Programación Modular y Orientación a Objetos - Laboratorio 8

Requisitos previos

El alumnado se desenvuelve correctamente en el entorno Eclipse y maneja adecuadamente la perspectiva Java para resolver problemas que requieren implementar varias clases relacionadas, entre las que hay tanto MAEs como TADs, y realizar las correspondientes pruebas unitarias con JUnit. Además, es capaz de realizar y comprender los diagramas de clases y de secuencia UML, y conoce los fundamentos teóricos del tratamiento de excepciones.

Objetivos

Este laboratorio sirve para afianzar los conocimientos adquiridos en clase sobre la identificación y tratamiento de situaciones excepcionales, y su objetivo consiste en reforzar todos los conceptos relativos a este tema.

Al finalizar este laboratorio, el alumnado deberá ser capaz de:

- Identificar las situaciones excepcionales que deban tenerse en cuenta para evitar problemas durante la ejecución de una aplicación Java.
- Implementar las excepciones que se anticipan a dichas situaciones excepcionales.
- Realizar las pruebas unitarias (JUnits) que permitan verificar que la solución implementada es correcta.

Motivación

Este laboratorio presenta un ejercicio sencillo en lo que se refiere a su diseño, pues consta solamente de tres clases. Sin embargo, requiere de la implementación y tratamiento de tres tipos de excepciones.

Tarea única: Roster de participantes

Esta actividad se va a centrar en un Roster (MAE), cuyo atributo principal es una ListaParticipantes (TAD), que contiene la información de los Participantes (TAD) de una determinada actividad que no interesa en este momento. La figura siguiente muestra el diagrama de clases que se va a implementar.

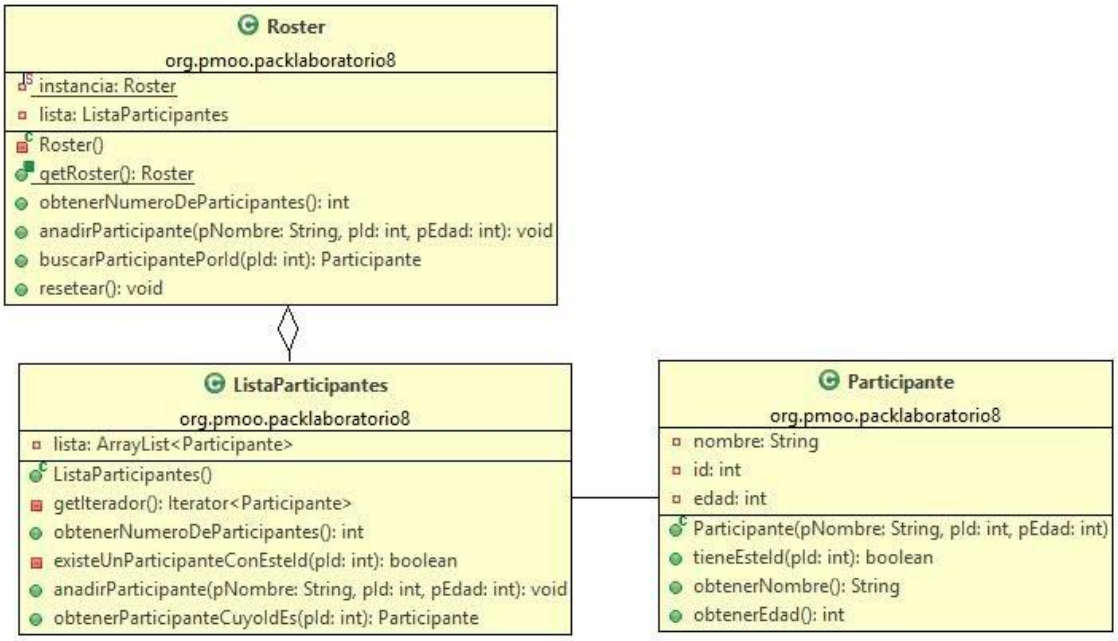

Figura 1 – Diagrama de clases para la gestión del Roster de participantes.

La aplicación almacena la información de los participantes. Por cada uno de ellos se guarda su nombre, un identificador, y la edad. Respecto a las funcionalidades que se quieren implementar, son muy básicas, e incluyen, por ejemplo, la búsqueda y alta de participantes en el Roster.

Se pide **implementar las clases siguientes**, cuyos atributos, métodos y tratamientos de excepciones asociadas se describen a continuación, **así como las pruebas unitarias (JUnit)** que permitan verificar su corrección.

- Clase Participante
  - **Atributos:** el nombre (*String*), un identificador (*int*), y la edad (*int*).
  - **Método constructor:** recibe como parámetro los valores con los que inicializar los atributos. El método constructor lanza una MenorDeEdadException si el valor del parámetro pEdad es menor de 18.
  - Otros métodos:
    - *tieneEsteId()* devuelve un booleano que indica si el identificador del participante es igual al identificador que recibe como parámetro.
    - *obtenerNombre()* devuelve el nombre del participante.
    - *obtenerEdad()* devuelve la edad del participante.

- Clase ListaParticipantes
  - **Atributo:** una lista de participantes, implementada como ArrayList.
  - **Método constructor:** crea una instancia vacía de ArrayList<Participante> y se la asigna al atributo lista.
  - Otros métodos:
    - *obtenerNumeroDeParticipantes()* devuelve el número de elementos que contiene la lista de participantes.
    - *getIterador()* devuelve un elemento de tipo Iterator para que los métodos de la clase ListaParticipantes puedan recorrer la lista con él. Como siempre, este método será privado.
    - *existeUnParticipanteConEsteId()* devuelve un booleano que indica si existe un participante en la lista cuyo identificador es igual al que recibe como parámetro. Se trata de un método privado, que solamente se utiliza dentro de la clase ListaParticipantes.
    - *anadirParticipante()* añade a la lista un participante cuyo nombre, identificador y edad son los que recibe como parámetros. En la lista no puede haber dos participantes con el mismo identificador. Por este motivo, el método anadirParticipante() lanza una YaExisteIDException si en la lista ya existe un participante cuyo identificador es el que se ha recibido como parámetro. Si se diera este caso, entonces NO añadiría ningún nuevo participante a la lista. En este método no se realiza el tratamiento de la excepción MenorDeEdadException.
    - *obtenerParticipanteCuyoIdEs()* devuelve el participante de la lista cuyo identificador es igual al que recibe como parámetro. Este método lanza una NoEncontradoException si no existe tal participante.
- Clase Roster (MAE)
  - **Atributos:** una lista de participantes (*ListaParticipantes*), y la instancia (única) de Roster.
  - **Método constructor:** inicializa el atributo de tipo ListaParticipantes. Se trata de un método privado, acorde a lo que dicta el patrón Singleton.
  - Otros métodos:
    - *getRoster()* devuelve la única instancia de Roster para que pueda ser accesible desde cualquier otra clase.
    - *obtenerNumeroDeParticipantes()* devuelve el número de participantes que tiene la lista de participantes del Roster.
    - *buscarParticipantePorId()* devuelve el participante que tiene como identificador el valor que se recibe como parámetro. Si no existe tal particante, entonces se devuelve el valor null.
    - *anadirParticipante()* intenta añadir a la lista de participantes del Roster un nuevo participante, cuyo nombre, identificador y edad son los que se reciben como parámetro (i.e. pNombre, pId, y pEdad). Si se captura una MenorDeEdadException, entonces no se hace nada, esto es, no se añade ningún participante a la lista. Este método también realiza el tratamiento de YaExisteIdException. En este caso, si al intentar añadir

a la lista de participantes del Roster un participante cuyo id ya existe (esto es, el identificador está asignado a otro participante – recuérdese que no se permiten identificadores repetidos), entonces lo que se hará es intentar añadir un participante con mismo nombre (pNombre) y misma edad (pEdad), pero con identificador igual a pId+1. Si también existiera ya en la lista de participantes del Roster un participante con identificador pId+1, entonces se intentaría con el identificador pId+2.... y así sucesivamente hasta que en algún momento se encuentre un valor pId+x que no esté asignado a ningún participante de la lista. La idea es que, de este modo, al final tiene que llegar un momento en el que se pueda añadir un nuevo participante {pNombre, pId+x, pEdad}, donde pId+x es el primer identificador, mayor o igual que pId, que está "libre".

- *resetear()* vacía la lista de participantes del Roster. La utilidad de este método quedará restringida al ámbito de las pruebas unitarias.

- Clases MenorDeEdadException, NoEncontradoException, y YaExisteIdException

Estas clases serán necesarias para gestionar el tratamiento de las situaciones excepcionales descritas anteriormente.