Práctica 3

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

Mario Corrales

Curso 2022-2023

Tarea a realizar

En esta Práctica se implementará el diseño basado en TDD y *Test First* realizado en la Práctica 2. En esta Práctica el objetivo es obtener los tests en VERDE y aplicar las técnicas de refactorización. Es decir, realizar las fases Green y Refactor del ciclo Red-Green-Refactor.

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 tomará la medida de la métrica estática ratio *code to test*.

La implementación se realizará aplicando integración continua y gestión automática de dependencias.

La implementación de las clases relacionadas con las reservas y alojamientos se hará para obtener todos los test desarrollados en la práctica 2 en verde. La implementación de las clases relacionadas con el gestor deberá utilizar la interfaz *IDatabaseManager*, clase encargada de gestionar una conexión con una base de datos y con diferentes operaciones relacionadas con el almacenamiento y recuperación de reservas. Por lo tanto, las clases relacionadas con el gestor de reservas no deben almacenar ninguna información sobre las reservas, si no delegar en la clase *IDatabaseManager*. Debido a que no se tiene implementación de esta clase, se realizán test en aislamiento para poder probar la clase del gestor. Esta interfaz estará disponible en aulas (hay que sustituir las clases 'X' e 'Y' por el nombre de las clases generadas en TDD).

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 4). En el repositorio no se tendrá ningún `.class` o `.jar` ni tampoco cualquier otro fichero que pueda generarse a partir de las clases (como la documentación en javadoc, informes de cobertura, etc).

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*)**.

Cuando se finaliza una tarea, se deberá integrar en la rama *develop*. Cuando se realicen pruebas y depuración deberá siempre hacerse en una rama diferente a la rama *develop*, incluso

diferente a la rama en la que se haya estado desarrollando la tarea de programación asignada, creando un issue para esto en el issue tracker.

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

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 (versión 1.4). 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 los/as autores/as necesiten.

Para comenzar el trabajo se realizará un nuevo repositorio que inicialmente contendrá los mismos ficheros que los entregados en la práctica 2 pero adaptados a la estructura de carpetas creadas por el arquetipo maven descrito en la sección .

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

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

compilar garantiza la obtención de todas las dependencias y genera los .class a partir de los fuentes.

documentar genera la documentación del proyecto con la documentación autocontenida en los fuentes mediante javadoc.

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

ejecutarTestSinAislamiento ejecuta todos los test que no dependan de mocks.

ejecutarTestEnAislamiento ejecuta todos los tests que se tengan basados en mocks.

ejecutarTest ejecuta todos los tests.

obtenerInformeCobertura obtener los informes de cobertura de sentencia, de rama, decisión/condición obtenidos con el total de todos los tests realizados. Para realizar esta tarea, se utilizará el plugin de maven jacoco.

ejecutarSonarQube ejecutar el análisis sobre la plataforma sonarqube usando el plugin maven correspondiente.

all ejecuta todas las tareas definidas anteriormente. Este objetivo es el que se tiene que ejecutar por defecto al ejecutar ant sin parámetros.

Como se ha mencionado antes, para la consecución de estos objetivos será necesario descansar en llamadas a maven. Además, es necesario definir las dependencias entre las tareas ant para que los objetivos se ejecuten adecuadamente y con independencia del sistema operativo.

Tests

Los tests serán implementados con JUnit 5 (versión 5.8.1). Además de los tests realizados en la Práctica 2, se realizarán pruebas de caja blanca para aumentar la cobertura. Los tests en aislamiento se realizarán utilizando la librería EasyMock (versión 5.0.1)

Se espera que el resultado de la ejecución de los tests en esta práctica sea **Verde** en todos los casos, tanto en las pruebas en aislamiento basadas en mocks como en las pruebas con los objetos reales. Los test de caja blanca realizados para aumentar la cobertura se categorizarán como: `@Tag(WhiteBox)`.

Refactorización

Una vez obtenidos los test en verde, se deberán aplicar técnicas de refactorización para mejorar el código desarrollado. Con respecto a la implementación en sí, se deberá eliminar el código duplicado y los *code smells* reportados por Sonarqube. Además, se deberán aplicar al menos 3 refactorizaciones del catálogo de Fowler, cuya referencia tenéis en aulas. Estas tres refactorizaciones deberán estar justificadas en el README del repositorio. Si no es posible realizar alguna de estas refactorizaciones, se deberá justificar adecuadamente en el README del repositorio.

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, 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 Maven

- El proyecto será un proyecto maven por lo que será necesario un archivo pom.xml correspondiente al arquetipo maven-archetype-quickstart (versión 1.4).
- La estructura de carpetas deberá corresponderse con la definida por el arquetipo maven antes indicado.
- Deberá ser un proyecto válido para jdk 11.
- En el archivo pom.xml los tags `groupId`, `artifactId` y `name` tendrán el siguiente contenido:

```
<groupId>com.uva.tds</groupId>
<artifactId>practica3-grupoX</artifactId>
<name>2022-2023 práctica 3 de TDS del grupo x</name>
```

- En los datos anteriores la x deberá ser sustituida por el número de grupo.
- 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 maven deberá indicar el conjunto de caracteres utilizado en los fuentes para evitar problemas de importación en entornos diferentes.

Normas de entrega

- El repositorio `git` y el centralizado en `gitLab`, deberá llamarse tds-practica3-grupoX, siendo X el número de grupo asignado.
- 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*.
- El proyecto Eclipse también deberá subirse a la tarea habilitada en aulas en formato comprimido.
- El proyecto Eclipse será un proyecto válido para Eclipse IDE for Java developers release 2022-06. En ese caso los archivos necesarios para el proyecto Eclipse (.project, .settings/*, .classpath) estarán incorporados al tracking de versiones y alojados en el repositorio.
- En el tracking de versiones y por tanto en el repositorio remoto solamente residirán los archivos de la configuración del proyecto, los fuentes de las clases de la solución, los fuentes de los tests y los archivos pom.xml (maven) y build.xml (ant).
- En los fuentes entregados se hará referencia a los autores de la práctica en cada archivo fuente con el tag `@author` del *javadoc* de la cabecera de la clase 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-persona empleado en la realización de la práctica.
  - Clases que forman parte de la solución
  - Clases de tests de las clases diseñadas. Se indicará la ratio *code to test* calculada para cada clase.
  - Justificación de las refactorizaciones realizadas (sólo de las 3 relacionadas con el catálogo de Fowler)
- Una vez finalizada la fase **Verde**, se deberá etiquetar el último commit con el tag FASE_GREEN. Una vez finalizada la fase **Refactor**, se deberá etiquetar el último commit con el tag FASE_REFACTOR
- Fecha límite de entrega: **21 de diciembre de 2022 a las 23:59**. No se admitirán entregas fuera de plazo o por otra vía distinta a la indicada en estas normas.

Recopilación de problemas frecuentes

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

- No se consiguen altos niveles de cobertura de condición-decisión (complejidad).
- Demasiados tests para tan baja cobertura de clases poco complejas.
- Los tests en aislamiento no reproducen los tests que no son en aislamiento. Debe probarse lo mismo.
- No se consigue hacer funcionar los mocks.
- No se aplica “rama por tarea”.
- Los objetivos ant no funcionan bien o varios objetivos ant hacen lo mismo.