Práctica 2

Tecnologías para el Desarrollo de Software
Grado en Ingeniería Informática
Mención Ingeniería del Software

Y. Crespo

Curso 2019-2020

Desiderata

En un sistema de información se requiere desarrollar los servicios de un boletín de noticias.

A continuación se describen las características que se ha planificado incluir en la iteración actual:

En relación con las noticias se desea que:

1. Una noticia tenga necesariamente un titular, la fecha de publicación, la fuente (nombre del medio o agencia de prensa), una URL donde se encuentra el contenido completo de la noticia y una categoría.

2. Las categorías en las que se puede encuadrar una noticia son: nacional, internacional, sociedad, economía, deporte y cultura.

3. Los titulares de las noticias deberán tener menos de 13 palabras y al menos una. Los titulares de una palabra son escasos pero admisibles, por ejemplo: "¡Vergüenza!".

4. Una noticia deberá poder ser comparada con otra en cuanto a la fecha de publicación indicando si es anterior, igual o posterior.

5. Se considerará una noticia «similar» a otra si coincide en el titular y la categoría, aunque la fuente y la URL con el contenido completo no coincidan. En relación a la fecha de publicación se considerará «similar» hasta con dos días de diferencia en la fecha (anterior o posterior).

Un boletín de noticias es una agrupación y selección de noticias con algún fin. En relación con el boletín de noticias se desea:

1. poder crear un boletín vacío.

2. poder agregar noticias a un boletín.

3. garantizar que el boletín no contiene noticias repetidas (idénticas).

4. poder crear un boletín a partir de una lista de noticias. El orden de aparición en la lista se considerará orden de llegada de la noticia al boletín.

5. poder saber cuántas noticias hay en un boletín.

6. poder saber la fecha de las noticias más recientes contenidas en el boletín y la fecha de las más antiguas.

7. obtener del boletín una lista en orden cronológico (de anterior a posterior) basado en la fecha de publicación de las noticias. Cuando dos noticias coincidan en fecha de publicación, se considerará como anterior la que primero se agregó al boletín.

8. obtener del boletín una lista de sus noticias ordenadas por categoría. En dicha lista siempre aparecerán en el orden en el que se mencionan en el punto 2 de las características de las noticias. Es decir, primero las nacionales, después las internacionales, etc. Dentro de cada categoría aparecerán las noticias en orden cronológico aplicando el mismo criterio que en el punto 7.

9. obtener del boletín una lista con las noticias «similares» a una dada contenidas en él, ordenadas aplicando el mismo criterio que en el punto 7.

10. obtener otros boletines cuyas noticias serán un subconjunto de las contenidas en el origen. Se considerarán varias posibilidades:

    a) dada una fecha en concreto, asegurando que en el boletín resultante todas las noticias han sido publicadas el día de la fecha indicada.

    b) dadas dos fechas, asegurando que en el boletín resultante todas las noticias han sido publicadas en el intervalo de fechas dado.

    c) dada una categoría de noticias, asegurando que en el boletín resultante todas las noticias serán de la categoría dada.

    d) combinación de las opciones a), b) con c), es decir, poder indicar que incluya todas las noticias de una categoría en una fecha, o todas las noticias de una categoría en un intervalo de fechas.

Tarea a realizar

A partir de esta desiderata inicial, la práctica consistirá en aplicar los pasos 1 y 2 del ciclo TDD (Rojo-Verde-Refactor) correspondientes a la fase Rojo:

1. Escribe los tests que ejercitan el código que deseas tener,

2. Comprueba que los tests fallan (fallo, no error).

Siguiendo el proceso de diseño dirigido por pruebas (TDD) :

(a) definiremos qué clases vamos a crear

(b) definiremos las clases de tests para realizar el diseño dirigido por las pruebas

(c) definiremos en las clases de tests cómo se usan los objetos de las clases que hemos creado

(d) especificaremos su funcionalidad en los tests

(e) describiremos esta funcionalidad en el javadoc de las clases creadas

El historial de *commits* debe permitir apreciar el proceso TDD.

Una vez establecida la primera versión de los tests que nos ha permitido crear los *stubs* de las clases, especificar su funcionalidad en dichos tests y describir ésta en el javadoc:

Aplique la filosofía *Test First*.

Añada nuevos tests (en clases de test separadas) teniendo en cuenta las técnicas de caja negra (pruebas unitarias (utilizando particiones basadas en los datos (fronteras del invariante), de secuencia (aleatorias o pruebas de particiones basadas en el estado según se necesite).

Aclaración: No se usará programación por contrato basada en asertos Java. Se diseñará aplicando Programación Defensiva.

Tests

Los tests serán implementados con JUnit. Se utilizará jUnit 5. A decisión del equipo se utilizará el módulo Vintage o el módulo Jupiter.

Se habrán creado los stubs necesarios de las clases diseñadas mediante TDD para compilar y ejecutar los tests.

Aplique criterios de modularidad para que las clases de test no crezcan demasiado. Considere la posibilidad de definir una o varias *fixtures*. Cree una o varias *suite*(s) para agrupar clases de tests. Como mínimo aporte una *suite* que incluye todos los tests desarrollados.

Se espera que el resultado de la ejecución de los tests en esta práctica sea **Rojo**. Por lo tanto, de haberse realizado algo de implementación de los stubs, ésta deberá ser una implementación falseada (*fake implementation*) con el propósito de que **los tests no produzcan errores sino fallos**. El objetivo es que todos los tests implementados fallen, salvo casos excepcionales claramente señalados y comentados. En dichos casos se escribirá `fail` como última instrucción del test para conseguir el fallo.

Los tests se categorizarán.

En el caso de utilizar JUnit 5 Vintage (compatibilidad con JUnit 4) se se utilizará la anotación `@Category` para indicar los que se han utilizado para ir diseñando la clase mediante TDD: `@Category(TDD.class)`, los test de caja negra realizados como *Test First* aplicando las técnicas de caja negra como se ha comentado anteriormente en el enunciado: `@Category(BlackBoxTestFirst.class)`. Se aportará una clase de test con pruebas de secuencia para las clases diseñadas que se categorizará como `@Category(Sequence.class)`.

En el caso de utilizar JUnit 5 Jupiter se utilizará la anotación `@Tag`. Para indicar los tests que se han utilizado para ir diseñando la clase aplicando TDD: `@Tag("TDD")`

Al final del documento se pueden consultar los criterios de evaluación de esta práctica mostrados en la Figura 1.

Características del proyecto Eclipse

- El proyecto Eclipse deberá nombrarse equipox-2019 (ejemplo: equipo1-2019). La x deberá ser sustituida por el número del equipo de prácticas. Este número se puede conocer por el canal Rocket creado para el equipo en `rocket.inf.uva.es`.
- Deberá ser un proyecto válido para Eclipse IDE (release instalada en los labs de la Escuela) y jdk 9 u 8.
- El proyecto deberá tener sus propios *settings* indicando el conjunto de caracteres utilizado para evitar problemas de importación en entornos diferentes.

- La carpeta `src` del proyecto Eclipse contendrá los fuentes `.java` de las clases diseñadas aplicando TDD y *Test First*.
- La implementación desarrollada por el equipo se alojará en un paquete Java que se nombrará es.uva.inf.tds.pr2.
- La carpeta `testsrc` del proyecto Eclipse contendrá los fuentes `.java` correspondientes a los test JUnit.
- Las clases y sus tests estarán en el mismo paquete aunque en carpetas de fuentes diferentes.

Control de versiones

Se llevará un control de versiones utilizando `git` y se utilizará `gitLab` (`gitlab.inf.uva.es`) como repositorio centralizado y `Atlassian Bitbucket` como repositorio de respaldo. El repositorio en `gitLab` deberá alojarse en un grupo creado para ello llamado practica2. Al finalizar, para realizar la entrega, se añadirá a la profesora al proyecto (cuando ya no se vayan a realizar más commits y pushes al repositorio centralizado en `gitLab` y al respaldo en `Atlassian Bitbucket`) tal y como se indica en las **normas de entrega**. En el repositorio no se tendrá ningún `.class` o `.jar` ni tampoco la documentación generada que puede generarse con *javadoc*. En esta práctica, como en la anterior, no se valorará el uso de `git` en cuanto a ramificación y merge. Aunque es recomendable emplear el método rama por tarea.

Issues, estimación y tiempo empleado

El proyecto en `gitlab.inf.uva.es` tendrá un listado de *issues* que se irán creando bien al comenzar el proyecto, bien durante el desarrollo del mismo. Al crear cada *issue*, se asignará a un miembro del equipo, se estimará (comando corto `/estimate`) el tiempo que se cree que será necesario para completar el *issue* y al finalizar, se indicará el tiempo que realmente ha sido necesario (comando corto `/spend`). En esta práctica, como en la anterior, no se valorará cómo se ha dividido el proyecto en *issues*, ni tampoco tendrá ningún efecto en la nota la diferencia entre el tiempo estimado y el empleado. Lo que se valora es que se haya estimado y se haya registrado lo real.

Normas de entrega

- El repositorio `git` y el centralizado en `gitLab`, así como el respaldo en `Atlassian Bitbucket`, deberá llamarse de la misma forma que el proyecto Eclipse.
- Cualquier *push* al repositorio una vez realizada la entrega será penalizado con 0 en la Práctica.
- La entrega se realizará añadiendo a la profesora (usuario `yania`) en `gitLab` (con rol Reporter) y `ycgc` (o `yania@infor.uva.es`) en `Atlassian Bitbucket`) (con permisos de sólo lectura al respaldo en `Atlassian Bitbucket`) cuando no se vaya a realizar ningún otro *commit & push*.
- Al entregar la práctica todo deberá quedar integrado en la rama **develop** (dado que no hay nada funcional).

- En el tracking de versiones y por tanto en el repositorio remoto solamente residirán los archivos de la configuración del proyecto Eclipse (.project, .settings/*, .classpath), los fuentes de los stubs de las clases de la solución y los fuentes de los tests.
- En los fuentes entregados se hará referencia a los autores de la práctica en cada archivo fuente con el tag `@author` de *javadoc* y solamente en ese punto.
- El proyecto tendrá un archivo `README.md` que indicará toda la información que quieran aportar los autores sobre su proyecto además de la siguiente información:
  - Tiempo total en horas-hombre empleado en la realización de la práctica.
  - Clases que forman parte de la solución y por cada clase: SLOC (Source Lines of Code) y LLOC (Logic Lines of Code, es decir, sin contar líneas de comentarios. Estos datos pueden obtenerse con Eclipse Metrics Plugin.
  - Clases de tests de las clases diseñadas (separadas por clases) y por cada clase de test correspondiente a una clase diseñada indicar SLOC y LLOC, y la suma de estos valores para todas las clases de test de una clase diseñada.
- Fecha límite de entrega: **2 de diciembre de 2019**. No se admitirán entregas fuera de plazo o por otra vía distinta a la indicada en estas normas.

Criterios de evaluación y recopilación de problemas frecuentes

En la Figura 1 se muestran los criterios que se aplicarán al evaluar esta práctica

A continuación se enumera una recopilación de problemas encontrados frecuentemente en este tipo de trabajos:

- No se aprecia TDD en el historial de commits.
- No hay especificación de la funcionalidad en javadoc.
- En la documentación de la clase no se reflejan las excepciones que se lanzan y por qué motivo.
- No se cambian los tipos autogenerados en los stubs, todo son `Objects`!!!
- Los tests TDD no especifican todo el comportamiento en casos no válidos. Es decir, no hay tests TDD con `expected` *exception class* (Vintage) o assertThrow (Jupiter) para indicar qué pasará cuando no se se puede ejecutar el método.
- No se separan los tests TDD de otros añadidos para completar el *Test First* con las técnicas de caja negra.
- No se categorizan los tests (con @Category en Vintage o @Tag en Jupiter).
- No se aplican técnicas de caja negra.
- Los tests son tan escasos que sólo por esa razón es imposible que se hayan aplicado bien las técnicas de caja negra (partición en clases de equivalencia con análisis de valores límite, fronteras del invariante, particiones basadas en el estado).
- No hay ninguna *suite* de tests.

- No hay ninguna *fixture*.
- El concepto de *fixture* está mal entendido. Hay una fixture pero está mal concebida, se necesitan varios tests agrupados que partan del mismo estado de los objetos. Preparar el estado de partida no debe ser una trivialidad de una instrucción.
- No hay pruebas de secuencia.
- Sobre el diseño de las clases es básico que no debemos tener funciones con efectos colaterales. Ejemplo: Los métodos que modifican el objeto no deben devolver `boolean`, si no pueden realizar algo porque un parámetro o el propio objeto que recibe el mensaje no cumple unas condiciones, deben lanzar excepción.
- `assertSame` para tipos básicos no es un buen concepto.
- En tests de constructores no basta con usar `assertNotNull`, hay que comprobar los valores con los que se ha creado el objeto.
- `assertNotNull` para probar los getters no es suficiente. Si hay que probar un getter, importa el valor que devuelve. Esto es especialmente importante cuando se trata de valores calculados y no acceso a valores almacenados.
- Se definen tests que esperan NullPointerException, esto no debe ser porque NullPointerException no es una excepción que uno deba lanzar controladamente, esconde problemas de programación.
- Los tests no son de tipo expected exception o assertThrow (casos no válidos) y tampoco tienen otros `assert`* (casos válidos). Esto no tiene sentido.
- Se debe eliminar *warnings* todo lo posible, en particular de las clases de tests.
- Se debe aplicar técnicas de modularidad en los tests para no tener una clase de tests de muchos métodos y no saber dónde buscar. Implica buen nombrado y categorización de clases de tests.
- No se especifican los TO DO's necesarios con sus comentarios de cambiar *fake implementation* en las clases o `fail`s añadidos en los tests.
- Los tests en los que hay comprobación de valores de listas, conjuntos, arrays,... se hacen muy largos y no utilizan o bien `assertArrayEquals` y definición de arrays literales para el resultado esperado, o bien Matchers de Harmcrest como `assertThat(listaObtenida, IsIterableContainingInOrder.contains(listaEsperado.toArray()));`

General												
scm	0,5		Gestión de la configuración. Version Control, Issue tracking. Se desglosa en 2 items									
javadoc	0,5		Clase y métodos con javadoc. Se deben reflejar las excepciones que se lanzan y por qué motivo									
Stubs	2		Los stubs están bien fabricados, se obtiene un buen diseño orientado a objetos									
Tests	7		Se desglosa en 5 items									
	10											
Desglose de scm												
scm1	0,25		Se usa control de versiones en gitLab con respaldo en bitbucket									
scm2	0,25		Se gestionan issues con estimación de tiempo e indicación de tiempo empleado									
	0,5											
Desglose de Tests												
tests1	3		Se aprecia TDD en el control de versiones. Los tests especifican la clase y sirven para crear los stubs de cada método y su javadoc									
tests2	1		Los tests no tienen errores, fallan todos . Se añaden fake implementations si se necesitan para no tener errores en los tests									
tests3	2		Se aplican bien las técnicas de caja negra. Se hace al menos una prueba de secuencia por cada clase.									
tests4	0,5		Se crea al menos una fixture y al menos una suite									
tests5	0,5		Se cumplen los principios FIRST (Fundamentalmente en este caso se mide: Independent, Repeatable, Self-Checking). No hay malos usos de los asserts. Calidad en los tests. Modularidad, legibilidad, comprensibilidad									
	7											
			scm1	scm2	javadoc	stubs	tests1	tests2	tests3	tests4	tests5	check
muy bien	A	10	0,25	0,25	0,5	2	3	1	2	0,5	0,5	10
bien	B	7,5	0,1875	0,1875	0,375	1,5	2,25	0,75	1,5	0,375	0,375	7,5
regular	C	5	0,125	0,125	0,25	1	1,5	0,5	1	0,25	0,25	5
mal	D	2,5	0,0625	0,0625	0,125	0,5	0,75	0,25	0,5	0,125	0,125	2,5
muy mal o inexistente	E	0	0	0	0	0	0	0	0	0	0	0
Nota:	Se podrá obtener hasta 1 punto entero más en esta práctica de la siguiente forma											
	Al entregar la práctica 3, se comparará											
-	cuánto se ha tenido que retocar del diseño de la clase y la implementación de los tests de la Práctica 2 para que encajen en la Práctica 3											
-	se medirá el grado de cobertura obtenido usando solamente los tests realizados en esta práctica											

Figura 1: Criterios de evaluación de la Práctica 2