Índice

Enunciado

75.10 Técnicas de diseño

Trabajo Práctico N° 2

Objetivo

El objetivo principal del TP es:

- Proveer un conjunto de clases en forma de framework lo suficientemente genéricas para que un usuario las pueda utilizar a fin de escribir test unitarios.
- Proveer distintos tipos de Asserts para ser utilizados en los tests.
- Generar un reporte con los resultados.

Restricciones

- Trabajo Práctico grupal implementado en java o C#
- Se deben utilizar las mismas herramientas que en el TP0 (git + maven + junit4 / git + VS 2012 + MS Test o NUnit).
- Todas las clases del sistema deben estar justificadas.
- Se debe modelar utilizando un modelo de dominio, y no usando herramientas tecnologicas como reflection, anotations, etc.
- Todas las clases deben llevar un comentario con las responsabilidades de la misma.
- El uso de herencia debe estar justificado. Se debe explicar claramente el porqué de su conveniencia por sobre otras opciones.
- Se debe tener una cobertura completa del código por tests
- Se pide ademas de tener los test unitarios junit/nunit, replicar tests utilizando el framework desarrollado en el TP.

* No se aceptaran TP's que violen alguna de las restricciones.

Criterios de Corrección

- Cumplimiento de las restricciones
- Documentación entregada
- Diseño del modelo
- Diseño del código
- Test Unitarios

Se tendrán en cuenta también la completitud del tp, la correctitud, distribución de responsabilidades, aplicación y uso de criterios y principios de buen diseño, buen uso del repositorio y uso de buenas prácticas en gral.

Calendario

Jue 17/10	Presentación del TP
Jue 31/10	Entrega TP, via campus

Desarrollo

Para llegar a la solución se decidió usar el patrón *Composite*. Test es la clase de la cual heredan el componente y el contenedor. El componente es un test específico, mientras que el contenedor es un grupo de tests. Es el único momento en el que se usa la herencia. Si bien no se utiliza ahora, esto permitirá agrupar distintos tipos de Tests luego. Por ejemplo si quisieran agruparse tests para un producto, podrían agruparse dentro de un contenedor que tenga productos. Luego este formará parte de otro contenedor general que tendrá éste y otros Tests, ya sean simples o compuestos.

Clases:

- Test: esta clase fue creada para poder aplicar el patrón *Composite*.

- TestCase: clase que representa cada "test individual", cada prueba en particular. Los *Assert* se deben utilizarse en esta clase.

- TestSuite: clase que contiene a los tests existentes. Además tiene un método para correrlos todos.

- Assertion: es la clase encargada de devolver el resultado de la comparación entre el/los operando/s recibido/s. El resultado de la operación lo arroja mediante excepciones definidas por nosotros.

- TestResult: clase que es contenedor del resultado de las pruebas. Se pasará un TestResult por parámetro al TestSuite, quien lo irá pasando a todos los tests que contiene para que lo completen. Posee 3 listas, en las cuales se guardarán los tests pasados (OK), fallados y que dieron error, según corresponda.

Nuevas Excepciones:

- ErrorException: excepción que implica que hubo un error al correr el proceso. Se arroja si los parámetros no fueron inicializados.

- OkException: excepción que se utiliza para indicar que la comparación de los operandos da un resultado positivo.

- FailureExcepcion: excepción utilizada para indicar que la comparación de los operandos dio un resultado negativo.

Diagrama de clases