Índice

Enunciado

75.10 Técnicas de diseño

Trabajo Práctico N° 2.3

Nota: Leer restricciones y criterios, se han agregado nuevos.

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
- Crítica del Diseño, Código, Tests y utilización de herramientas por parte del otro grupo.

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 14/11	Presentación del TP
Jue 21/11	Entrega TP, via campus

Comentarios del código del Grupo 7

Diseño y Código

- Nos parece que confundieron *Error* con *Failure*. El *Failure* es cuando falla. El *Error* es un caso no esperado en la ejecución del test (por ejemplo cuando las variables para comparar no están seteadas). Esto no es un error, pero por convención se usa la otra "forma de nomenclatura": *Error* cuando pincha, *Failure* cuando no se obtiene lo que se espera.

- El "Reporte por pantalla" es poco claro. Primero, cada informe en sí es complejo de entender. Después se mezclan los informes nuevos con los informes viejos. Esto hace que sea difícil ver el resultado.

- Nos da la impresión de que utilizaron muchas interfaces con la idea de generar "una especie de header" con todas las firmas de los métodos, y luego otro con los métodos implementados. Esto causó que haya una gran cantidad de clases, que en un primer momento, era difícil de leerlos (y daba la impresión de que era un proyecto grande). Pero tras leer y entender su código, vimos que estuvo muy bien la jerarquía y elección de paquetes para las clases.

- Esto es algo menor, pero cuando recibimos su TP, había muchos *warnings* en el proyecto. La gran mayoría se debían a *import* que no se usaban (esto tiene relación con lo dicho de de muchos paquetes y clases del párrafo anterior).

- Cuando recibimos su TP, también había un par de métodos comentados. Suponemos que esto fue porque llegaron muy justos de tiempo con la última entrega.

Tests

- Nos pareció poco claro crear un nuevo Test. Aunque una vez que se entiende, es fácil. Pero para eso lo mejor es ver un ejemplo, ya que con sólo ver el código se complica.

- Por algunas cosas dichas anteriormente (principalmente el orden de los paquetes y la jerarquía de las clases), vimos que a la hora de hacer muchos tests, hay poco código duplicado.

Herramientas utilizadas

- El Grupo 7 utilizaron las herramientas del lenguaje Java especificadas en el enunciado (git + maven + junit4), lo mismo que en nuestro TP.

- La diferencia es que ellos trabajaron en el sistema operativo Linux, mientras que nosotros en Windows. Esto no trajo problemas a la hora de intercambiar TP, ya que Java es multiplataforma.

Hipótesis

- Al momento de evaluar los tests, no se van a ejecutar 2 tests con el mismo nombre.

- Si se le cambian los datos (los valores a comparar por el "assert") a un test que ya dio "OK", eso no hace que el test se corra de vuelta (en la próxima ejecución).

- Un TestSuite cuyos resultados fueron todos "OK" no va a volver a correrse. Si quiere volver a correrse, debe crearse un nuevo TestSuite, o cambiarle el nombre.

- La clase **TestRunStoreAcccess** controla el acceso al **TestRunStore**. Es un *Singleton*. Por defecto si se la quiere usar sin inicializar toma un ".txt".

- El objetivo de recordar las corridas es para luego correr solamente lo que dio erróneo. Por esa razón, lo único que se graba en un archivo son los tests que dieron "OK", que es lo que se debe ignorarse en la próxima corrida.

Clases nuevas

TestRunStoreAccess

Clase que es un *Singleton,* y sirve para controlar el acceso a la utilización del **TestRunStore**. Si no se especifica un **TestRunStore**, por defecto se utilizará **TxtFileTestRunStore**.

TestRunStore

Interfaz que deben cumplir todos los tipos de "Store".

Por cómo **TestRunStoreAccess** utiliza a su "Store", con esta jerarquía de "Store" se aceptarían más tipos de "Store". Simplemente se crea una nueva clase de "Store" deseado, y se le indica a **TestRunStoreAccess** que use ese "Store".

MemoryTestRunStore

Clase que implementa **TestRunStore**. Este "Store" guarda la información de las ejecuciones de test en memoria.

Cabe destacar que en una nueva corrida del programa, se pierden los datos de las ejecuciones anteriores.

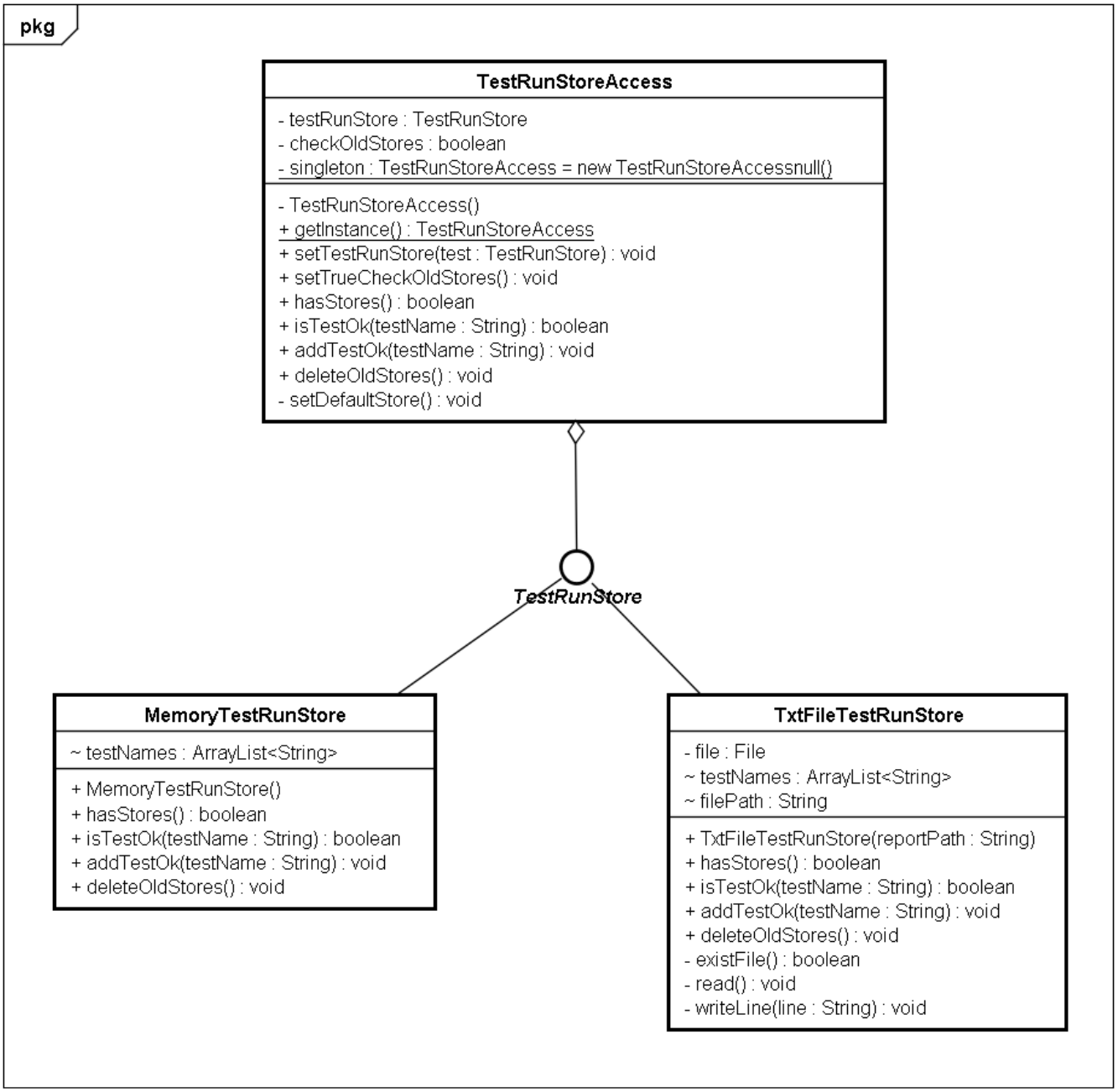
TxtFileTestRunStore

Clase que implementa **TestRunStore**. Este "Store" guarda la información de las corridas en memoria. Este "Store" guarda en un archivo de texto plano con extensión '.txt' la información de las ejecuciones de test.

En una nueva corrida del programa, se puede recuperar información de ejecuciones pasadas.

Diagrama de clases

Diagrama de clases del paquete nuevo: Store



En el resto de los paquetes y clases, aunque se haya modificado algunas líneas de código para agregar las funcionalidades nuevas, se mantuvieron las mismas relaciones, jerarquías y usos. Por lo tanto los diagramas quedan igual a la versión anterior del TP.

Diagrama de clases de Diseño

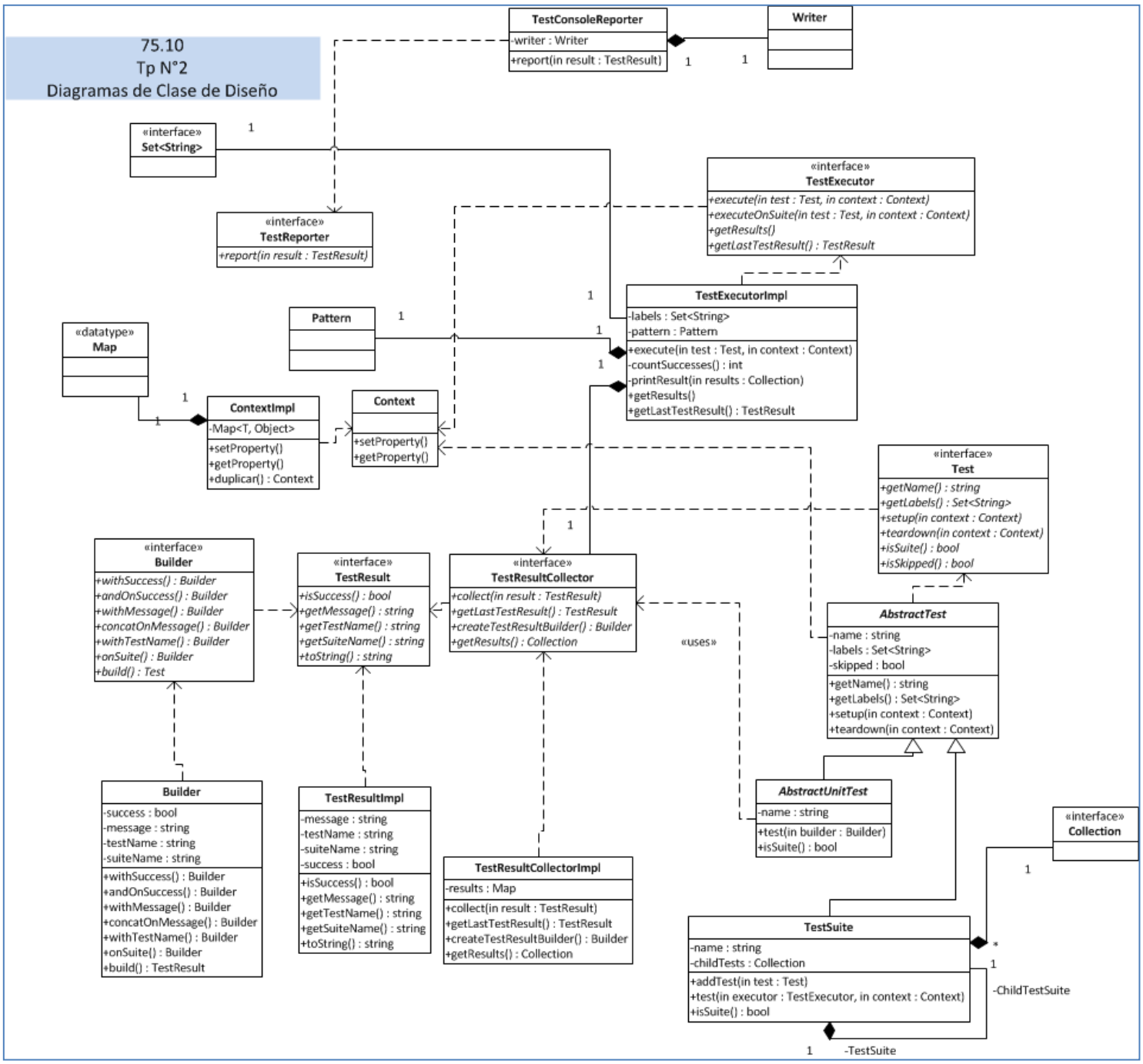


Diagrama de clases de Dominio