Índice

Enunciado

75.10 Técnicas de diseño

Trabajo Práctico N° 2.2

Objetivo

- Taggear tests (un string, ejemplo: FAST, SLOW, DB, INTERNET, etc)
- Poder agregar más de un tag a un test.
- Permitir desactivar un test con SKIP.
- Poder ejecutar test pertenecientes a un tag particular.
- Poder ejecutar test pertenecientes a varios tags.
- Poder ejecutar test pertenecientes a varios tags y/o que su nombre de test case coincida con una expression regular y/o que su nombre de test suite coincida con una expresión regular.
- Tomar el tiempo a todos los test para poder agregarlos en reportes.
- Reporte textual por línea de comando. (Utilizar formato de la entrega anterior).
- Que el reporte textual por línea de comando anterior sea progresivo en real time (a medida que vaya teniendo resultados los muestra).
- Implementar un reporte xml siguiendo el siguiente XSD: https://svn.jenkins-ci.org/trunk/hudson/dtkit/dtkit-format/dtkit-junit-model/src/main/resources/com/thalesgroup/dtkit/junit/model/xsd/junit-4.xsd

Casos de Prueba de ejemplo

Caso	GIVEN	WHEN	THEN
Ejecutar <test case> con cierto tag de un <test suite>	<test case> llamado T1, T2 y T3 en un <test suite> TS1, donde T1 y T3 poseen un tag SLOW	se pide ejecutar los <test case> del <test suite> TS1 que poseen el tag SLOW	debería ejecutar los <test case> T1 y T3.
Ejecutar <test case> con cierto filtro de tag de un <test suite>	<test case> llamado T1, T2 y T3 en un <test suite> TS1, y los <test case> llamado T4, T5 y T6 en un <test suite> TS2, y los <test case> llamado T7, T8 y T9 en un <test suite> TS1.1 donde T1, T3, T5, T7 y T9 son etiquetados con un tag IMPARES y se skipea a T1	se pide ejecutar los <test case> del <test suite> TS1 que poseen el tag SLOW y cuyo nombre comience con T1	debería ejecutar los <test case> T3, T7 y T9.
Ejecutar <test case> con cierto tag de un <test suite> con skips	<test case> llamado T1, T2 y T3 en un <test suite> TS1 donde T1, T3 son etiquetados con un tag SLOW y se skipea a T1	se pide ejecutar los <test case> del <test suite> TS1 que poseen el tag SLOW	debería ejecutar los <test case> T3,
Ejecutar <test case> que pertenecen a alguno de una serie de tags.	T1, tag: SLOW, DB T2, tag: SLOW T3, tag: DB T4, tag: FAST T5, tag: SMOKE T6: tag: -	Se pide correr tests pertenecientes a tags: DB, FAST, SMOKE	Se deberían ejecutar: T1, T3, T4, T5
Ejecutar <test case> que coincidan en un tag y no comiencen con cierto nombre.	T1: tag: SLOW T2: tag: FAST T3: tag: SLOW T<no correr>: tag: SLOW	Se pide correr tests pertenecientes a tag SLOW y que no comiencen con nombre "no correr"	Se deberían ejecutar: T1, T3
Ejecutar <test case> que pertenezcan a alguno de una serie de tags y que ademas su nombre contenga cierta clave.	T1: tag: DB T2: tag: DB T3: tag: SLOW T<... mysql ....1>: tag: DB T<... mysql ....2>: tag: DB T<... mysql ....3>: tag: -	Se pide correr tests pertenecientes a tag DB y que ademas contengan en algun lugar de su nombre la palabra "mysql".	Se deberían ejecutar: T<... mysql ....1> T<... mysql ....2>
Ejecutar una serie de <test cases> y tomarles el tiempo	TS: T1, T2, T3 T1: tarda x T2: tarda y T3: tarda z	Se pide correr TS.	Se deberían ejecutar: T1, T2, T3 y obtener los tiempos de cada uno: T1: x (+/- un delta) T2: y (+/- un delta) T3: z (+/- un delta) TS: x+y+z (+/- un delta)

Generacion del Reporte en forma progresiva.	TS: T1, T2, T3 T1: tarda x T2: tarda y T3: tarda z	Se pide correr TS.	Se deberían ejecutar: T1, T2, T3 Se debería ver el resultado de: T1 antes de x+y T2 antes de x+y+z

Formato de salida del Reporte XML

Ver:
https://svn.jenkins-ci.org/trunk/hudson/dtkit/dtkit-format/dtkit-junit-model/src/main/resources/com/thalesgroup/dtkit/junit/model/xsd/junit-4.xsd

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 tecnológicas como reflection, annotations, 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.

* 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 07/11	Presentación del TP
Jue 14/11	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 abstracta es la clase padre de TestCase y TestSuite. La misma fue creada para poder aplicar el patrón Composite.

TestCase

Clase que representa cada “test individual”, o sea cada prueba en particular.

Para poder ejecutar correctamente "runTest", primero hay que setear los valores a comparar, con "setAssertValue".

TestSuite

Clase que contiene a los tests existentes. Puede contener tanto TestCases como otros TestSuites (grupos de tests).

Posee una familia de métodos *runTest* (con regex, sin regex, se le pasa un TestResult, no se le pasa un TestResult, etc.) para correr todos los TestCase contenidos en ella.

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.

Si el *assert* falla, retorna un FailureException. En el caso de que se produzca un error, se lanzara un error acorde a eso (propio de java).

TestResult

Clase que guarda el resultado de todos los TestCase corridos dentro de un TestSuite y otros TestResults de los TestSuite que contiene. Para obtener los valores, se la debe recorrer recursivamente.

Después de ordenar los resultados de los tests, puede mostrar los resultados tanto por consola como a través de un archivo de texto

TestAssertResult

Clase que se utiliza para guardar el resultado de correr un test. Los posibles valores que va a recibir son "Ok", "Fail" o "Error".

FailureExcepcion

Excepción utilizada para indicar que la comparación de los operandos dio un resultado negativo, o sea cuando el *assert* no es satisfactorio.

TestConditions

Clase que contiene todas las restricciones a la hora de correr un test (tanto para TestCase como TestSuite), como por ejemplo un RegEx para el nombre de TestCase ó TestSuite, como una serie de tags.

TestConditionsBuilder

Clase que se encarga de construir un *TestConditions*. Tiene valores por defecto de todos los atributos de *TestConditions*, entonces si el usuario no le pasa alguno, no hay problema.

Timer

Clase que se encarga de la lógica de tiempos. TestCase lo usa para contabilizar el tiempo que tarda en ejecutar su función principal (la que checkea el test).

Diagrama de clases