Práctica 4

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

Mario Corrales

Curso 2020-2021

La realización de esta práctica equivale al 25 % de la asignatura. La práctica tiene carácter **individual**.

Tarea a realizar

El objetivo de la práctica es realizar el ciclo TDD (Rojo-Verde-Refactor) completo para el supuesto que se describe y de acuerdo a las instrucciones que se dan a continuación.

Desiderata

En una ciudad se va a implementara un sistema de alquiler de bicis. En el primer sprint del desarrollo del software de información sobre el sistema de alquiler se ha decidido incluir las siguientes necesidades:

El sistema está formada por varias estaciones. Un sistema para serlo deberá tener al menos una estación. Una estación está univocamente identificada en el sistema por un identificador único en todo el sistema (cadena de texto). Además, cada estación tiene al menos un punto de recogida de bicis. También tiene unas coordenadas del lugar donde se encuentra. Se desea que se pueda añadir nuevos puntos de recogida, eliminar puntos de recogida y cambiar el estado de un punto de recogida.

Un punto de recogida tiene un identificador único en todo el sistema (cadena de texto) y un estado. Los estados pueden ser Libre, Ocupado o Inactivo. Existen restricciones para cambiar el estado de un punto de recogida: De Libre se puede pasar a Ocupado o Inactivo De Ocupado se puede pasar a Libre. De Inactivo se puede pasar a Libre.

Las coordenadas están definidas por una latitud y una longitud, en formato decimal. La latitud puede tener valores entre -90 y 90 y la longitud entre -180 y 180. Además, se debe poder obtener la distancia entre dos coordenadas en metros.

El sistema de reserva debe permitir añadir nuevas estaciones, eliminar estaciones, obtener todas las estaciones del sistema, alquilar una bici dado un punto de recogida (si es posible), devolver una bici (si es posible), desactivar un punto de recogida (si es posible), buscar las estaciones cercanas (500m) a unas coordenadas dadas, obtener la distancia entre dos estaciones y obtener el número de puntos de recogida de una estación que estén en el estado indicado.

Para todas las operaciones que no se puedan realizar, el sistema lo deberá notificar.

Se ha decidido que se pueda obtener toda la información sobre el sistema (todas sus estaciones en formato xml) y recíprocamente, dado un xml que contiene información sobre las

estaciones del sistema, construir el objeto que representa dicha sistema. Estos ficheros xml tienen que tener su DTD como validación.

Como punto de partida, en aulas se encuentran los stubs de las clases Estación, PuntoRecogida y Coordenadas que se han obtenido como resultado de un proceso TDD realizado por otros miembros de la organización.

Control de versiones

Se llevará un control de versiones utilizando `git` y se utilizará `gitLab` (`gitlab.inf.uva.es`) como repositorio centralizado.

Al finalizar, para realizar la entrega, se añadirá al profesor al proyecto (cuando ya no se vayan a realizar más *commits* y *pushes* al repositorio centralizado en `gitLab`, tal y como se indica en las **normas de entrega** (ver página 5). En el repositorio no se tendrá ningún `.class` o `.jar` ni tampoco la documentación de las clases que puede generarse con *javadoc* en cualquier momento.

En esta práctica **SÍ** se valorará el uso de `git` en cuanto a ramificación (*branchs*) y fusión (*merge*). Las ramas del proyecto incluirán: la rama *master*, una rama *develop* y se tendrá una **rama por tarea (*issue*)**.

Las integraciones en *master* deberán hacerse solamente de partes funcionales testadas, con los tests en verde.

Se utilizarán pruebas en aislamiento para poder probar la implementación realizada en la fase green, tenieno en cuenta que se aportan los stubs de las clases que ha diseñado otro "equipo" y que aún no están implementadas.

Integración Continua

El proyecto estará gestionado automáticamente mediante ant y maven.

En el repositorio `gitLab` no se tendrá ningún .class o .jar pero deberá bastar hacer un *pull* o *clone* del repositorio y utilizar el script ant (build.xml) para compilar, ejecutar, ejecutar los tests, analizar la cobertura, etc. Por ello, se gestionarán las dependencias externas del proyecto mediante maven. El script ant básicamente se utilizará para delegar en maven y así no tener que memorizar las tareas y parámetros maven necesarios para realizar el objetivo.

El proyecto deberá responder a un arquetipo maven descrito con el pom.xml correspondiente. Estará basado en el arquetipo maven-archetype-quickstart. La versión básica del pom.xml obtenida a partir del arquetipo indicado se debe revisar y modificar para añadir la gestión de dependencias, actualizar versiones así como la configuración de plugins necesarios para el análisis de cobertura, el análisis de calidad y su generación de informes y cualquier otro que el/la autor/a necesite.

El script ant (build.xml) deberá contar al menos con los siguientes objetivos que tendrán que llamarse exactamente así:

compilar garantiza la obtención de todas las dependencias y genera los .class a partir de los fuentes (depende de limpiar). Este objetivo es el default.

ejecutarTestsTDDyCajaNegra se ejecutan los tests categorizados como TDD y caja negra sin incluir secuencia.

ejecutarPruebasSecuencia se ejecutan solamente las pruebas de secuencia.

ejecutaCajaBlanca se ejecutan solamente las pruebas categorizadas de caja blanca que se añadieron para aumentar la cobertura

ejecutarTodoEnAislamiento incluirá la ejecución de los tres objetivos anteriores.

obtenerInformeCobertura obtener los informes de cobertura de sentencia, de rama, decisión/condición obtenidos con el total de todos los tests realizados (sin incluir los tests en aislamiento).

documentar genera la documentación del proyecto, incluida la carpeta doc con la documentación autocontenida en los fuentes mediante javadoc. La carpeta doc no debe alojarse nunca en el control de versiones.

limpiar deja el proyecto limpio eliminando todo lo generado con las ejecuciones de los objetivos.

Como se ha mencionado antes, para la consecución de estos objetivos será necesario descansar en llamadas a maven.

1. Fase 1: Rojo

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 clase o 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 pruebas de secuencia (aleatorias o pruebas de particiones basadas en el estado según se necesite).

Tenga en cuenta que los stubs de la clases `Estación`, `PuntoRecogida` y `Coordenadas` ya han sido creados.

2. Fase 2: Verde

En esta fase se implementará el diseño basado en TDD realizado en la Fase 1.

Se utilizarán mocks para conseguir que los tests de la clase diseñada pasen en verde.

Tests

Los tests serán implementados con JUnit 5. Se probará en aislamiento la clase que implementa un sistema de reserva de bicis utilizando mocks objects (basados en EasyMock). Se espera que el resultado de la ejecución de los tests en esta práctica sea **Verde**.

Los tests en aislamiento basados en mocks se categorizarán utilizando la anotación `@Tag` para indicar su naturaleza con la categoría: `@Tag(Isolation)`.

Los test de caja blanca realizados para aumentar la cobertura se categorizarán como: `@Tag(WhiteBox)`.

Se debe tener en cuenta que se pueden aportar varias categorías a una clase de test o a un test individual. Por ejemplo: `@Tag({WhiteBox, Isolation})`.

Tests en la Fase 1

Como resultado de la Fase 1 se espera que el resultado de la ejecución de los tests 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.

2.1. Tests en la Fase 2

Los tests de la Fase 2 serán de pruebas de caja blanca, para realizar pruebas en aislamiento y para aumentar la cobertura.

Se probará en aislamiento la clase que implementa un sistema de reserva de bicis (se aislará del resto de clases) utilizando mocks objects (basados en EasyMock).

Se espera que el resultado de la ejecución de los tests en esta fase sea **Verde** en todos los casos.

En esta práctica se medirá el grado de cobertura alcanzado y se intentará maximizar dicho grado de cobertura con la menor cantidad de test posibles. También se medirá la ratio estática *code to test* y la complejidad ciclomática por método y por clase de la clase implementada.

Fase 3: Refactor

Complete el ciclo TDD con la fase Refactor. Elimine duplicaciones. Mejore el código mediante Refactoring. Siga las recomendaciones para mejorar la calidad que encontrará ejecutando el plugin SonarLint de eclipse y los catálogos de refactorizaciones vistos en clase.

Realice las refactorizaciones con Eclipse. En el menú Refactor de Eclipse se podrá ver con la opción History las operaciones de refactoring que se han aplicado.

Debe recordarse que el comportamiento de Eclipse por defecto es tener una única carpeta .refactoring para todo el workspace y no por proyecto. Esta configuración debe cambiarse para que Eclipse almacene el historial de refactoring para el proyecto. Para hacer esto, se debe ir a las propiedades del proyecto, en la opción Refactoring History marcar "Persist proyect refactoring history in proyect folder instead of workspace".

Compruebe que esto se almacena correctamente en el repositorio central y que cuando se hace un *pull* se puede consultar el historial de refactoring realizado. Para ello añada al versionado y al repositorio remoto la carpeta oculta '.refactoring' en la que se hace persistente el historial de refactoring.

Recuerde que al inicio de un proceso de refactoring, los tests deben pasar en verde. Al finalizar esta fase el resultado de la ejecución de tests debe seguir siendo VERDE.

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 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 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.

Características del proyecto Eclipse y Maven

- El proyecto Eclipse deberá nombrarse tds-login-pr4 con el login del alumno en los laboratorios de la Escuela (ejemplo: tds-migarci-pr4).
- Deberá ser un proyecto válido para Eclipse IDE for Java developers release 2020-09 y jdk 11.
- El proyecto será un proyecto maven por lo que será necesario un archivo pom.xml correspondiente al arquetipo maven-archetype-quickstart.
- La estructura de carpetas deberá corresponderse con la definida por el arquetipo maven.
- En el archivo pom.xml los tags `groupId`, `artifactId` y `name` tendrán el siguiente contenido:

```
<groupId>2020-2021-tds-login</groupId>
<artifactId>p4</artifactId>
<name>2020-2021 práctica 4 de TDS del alumno login</name>
```

- Las clases y sus tests estarán en el mismo paquete aunque en carpetas de fuentes diferentes (las carpetas que genera para ello el arquetipo maven).
- El proyecto deberá tener sus propios *settings* indicando el conjunto de caracteres utilizado para evitar problemas de importación en entornos diferentes (indicado en el pom file).

Normas de entrega

- El repositorio `git` y el centralizado en `gitLab` 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 al profesor (usuario `marcorr` en `gitLab` con rol Reporter al repositorio que contiene el proyecto en `gitLab` cuando no se vaya a realizar ningún otro *commit & push*.
- El script ant (build.xml) debe tener los objetivos descritos en el enunciado.

- Al entregar la práctica todo deberá quedar integrado en la rama *master*.
- Se debe realizar también la entrega en la tarea habilitada en aulas.
- En el tracking de versiones y por tanto en el repositorio remoto solamente residirán los archivos de la configuración del proyecto Eclipse, los fuentes de las clases de la solución, los fuentes de los tests y los archivos pom.xml (maven), build.xml (ant) y la carpeta .refactoring para poder consultar el historial de refactoring.
- En los fuentes entregados se hará referencia al autor 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 quiera aportar el autor 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. Se indicará la ratio *code to test* calculada.
  - Breve explicación del proceso TDD que se ha seguido, indicando algunas ventajas y inconvenientes de este proceso, según vuestra propia experiencia.
- Fecha límite de entrega: **14 de enero de 2021 a las 23:59**. No se admitirán entregas fuera de plazo o por otra vía distinta a la indicada en estas normas.