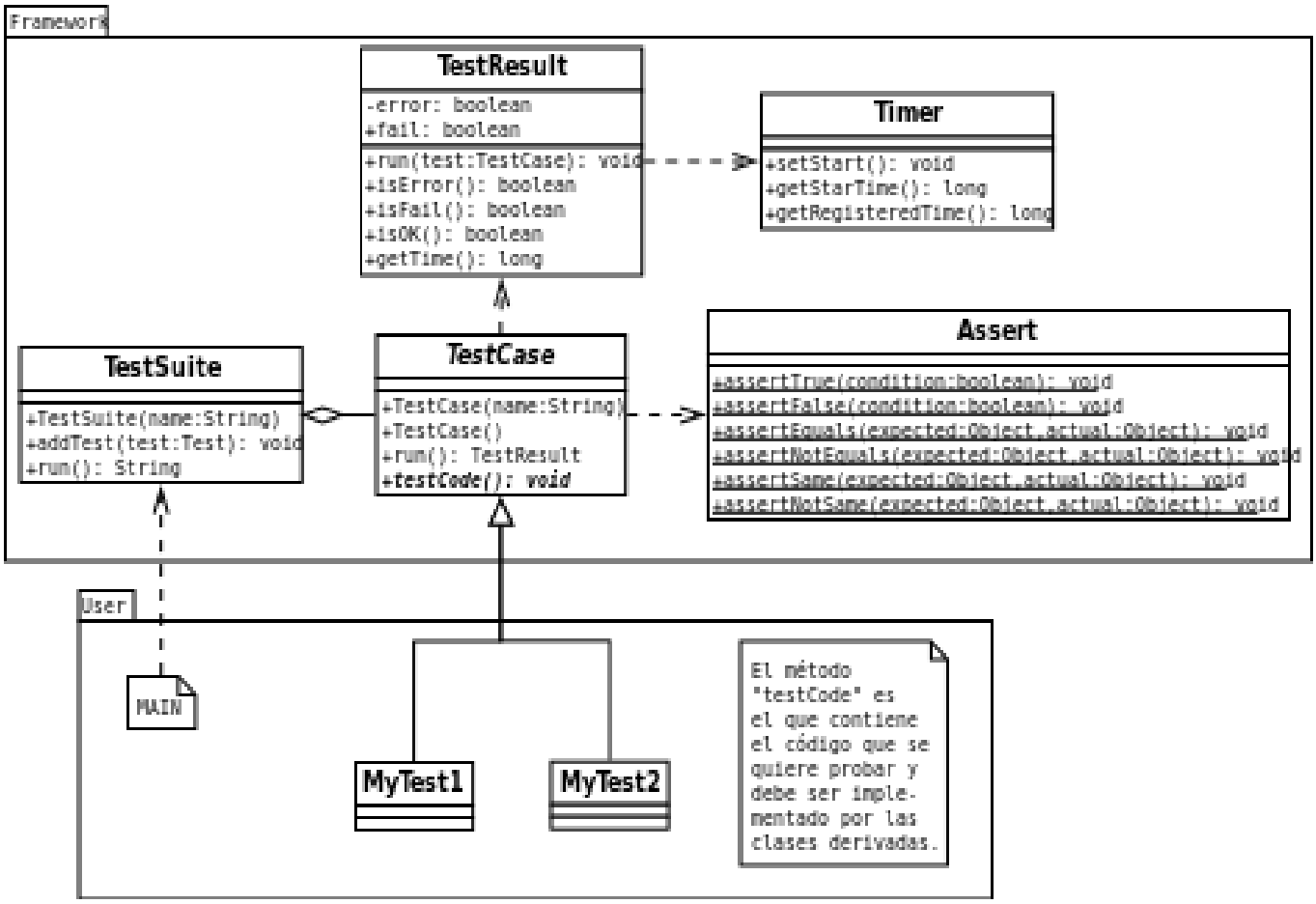
Grupo 5

Nombre y apellido	Padrón	Correo electrónico
Ezequiel Perez Dittler	91135	ezeperez26@gmail.com
Diego Montoya	91939	diegormontoya@gmail.com
Pablo Musumeci	92165	pablomusumeci@yahoo.com.ar

2º Cuatrimestre 2013

TP2

Diagrama



Diseño

Al no poder utilizar reflection ni anotations, el diseño de nuestro TestCase solo puede probar el contenido de un único método, llamado *testCode*. El usuario debe heredar de dicha clase, por cada test que quiera realizar e implementar su código de prueba en el método *testCode*. La mayor crítica que nosotros mismos tenemos para con nuestro diseño, es que implica tener que crear una clase por cada método que se quiera probar.

El framework incluye una clase estática Assert, cuya funcionalidad consiste en proveer al código cliente una manera de realizar sus comprobaciones dentro de los tests. Se decidió que

esta clase sea estática debido a que no se requiere una instancia para su utilización.

En un principio se había pensado utilizar el patrón Composite, para poder armar una jerarquía de TestCases y TestSuites, donde podían coexistir Suites dentro de otras Suites, que a su vez contenían Cases con el código a probar. Dicha idea fue desestimada debido a que, a nuestro juicio, las Suites sirven para agrupar pruebas que se encuentran relacionadas conceptualmente, ya que cada TestCase posee un único método de testing. Componer Suites no aportaba un beneficio considerable, y añadía una complejidad evitable a la hora de interpretar los resultados de los Tests.

Caso de Uso

Para hacer uso del framework, el cliente debe:

- Crear una clase que herede de TestCase, e implemente el método *testCode()* en el cual se encuentra el código que desea probar. Las comprobaciones que realice deberán ser implementadas con los métodos estáticos de la clase Assert provista por el framework.

- Crear (al menos una) instancia de la clase TestSuite, donde agrupará los tests según un criterio que el cliente elija.

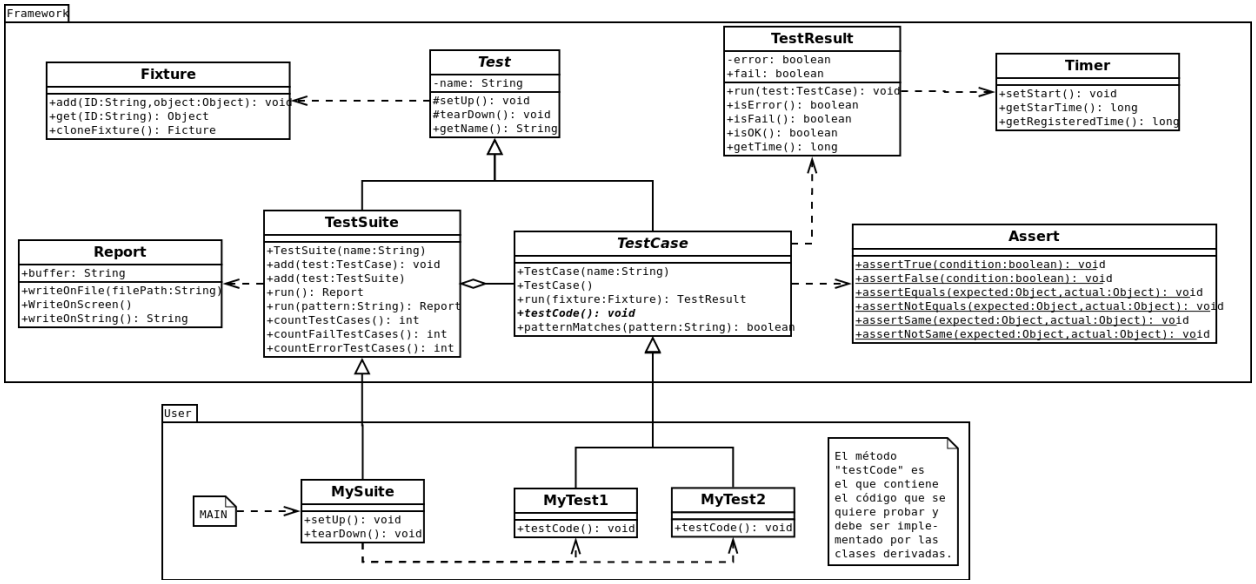
- Agregar los tests a la instancia de TestSuite por medio del método *add(TestCase)*

- Ejecutar el método *run()* de TestSuite. El mismo devolverá un String para que el usuario pueda decidir que hacer con él, ya sea mostrarlo por salida estándar, o guardarlo en un archivo (entre otras opciones posibles).

El código del proyecto provee a modo de ejemplo una clase *Calculator* que realiza operaciones básicas (sumar, restar, multiplicar y dividir) y los test de dicha calculadora realizados a partir del framework desarrollado.

TP 2.1

Diagrama



Diseño

En base los nuevos requerimientos del enunciado, se tuvieron las siguientes consideraciones:

- Como TestSuite y TestCase debe permitir ejecutar algo antes (setUp) y después (tearDown) de cada test, se tomaron estas funcionalidades comunes para definir una clase base, cuya implementación se refleja en la clase **Test**.

- En la propuesta inicial del framework, el TestSuite ofrecía las herramientas necesarias para que el desarrollador pueda utilizarlo sin necesidad de extender la clase. Como ahora se debe ofrecer la posibilidad de que un TestSuite pueda ejecutar una porción de código antes y después de correr la suite, el desarrollador debe extender de TestSuite y redefinir los métodos SetUp y TearDown.

- Si el desarrollador desea crear variables en el setUp para utilizar en el TestCase o todos los test que contiene un TestSuite, se provee la herramienta Fixture, donde se deben agregar las mismas. El test ya provee el fixture como un atributo que ya puede ser manipulado para tal uso.

Se agrega un atributo de de la siguiente manera:

```
public void setUp() {
	fixture.add("myNumber", 4);
	fixture.add("myString", "This is a test String");
 }
```

Se obtiene un atributo del fixture de la siguiente manera:

```
public void testCode() throws AssertException {
	int number = (int) fixture.get("myNumber");
	String string1 = (String) fixture.get("myString");
}
```

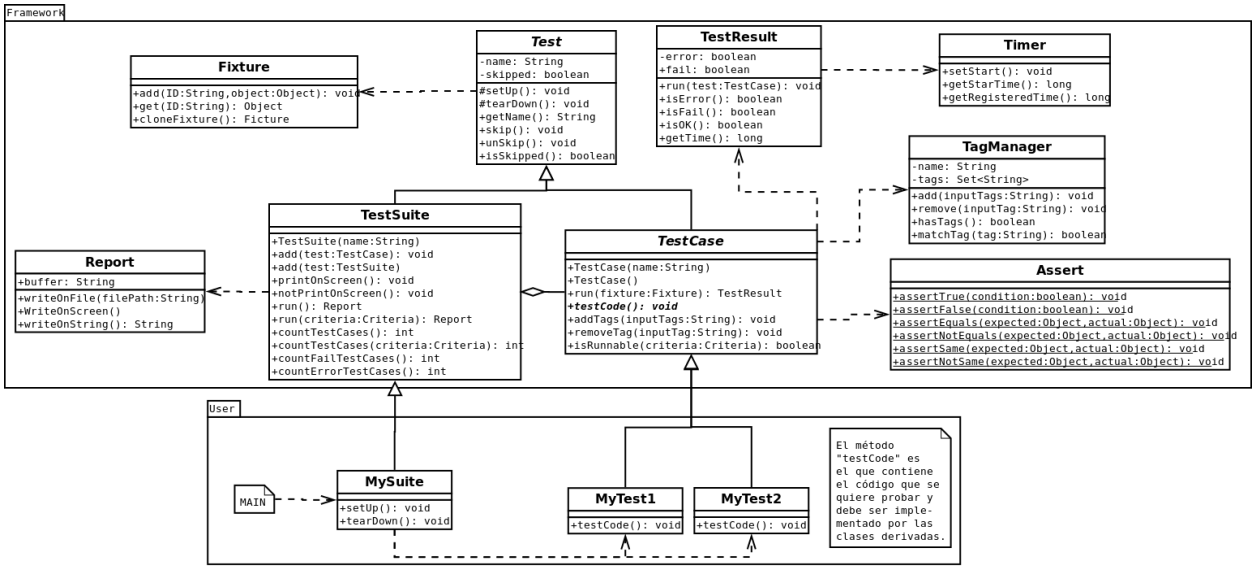
- La clase TestSuite deja de retornar un String en su método *run()*, para devolver un objeto del tipo **Report**, el cual contiene el resultado de los tests ejecutados, y tiene métodos que permiten:

    - *WriteOnScreen()*: Imprime el contenido del Report en pantalla.
    - *WriteOnFile(filePath)*: Guarda el contenido en el archivo indicado.
    - *WriteOnString()*: Devuelve un string con el contenido.

- Se permite la ejecución selectiva de TestCases, basándonos en el criterio de si cumple o no con una expresión regular provista por el código cliente. Para no obligar al cliente a tener que obligatoriamente utilizar una expresión regular, se provee un método *run()* sin parámetros, que ejecuta todos los TestCases contenidos dentro de un TestSuite.

TP 2.2

Diagrama general



(Se puede ver mejor la imagen en el [repositorio](#))

Skip de un test

Se agregó a la clase abstracta **Test**, una variable que indica si debe ser omitido o no. Dicha idea se debe a que se podría pensar en omitir un case simplemente, o toda una suite.
Su modo de uso es el siguiente:

```
Test testCase = new MyTestCase();
testCase.skip();      // Skip the test
testCase.unSkip();    // Unskip the test
if (isSkipped()) {
      System.out.println("The test is skipped");
}
```

Gestión de etiquetas

En el TP2.2 se solicita la inclusión de etiquetas (tags) y la posibilidad de poder ejecutar solo los tests que cumplan con alguna o todas las etiquetas dadas, además de expresiones regulares que se ya se maneja. La administración de etiquetas se realiza con la clase **TagManager**, que es incluido en los TestCases para que éstos gestionen sus etiquetas. El TagManager es transparente al desarrollador cliente. Si éste desea agregar o eliminar etiquetas a un TestCase, debe hacerlo de la siguiente manera:

```
TestCase test = new MyTestCase();
test.addTags("SLOW:FAST"); // Add multiple tags.
```

```
test.removeTag("SLOW");          // Remove a tag.
```

El método addTags permite recibir varios tags con el separador ":". El método removeTag permite eliminar de a un solo tag.

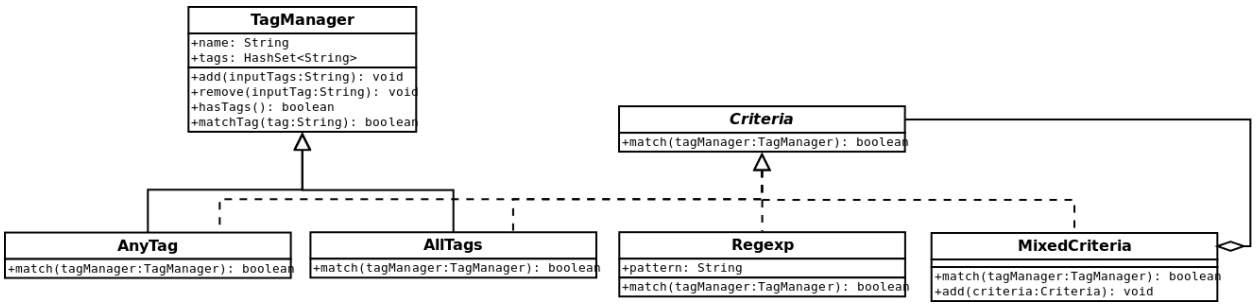
Solo se pueden etiquetar TestCase.

Ejecución selectiva de Tests

Cuando se desean ejecutar selectivamente algunos tests, pueden darse diversas variantes, como por ejemplo

- Ejecutar los tests que posean TODAS las etiquetas dadas
- Ejecutar los tests que posean ALGUNA de las etiquetas dadas
- Ejecutar los tests que posean TODAS las etiquetas dadas y cumplan con cierta expresión regular
- Ejecutar los tests que posean ALGUNA las etiquetas dadas y cumplan con cierta expresión regular

Como pueden ocurrir más variantes, se definió la interfaz *Criteria*, donde se ve aplicado el patrón **Command**, ya que se pide a quienes implementan dicha interface que respondan si un *TagManager* cumple con el criterio dado.



(Se puede ver mejor la imagen en el repositorio)

Como se puede requerir que un tests cumpla un criterio con un conjunto de etiquetas, las clases AllTags y AnyTag extienden de TagManager para poder manipularlo como si de un TagManager se tratase.

La funcionalidad de ejecución selectiva mediante una expresión regular, se adaptó a este nuevo diseño, implementando la interface Criteria en la clase Regexp. Esta clase no hereda de TagManager porque no precisa tener conocimiento sobre las etiquetas ni gestionarlas, pero puede decir si un *TagManager* cumple con una expresión regular analizando su nombre (que será el nombre que lleva el TestCase que lo contiene).

Por último, para poder generar todas las variantes de selección de tests, se hizo uso del patrón **Composite** en la clase *MixedCriteria*, quien representa un contenedor de diferentes criterios, dice si un *TagManager* cumple con todos los criterios que en ella se guardan. Por ejemplo, el criterio “ejecutar los tests que posean ALGUNA las etiquetas dadas y cumplan con cierta expresión regular” se forma con un MixedCriteria al que se le añaden un AnyTag y Regexp.

Implementación de Runner

Debido a todas las variantes de etiquetas y expresiones regulares que pueden aplicarse como filtro para ejecutar selectivamente a los tests, se diseño la clase Runner por lo siguientes motivos:

- No sobrecargar a TestSuite para contemplar todas las variantes de filtros
- Ocultar la implementación de Criteria al desarrollador cliente

Al crear un Runner, se le debe pasar por parámetro en el constructor el TestSuite que se desea correr. Las variantes de ejecuciones que permite Runner son:

- *runAll()* corre todos los tests, sin aplicar filtro alguno.
- *runWithAnyTags(String tags)* corre los tests que contiene alguno de los tags recibidos como parámetro. Se pueden establecer varios tags separandolos con el caracter “:”.
- *runWithAllTags(String tags)* corre los tests que contiene todos los tags recibidos por parámetro. Se pueden establecer varios tags separandolos con el caracter “:”.
- *runWithRegexp(String regexp)* corre los test cuyo nombre cumple con la expresión regular especificada.
- *runWithAnyTagsAndRegexp(String tags, String regexp)* corre los tests que contiene alguno de los tags recibidos como parámetro y, además, cuyo nombre cumple con la expresión regular especificada.
- *runWithAllTagsAndRegexp(String tags, String regexp)* corre los tests que contiene todos los tags recibidos como parámetro y, además, cuyo nombre cumple con la expresión regular especificada.

En todos los casos, el los métodos del Runner devuelve el Report usado en la entrega anterior que contiene el reporte generado.

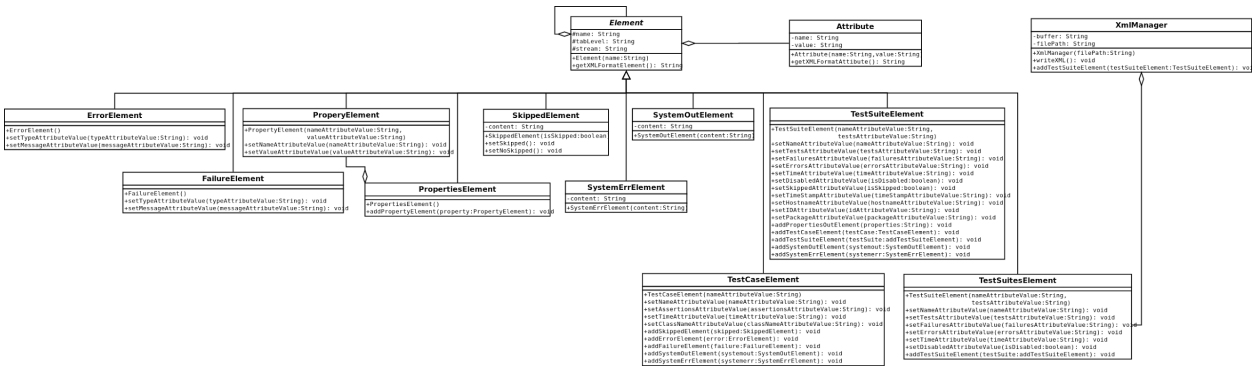
Reporte textual por línea de comando progresivo en tiempo real

El reporte textual que se hacía previamente no era progresivo en tiempo real, ya que recopilaba la información luego de haber ejecutado los tests para generar el reporte. Además, la funcionalidad de generación de reporte estaba contenida dentro de TestSuite y una parte en TestResult.
Debido a que el nuevo requerimiento añadiría complejidad al código TestSuite, se delegó la responsabilidad de impresión de resultados a la clase Printer, para simplificar el código y

Además, se permite que la impresión sea opcional. Antes de correr un TestSuite se debe ejecutar el método *printOnScreen* si desea imprimir el reporte o *notPrintOnScreen* si no se desea imprimir el reporte. El valor por defecto es imprimir el reporte.

Reporte en formato XML

Para el reporte en formato XML se diseñaron un conjunto de clases que se encuentran en el paquete *xml*.



(Se puede ver mejor la imagen en el repositorio)

Su uso está encapsulado en el Runner, donde se debe especificar la ruta del archivo XML al momento de crearlo y se puede generar el archivo al ejecutar el método *writeXML()*.