**TECNICAS DE DISEÑO (75.10) – FIUBA**

**TRABAJO PRÁCTICO N° 2**

**INTEGRANTES**

|  |  |  |  |
| --- | --- | --- | --- |
| **PADRON** | **NOMBRE Y APELLIDO** |  | **MAIL** |
| **89630** | **Martín Andrés Lucero** |  | [**luceromartinandres89630@gmail.com**](mailto:l%75c%65rom%61%72%74%69%6e%61%6ed%72%65s%38%39%36%330@%67ma%69l%2e%63o%6d) |
| **85828** | **Natalia Merino** |  | **nati.merino@gmail.com** |
| **84441** | **Leandro Pellegrino** |  | **lrpellegrino@gmail.com** |

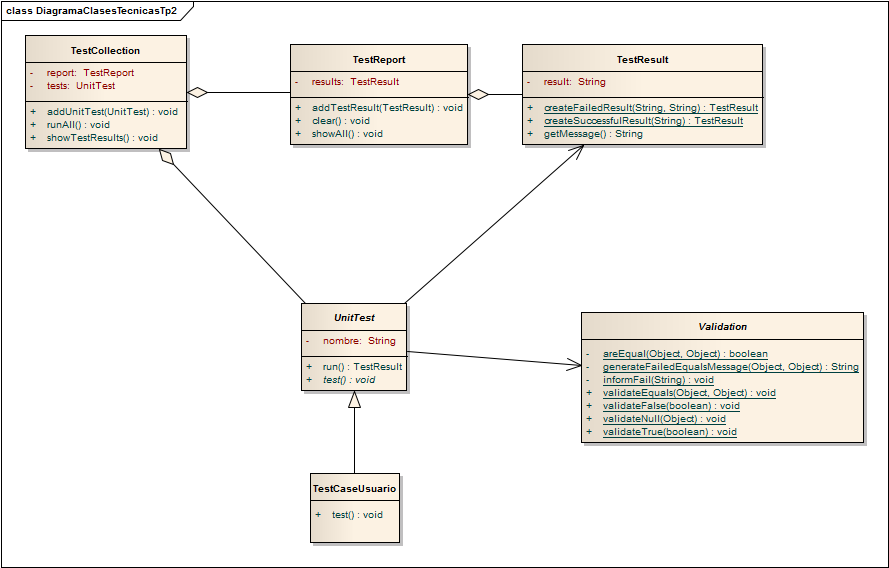
**Índice**

Diagrama de Clases 2

Diseño del Framework 2

Falencias del Diseño 3

# Diagrama de Clases



# \*La clase TestCaseUsuario no forma parte del framework. Su inclusión es a modo de ilustración

# Diseño del Framework

Se comenzó por crear una relación contenedor – contenido entre TestCollection y UnitTest, teniendo TestCollection métodos para agregar, correr todos y mostrar el resultado de los tests contenidos. Similarmente se creó otra relación contenedor – contenido entre TestReport y TestResult para almacenar resultados de un conjunto de tests, y a TestReport se lo compuso con TestCollection para que esté último se abstraiga de la forma en que se almacenan y muestran los resultados.

Luego se implementó la clase utilitaria Validation, con todos métodos estáticos, de forma similar a la implementación de Assert de junit. La idea fue que sea una clase abstracta con todos métodos estáticos ya que no tiene estado (solo comportamiento -> clase utilitaria).

El problema principal a resolver fue cómo correr un método creado por el cliente (su método que hace el testing de algo). Si el lenguaje permitiese pasar métodos como argumentos, el problema se resolvería pasándole dicho método a UnitTest para que lo ejecute y verifique si algún validation falló. Como esto no es posible en Java, y al no poder usar reflection, se pensó en utilizar la herencia.

Para hacer un test, el usuario debe heredar de UnitTest y redefinir el método test(), en donde escribe su código de prueba. El framework se encarga de ejecutarlo e informar el resultado.

# Falencias del Diseño

El principal problema del diseño es que el cliente debe crear tantas clases que hereden de UnitTest como tests quiera hacer. Esto es así por el modelo de dominio pensado. El uso de herencia se justifica por lo dicho en el apartado anterior, pero esto no implica que sea la mejor solución para este trabajo práctico.

Por otro lado, y pensando en los objetivos de la materia, parece ser que se podría implementar algún patrón de creación de instancias (factory, builder). No se encontró la forma de hacerlo para la creación de los tests del usuario, ya que a priori no se conocen los nombres que le pondrá el usuario a las clases de sus tests (no son conocidos los nombres ni la cantidad).

Finalmente se podría discutir la falta de interfaces en el diseño. Al tener pocas clases cuyas responsabilidades son pequeñas, no se vio la necesidad de armar interfaces. Siguiendo otra línea de pensamiento, si el framework se desea expandir en un futuro, surgirán más clases y se deberá hacer uso de interfaces.