

Índice

Enunciado.....	2
Desarrollo.....	7
Clases	8
Test	8
TestCase	8
TestSuite.....	8
Assertion	8
TestResult	8
TestAssertResult	8
FailureExcepcion	9
Diagrama de clases.....	10

Enunciado

75.10 Técnicas de diseño

Trabajo Práctico N° 2.1

Objetivo

- Identificar los test por nombre.
- Identificar suites por nombre.
- Permitir ejecutar algo antes y después de cada test. (setup/teardown).
- Permitir ejecutar algo antes y después de cada test suite. (setup/teardown).
- Permitir componer test suites en test suits sin ninguna restricción en la cantidad de niveles.
- Poder acceder al fixture creado en el setup desde un test (tanto al fixture del test case, como de los test suites en los que viva el test case particular).
- Diferenciar entre Failures y Errors. (Failures lanzan AssertException o algo por el estilo, error el resto).
- Poder ejecutar solo los tests cuyo nombre coincida con una regular expression.
- Generar un archivo con un reporte textual de la ejecución de los tests.

75.10 Técnicas de diseño

Casos de Prueba de ejemplo

Caso	GIVEN	WHEN	THEN
Unicidad en los nombres de los <test case>	<test case> llamado A en un <test suite> B	creo otro <test case> llamado A en el <test suite> B	no me debería dejar crear el ultimo, ya que existe un <test case> con ese nombre.
Unicidad en los nombres de los <test suites>	<test suite> llamado A en un <test suite> B	creo otro <test suite> llamado A en el <test suite> B	no me debería dejar crear el ultimo, ya que existe un <test suite> con ese nombre.
setup en <test case>	<test case> con un <setup>	corre el <test case>	primero se corre <setup> y luego el <test case>
setup en <test suite>	<test suite> con un <setup> y dos <test cases>	corre el <test suite>	se corre: 1. <setup> 2. primer <test case> 3. <setup> 4. segundo <test case>
setup en <test suite> de <test suite>	<test suite> con un <setup> y dos <test suites> que tienen 1 <test case> cada uno y un <setup>.	corre el <test suite> de mas alto nivel.	se corre: 1. <setup> del <test suite> de más alto nivel. Luego para cada test suite: 1. <setup> del <test suite> 'hijo'. 2. <test case> del <test suite> hijo.
Acceso al fixture de un <test case> y <test suite>	<test suite> A dentro de <test suite> B. <test case> dentro de <test suite> A. <test case> hace uso de objeto definido en <setup> del <test suite> A, otro de <test suite B> y otro de su propio <setup>. Corroborado con asserts.	corre el <test suite> de mas alto nivel.	la ejecución en exitosa.
Failure en <test case>	<test case> que hace un assertTrue(false)	se corre el <test case>	El resultado de la ejecución es un <test case> failed.
Error en <test case>	<test case> que lanza una exception.	se corre el <test case>	El resultado de la ejecución es un <test case> con error.

75.10 Técnicas de diseño

Test por nombre	<test case> llamado "my special test case" <test case> llamado "my special test case 1" <test case> llamado "my special" <test case> llamado "a test"	se corre solo los <test case> que coinciden con el pattern: .*special.*	Se corren todos menos "a test case"
Test por nombre inexistente	<test case> llamado "my special test case" <test case> llamado "my special test case 1" <test case> llamado "my special" <test case> llamado "a test"	se corre solo los <test case> que coinciden con el pattern: .*no existe	No se corre ningun test.

Agregar los que crean convenientes para tener una cobertura completa.

Formato de salida del Reporte

<Test suite name>

[ok|error|fail] <Test case name>

[success|failure] Summary

=====

Run: <number of test cases>

Errors: <number of errors>

Failures: <number of failures>

Para los nombres de los tests suites de tests suites, concatenar con ':'

Si tengo un test suite: 'MyProject' que a su vez tiene un Test suite 'models' y que a su vez tiene un test suite 'unit', el nombre final del test suite que contiene a los test cases del de más abajo ('unit') debería quedar: 'MyProject.model.unit'

Ejemplo de reporte:

MyProject.model

[ok] a test

MyProject.model.unit

[ok] my simple test

[fail] another test

[error] a test with an error

MyProject.ia.logic

[ok] a test

[failure] Summary

=====

Run: 5

Errors: 1

Failures: 1

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
- Se pide además de tener los test unitarios junit/nunit.

* 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 31/10	Presentación del TP
Jue 07/11	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.

```

classDiagram
    pkg --> Assertions
    pkg --> FailureException
    pkg --> Test
    pkg --> TestSuite
    pkg --> TestCase
    pkg --> TestResult

    class pkg {
        ~tests: List<Test>
        ~testResults: List<TestResult>
        ~name: String
        ~reportPath: String
        +TestResult()
        +TestResult(aName: String)
        +addTestResult(testResultName: String) TestResult
        +addFailure(test: Test) void
        +addError(test: Test) void
        +addPassed(test: Test) void
        +selfReportPath(path: String) void
        +getIsPassed() List<Test>
        +fillIsPassed(list: List<Test>, result: String, testResult: TestResult) void
        +getIsFailure() List<Test>
        +getIsError() List<Test>
        +showResults() void
        +showList(list: List<Test>) void
        +showReport() void
        +showTestResultData() void
        +printTestsList() void
        +printTestResults() void
        +writeln(line: String) void
        +getResult(testName: String) String
    }

    class Test {
        #testCaseName: String
        #testType: String
        #fixtures: Map<String, Object>
        +runTest(result: TestResult) void
        +runRegEx(result: TestResult, regEx: String) void
        +setUp() void
        +tearDown() void
        +getName() String
        +setName(aName: String) void
        +repeatedTest(anotherTest: Test) boolean
        +addAFixtureItem(name: String, value: Object) void
        +getAFixtureItem(name: String) Object
        +setUpVariablesFromSuite(fixtures: Map<String, Object>) void
    }

    class TestSuite {
        -tests: Vector<Test> = new Vector<Test>()
        +TestSuite(name: String)
        +TestSuite()
        +runRegEx(result: TestResult, regEx: String) void
        +runTest(result: TestResult) void
        +runTest() TestResult
        +addTest(test: Test) void
        +setUp() void
        +tearDown() void
    }

    class TestCase {
        -testValue1: Object
        -testValue2: Object
        -valuesQuantity: int
        -valuesAreSet: boolean
        +TestCase(name: String)
        +getAFixtureItemFromSuite(name: String) Object
        +setAssertValue(aTestValue1: Object, aTestValue2: Object) void
        +setAssertValue(aTestValue: Object) void
        +runRegEx(result: TestResult, regEx: String) void
        +runTest(result: TestResult) void
        +testBody() void
        +setUp() void
        +tearDown() void
    }

    class FailureException {
        -serialVersionUID: long = -9133763983524959093L
        +FailureException(message: String)
        +FailureException()
    }

    class TestResult {
        -test: Test
        -result: String
        +TestAssertResult(aTest: Test, resultName: String)
        +getTest() Test
        +getResult() String
    }

    pkg --|> Assertions
    pkg --|> FailureException
    pkg --|> Test
    pkg --|> TestSuite
    pkg --|> TestCase
    pkg --|> TestResult

    Test --|> TestSuite
    Test --|> TestCase
  
```