Práctica 2

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

Mario Corrales

Curso 2022-2023

Desiderata

En un sistema de información se requiere desarrollar los servicios de un gestor de reservas de alojamientos. A continuación se describen las características que se ha planificado incluir en la iteración actual:

En relación con las reservas y alojamientos, se desea que:

1. Un alojamiento tiene que tener un identificador, un nombre, una descripción y unas coordenadas geográficas.
2. El identificador de un alojamiento debe de ser único y debe tener entre 1 y 15 caractéres.
3. El nombre de un alojamiento es obligatorio y no puede estar vacío.
4. La descripción de un alojamiento es obligatoria y no puede estar vacía.
5. Las coordenadas son obligatorias y contienen latitud y longitud expresadas en grados decimales. La latitud debe ser un valor entre -90 y 90 y la longitud debe ser un valor entre -180 y 180.
6. Una reserva tiene que tener un identificador, un alojamiento, una fecha de creación, una fecha de modificación, un estado y un dni de la persona que reserva.
7. El identificador de una reserva debe de ser único y debe de tener entre 1 y 12 caractéres.
8. Estados posibles de una reserva: Realizada, Confirmada, Pagada o Cancelada.
9. El dni proporcionado debe cumplir con el formato definido en la sección *Formato DNI en España*.
10. Al crear una reserva, se debe proporcionar la fecha de creación, el alojamiento y el dni. El estado al crearse debe de ser Realizada y la fecha de modificación debe ser igual que la de creación.
11. Debe ser posible realizar algunas operaciones para cambiar el estado: Confirmar la reserva si ha sido Realizada. Pagar la reserva si ha sido Confirmada. Cancelar la reserva solo si está en el estado Confirmada o Pagada. Cuando se realize cualquiera de estas operaciones, se debe proporcionar la fecha de modificación.

Resto	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Letra	T	R	W	A	G	M	Y	F	P	D	X	B	N	J	Z	S	Q	V	H	L	C	K	E

Figura 1: Letra de control según el resto de la división del número de dni entre 23

12. Una reserva deberá poder ser comparada con otra usando la fecha de creación, indicando si es anterior, igual o posterior.
13. Se considerará que un alojamiento es igual a otro si coinciden los identificadores.
14. Se considerará que una reserva es igual a otra si coinciden los identificadores.
15. Se pueden modificar el alojamiento o el dni solo si la reserva está en el estado Realizada.

Además, se debe poder gestionar las reservas a través de un gestor, dónde se deben poder realizar las siguientes operaciones:

1. Añadir o eliminar reservas del gestor (no se pueden añadir reservas iguales).
2. Obtener cuántas reservas hay en un gestor.
3. Dado un dni en concreto, obtener todas las reservas que han sido realizadas con ese dni.
4. Obtener la fecha de las reservas más recientes contenidas en el gestor y la fecha de las más antiguas.
5. Obtener del gestor una lista en orden cronológico (de anterior a posterior) basado en la fecha de creación de las reservas.
6. Dada una fecha en concreto, obtener todas las reservas que han sido creadas el día de la fecha indicada.
7. Dadas dos fechas, obtener todas las reservas que han sido creadas en el intervalo de fechas dado (incluidas ambas).
8. Dado un estado de una reserva, obtener todas las reservas que están en el estado proporcionado.
9. Combinación de las opciones anteriores, es decir, poder indicar que incluya todas las reservas en un determinado estado y en una fecha, o todas las reservas en un determinado estado y en un intervalo de fechas.

Formato DNI en España

El número del documento nacional de identidad está formado por ocho dígitos y un carácter alfabético de control. Esta letra se obtiene a partir del número completo del DNI dividido entre el número 23. Al resto resultante de dicha división, que está comprendido entre 0 y 22, se le asigna la letra de control según una equivalencia (ver figura 1). No se utilizan las letras: I, Ñ, O, U. La I y la O se descartan para evitar confusiones con otros caracteres, como 1, l o 0. La Ñ se descarta para evitar confusiones con la N.

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 las clases de tests para realizar el diseño dirigido por las pruebas

(b) definiremos qué clases vamos a crear

(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.

Añada nuevos tests teniendo en cuenta las técnicas de caja negra de particiones basadas en el estado según se necesite.

Tests

Los tests serán implementados con JUnit. Se utilizará jUnit 5. 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*.

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.

Características del proyecto Eclipse

- El proyecto Eclipse deberá nombrarse con 'tds-practica2-grupox', siendo x el número del grupo asignado.
- Deberá ser un proyecto Eclipse válido para Eclipse IDE for Java developers release 2022-06 y jdk 11.
- La carpeta `src` del proyecto Eclipse contendrá los fuentes `.java` de las clases diseñadas aplicando TDD y *Test First*.
- Los stubs desarrollados por el equipo se alojará en un paquete Java que se nombrará (`tds.practica2`).
- Las clases de test deberán alojarse en un paquete llamado `tds.practica2.test`

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**. En el repositorio no se tendrá ningún `.class` 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 sí se valorará que se haya hecho. Es recomendable emplear el método rama por tarea explicado en clase.

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` deberá llamarse con el mismo nombre que el proyecto java.
- 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*.
- Cualquier *push* al repositorio una vez realizada la entrega será penalizado con 0 en la Práctica.
- Al entregar la práctica todo deberá quedar integrado en la rama master.
- 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 las librerías necesarias.
- 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.
- El proyecto Eclipse también deberá subirse a la tarea habilitada en aulas en formato comprimido.
- El proyecto deberá compilar y funcionar con jdk 11.
- 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-persona empleado en la realización de la práctica.
  - Clases que forman parte de la solución.
- El último commit debe de estar etiquetado con la tag **FASE_RED**
- Fecha límite de entrega: **25 de noviembre de 2022 a las 15:00 (hora española)**.
- 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 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.
- Los tests TDD no especifican todo el comportamiento en casos no válidos. Es decir, no hay tests TDD con assertThrow para indicar qué pasará cuando no se se puede ejecutar el método.
- 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 *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.
- 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.
- 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 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.
- Los tests en los que hay comprobación de valores de listas, conjuntos, arrays,... se hacen muy largos y no utilizan `assertArrayEquals` y definición de arrays literales para el resultado esperado.