Universidad de Buenos Aires
Facultad De Ingeniería
Año 2013 - 2do Cuatrimestre

Técnicas de Diseño (75.10)

TRABAJO PRÁCTICO N°2
FECHA: 31/10/2013

INTEGRANTES:	PADRON
Franetovich, Damian <damian168@gmail.com>	88924
Rial, Sebastián <riseba@gmail.com>	90309
Pivetta, Agustín <aguspivetta@gmail.com>	90789

Índice

1. Enunciado

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.

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 ademas de tener los test unitarios junit/nunit.

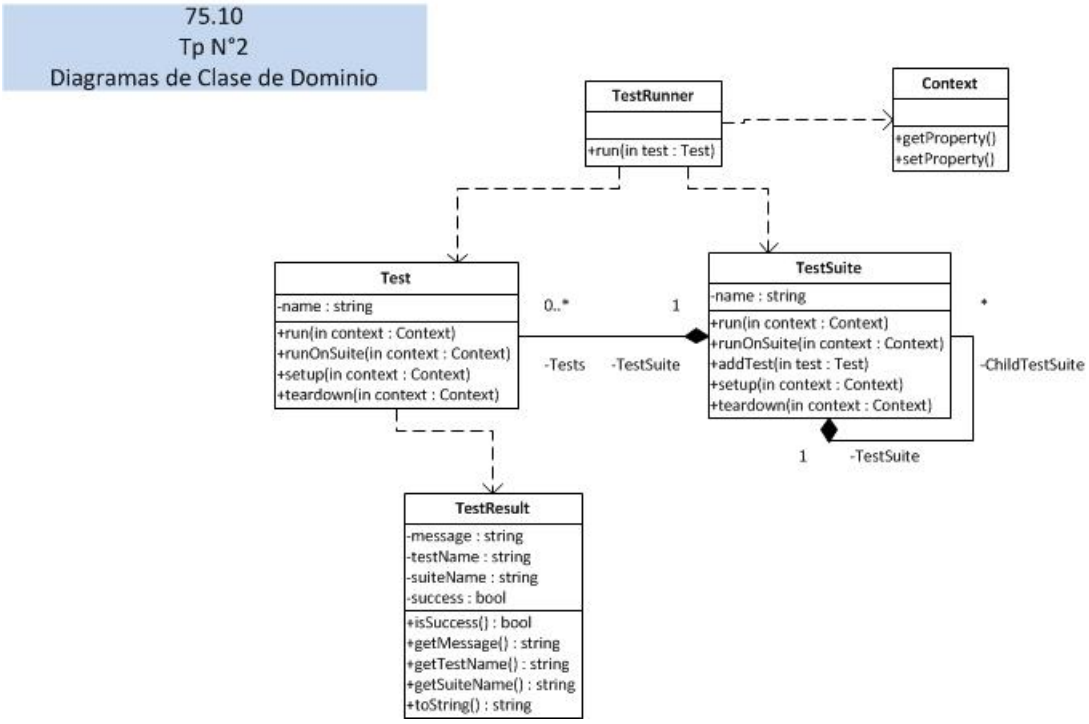
* No se aceptaran TP?s que violen alguna de las restricciones.

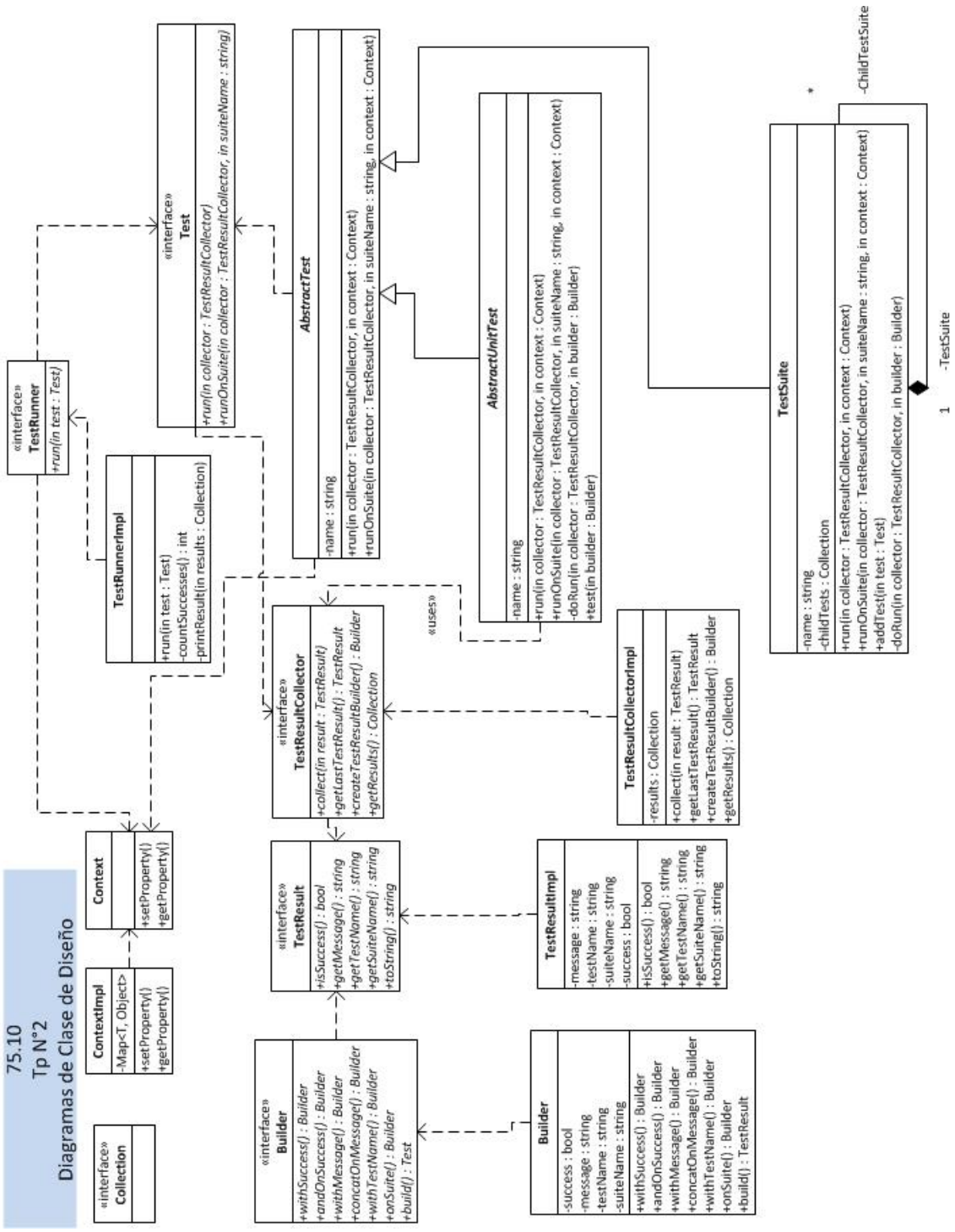
Criterios de Corrección

- 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.

2. Diagramas


75.10
Tp N°2
Diagramas de Clase de Dominio
TestRunner
+run(in test : Test)
Context
+getProperty()
+setProperty()
Test
-name : string
+run(in context : Context)
+runOnSuite(in context : Context)
+setup(in context : Context)
+teardown(in context : Context)
0..*
1
-Tests
-TestSuite
TestSuite
-name : string
+run(in context : Context)
+runOnSuite(in context : Context)
+addTest(in test : Test)
+setup(in context : Context)
+teardown(in context : Context)
*
-ChildTestSuite
1
-TestSuite
TestResult
-message : string
-testName : string
-suiteName : string
-success : bool
+isSuccess() : bool
+getMessage() : string
+getTestName() : string
+getSuiteName() : string
+toString() : string


75.10
Tp N°2
Diagramas de Clase de Diseño
«interface»
Collection
ContextImpl
-Map<T, Object>
+setProperty()
+getProperty()
Context
+setProperty()
+getProperty()
«interface»
TestRunner
+run(in test : Test)
TestRunnerImpl
+run(in test : Test)
-countSuccesses() : int
-printResult(in results : Collection)
«interface»
Test
+run(in collector : TestResultCollector)
+runOnSuite(in collector : TestResultCollector, in suiteName : string)
AbstractTest
-name : string
+run(in collector : TestResultCollector, in context : Context)
+runOnSuite(in collector : TestResultCollector, in suiteName : string, in context : Context)
AbstractUnitTest
-name : string
+run(in collector : TestResultCollector, in context : Context)
+runOnSuite(in collector : TestResultCollector, in suiteName : string, in context : Context)
-doRun(in collector : TestResultCollector, in builder : Builder)
+test(in builder : Builder)
«uses»
«interface»
TestResultCollector
+collect(in result : TestResult)
+getLastTestResult() : TestResult
+createTestResultBuilder() : Builder
+getResults() : Collection
TestResultCollectorImpl
-results : Collection
+collect(in result : TestResult)
+getLastTestResult() : TestResult
+createTestResultBuilder() : Builder
+getResults() : Collection
TestSuite
-name : string
-childTests : Collection
+run(in collector : TestResultCollector, in context : Context)
+runOnSuite(in collector : TestResultCollector, in suiteName : string, in context : Context)
+addTest(in test : Test)
-doRun(in collector : TestResultCollector, in builder : Builder)
*
-ChildTestSuite
1
-TestSuite
«interface»
TestResult
+isSuccess() : bool
+getMessage() : string
+getTestName() : string
+getSuiteName() : string
+toString() : string
TestResultImpl
-message : string
-testName : string
-suiteName : string
-success : bool
+isSuccess() : bool
+getMessage() : string
+getTestName() : string
+getSuiteName() : string
+toString() : string
«interface»
Builder
+withSuccess() : Builder
+andOnSuccess() : Builder
+withMessage() : Builder
+concatOnMessage() : Builder
+withTestName() : Builder
+onSuite() : Builder
+build() : Test
Builder
-success : bool
-message : string
-testName : string
-suiteName : string
+withSuccess() : Builder
+andOnSuccess() : Builder
+withMessage() : Builder
+concatOnMessage() : Builder
+withTestName() : Builder
+onSuite() : Builder
+build() : TestResult

3. Justificaciones

- **Herencia**: Abstract Test, AbstractUnitTest, TestSuit
  Motivación:
  Reducir la duplicidad de código. Los AbstractUnitTest y los TestSuit tienen caracteristicas comunes a ambos, las cuales agrupamos en la clase AbstractTest. En principio solo encontramos la necesidad de un nombre como caracteristica comun, pero podría haber mas, entonces esto ayuda también a encapsular el cambio.

- **Patrones utilizados:**

  Builder:
  Utilizado en: TestResult.java, clase ?Builder?
  Motivación: Se requería construir un TestResult (resultado de un test), como resultado de distintos eventos que modificaban el estado del mismo. El Builder permite hacer esto de una forma clara.

  Composite:
  Utilizado en: Test, TestSuite
  Motivación: La forma en que se organizan los test con la estructura de un arbol presentan una oportunidad de utilizar este patrón beneficianos de sus ventajas y sin tener que lidiar con sus desventajas (la interfaz de los componentes es válida tanto para elementos hoja como para elementos no hoja).

  Visitor:
  Utilizado en: TestResultCollector
  Motivación: Simplifica la recolección de resultados en la jerarquia de tests.

  Command:
  Utilizado en: TestRunner
  Motivación: Simplificación de interfaz de runner.

- **Clases**:

  Las justificaciones pueden encontrarse en el codigo como comentario en cada clase, aquí se expone un resumen de las mismas.

  Interfaces:

  Test:
  Interfaz genérica para un test.

  TestResult:
  Segregacion de interfaz.

  TestResultCollector:

Separar implementacion del colector de resultados.

TestRunner:
Segregacion de interfaz.

Assert1:
De esta forma los asserts que tengan el tipo de firma de Assert1 (recibiendo solo 1 parametro) solo deben redefinir el metodo apply para poder ser utilizados.

Assert2:
Igual Assert1.

Context:
Persistencia a travez de la corrida de los tests, de los elementos creados en los Setup.

Clases Concretas:

AssertXXXXX:
Implementan los distintos tipos de verificación.

Asserts:
Permite utilizar los asserts sin tener que crear instancias de los mismos.
AbstractUnitTest:
Implementacion para los tests unitarios.

TestResultCollectorImpl:
Implementacion del colector de resultados. Para poder obtener todos los resultados en el mismo lugar, en el caso de estar corriendo una estructura tipo arbol de tests.

TestResultImpl:
Implementación de Resultado de tests, tiene cierta información que se cree de utilidad para quien ha corrido los tests.

TestRunnerImpl:
Implementación del TestRunner, permite correr un test o suite de tests y obtener un resumen sobre los resultados.

TestSuite:
Contiene una colección de tests, permite organizar y correr tales tests, y tambien permite correr otros TestSuites, lo que otorga la posibilidad de correr los tests con una estructura del tipo arbol.

ContextImpl:

Implementación de Persistencia de contexto.