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

Técnicas de Diseño (75.10)

TRABAJO PRÁCTICO N°2.2
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

- Poder expresar un test de performance que falle si el test tardó más del tiempo especificado.
- Recordar corridas anteriores, para luego poder solo correr los últimos fails/errors + los nuevos (Se debe indicar a la hora de correr los test si es que se quiere correr bajo ese modo).
- Ofrecer al menos dos ?stores? posibles para recordar las corridas. Se debe permitir al usuario elegir el que quiera. Plus: ofrecer una manera sencilla de agregar un nuevo ?store? por parte del usuario. (Es decir que sea ?pluggable?).

Casos de Prueba de ejemplo

Caso	GIVEN	WHEN	THEN
Test de performance fallido.	T1 (sin fallas o errores) que dura X + 1 con límite de X tiempo.	Correr T1	T1 debería ser fallido por tardar más tiempo.
Test de performance exitoso.	T1 (sin fallas o errores) que dura X con límite de X + 1 tiempo.	Correr T1	T1 debería ser exitoso.
Recordar Tests anteriores Con fallidos, errores, y nuevos.	Dado una corrida anterior donde: - T1 fue exitoso. - T2 dio error. - T3 fue fallido. Y: Un T4 nuevo . Todos pertenecientes a un TS.	Correr el TS indicando que solo se quieren correr los fallidos/errores/nuevos.	Se debería solo correr: T2, T3 y T4.
Recordar Tests anteriores Sin fallidos, ni errores, ni nuevos.	Dado una corrida anterior donde: - T1 fue exitoso. - T2 fue exitoso. - T3 fue exitoso. Todos pertenecientes a un TS.	Correr el TS indicando que solo se quieren correr los fallidos/errores/nuevos.	No se debería correr ningun test.

Restricciones

- Se debe trabajar sobre el trabajo práctico de otro grupo.
- 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 debe realizar una crítica del Diseño, Código, Tests y herramientas utilizadas sobre el TP que les ha tocado.
- Se deberá hacer un commit del código recibido de otro grupo en un branch del repositorio propio, y de ahí en todos los cambios se realizarán sobre ese branch.

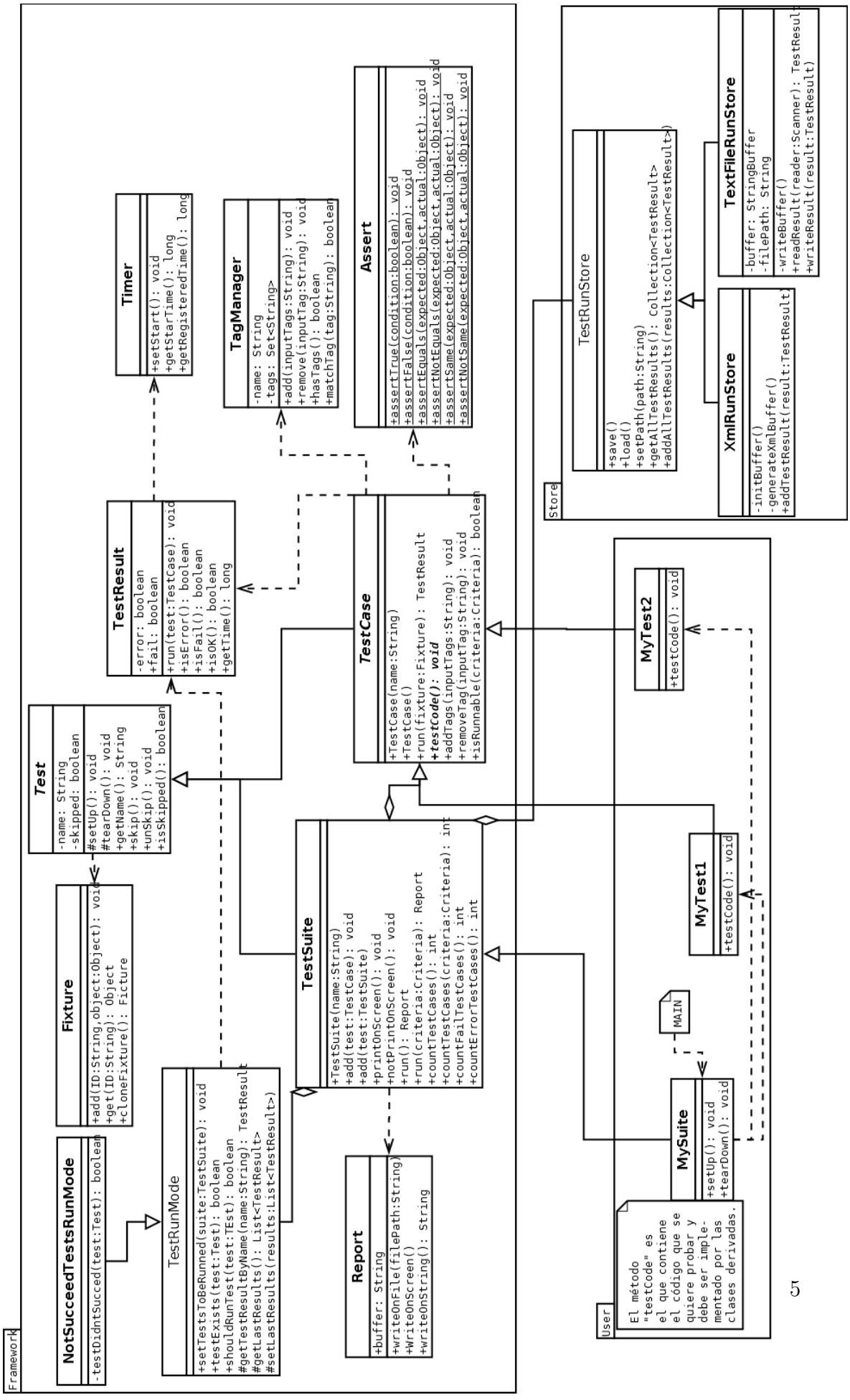
* No se aceptaran TP?s que violen alguna de las restricciones.

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


Framework
NotSucceedTestsRunMode
-testDidntSucceed(test:Test): boolean
Fixture
+add(ID:String,object:Object): void
+get(ID:String): Object
+cloneFixture(): Ficture
Test
-name: String
-skipped: boolean
#setUp(): void
#tearDown(): void
+getName(): String
+skip(): void
+unSkip(): void
+isSkipped(): boolean
TestRunMode
+setTestsToBeRunned(suite:TestSuite): void
+testExists(test:Test): boolean
+shouldRunTest(test:TEst): boolean
#getTestResultByName(name:String): TestResult
#getLastResults(): List<TestResult>
#setLastResults(results:List<TestResult>)
TestResult
-error: boolean
+fail: boolean
+run(test:TestCase): void
+isError(): boolean
+isFail(): boolean
+isOK(): boolean
+getTime(): long
Timer
+setStart(): void
+getStarTime(): long
+getRegisteredTime(): long
TagManager
-name: String
-tags: Set<String>
+add(inputTags:String): void
+remove(inputTag:String): void
+hasTags(): boolean
+matchTag(tag:String): boolean
Assert
+assertTrue(condition:boolean): void
+assertFalse(condition:boolean): void
+assertEquals(expected:Object,actual:Object): void
+assertNotEquals(expected:Object,actual:Object): void
+assertSame(expected:Object,actual:Object): void
+assertNotSame(expected:Object,actual:Object): void
TestCase
+TestCase(name:String)
+TestCase()
+run(fixture:Fixture): TestResult
+testCode(): void
+addTags(inputTags:String): void
+removeTag(inputTag:String): void
+isRunnable(criteria:Criteria): boolean
TestSuite
+TestSuite(name:String)
+add(test:TestCase): void
+add(test:TestSuite)
+printOnScreen(): void
+notPrintOnScreen(): void
+run(): Report
+run(criteria:Criteria): Report
+countTestCases(): int
+countTestCases(criteria:Criteria): int
+countFailTestCases(): int
+countErrorTestCases(): int
Report
+buffer: String
+writeOnFile(filePath:String)
+WriteOnScreen()
+writeOnString(): String
MyTest2
+testCode(): void
MyTest1
+testCode(): void
MAIN
MySuite
+setUp(): void
+tearDown(): void
User
El método "testCode" es el que contiene el código que se quiere probar y debe ser implementado por las clases derivadas.
Store
TestRunStore
+save()
+load()
+setPath(path:String)
+getAllTestResults(): Collection<TestResult>
+addAllTestResults(results:Collection<TestResult>)
XmlRunStore
-initBuffer()
-generateXmlBuffer()
+addTestResult(result:TestResult)
TextFileRunStore
-buffer: StringBuffer
-filePath: String
-writeBuffer()
+readResult(reader:Scanner): TestResult
+writeResult(result:TestResult)

3. Justificaciones

- **Herencia**: TestRunStore, XmlRunStore, TextFileRunStore
  Motivación:
  Reducir duplicidad de código. Poder utilizarse cualquier clase concreta que respete la interfaz de TestRunStore. TestRunMode, NotSucceedTestsRunMode
  Motivación:
  Reducir duplicidad de código.

- **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**:

  Clases Abstractas:

  TestRunStore:
  Implementa funcionalidad y define interfaz para un Store TestRunMode:
  Implementa funcionalidad y define interfaz para un Modo de ejecución

  Clases Concretas:

  TextFileRunStore:
  Implementa un Store utilizando un archivo de texto. XmlRunStore:
  Implementa un Store utilizando un archivo XML. NotSucceedTestsRunMode:
  Setea un test Suite para que solo corra los tests que no han sido exitosos en corridas anteriores.