EDA (ETS de Ingeniería Informática). Curso 2015-2016

Práctica 1. Elementos para el diseño y uso de una EDA en Java

Sesión 1: La Cola de Prioridad de las Urgencias de un Hospital

Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València

1. Objetivos de la práctica

El principal objetivo de esta práctica es que el alumno aplique al diseño de una aplicación concreta los conceptos Java para Estructuras de Datos (EDAs) estudiados en el Tema 1 de la asignatura. Específicamente, el alumno deberá ser capaz de implementar y utilizar eficazmente la jerarquía Java de la *Cola de Prioridad* que usa una aplicación de simulación de un hospital, sala de urgencias incluida.

Al mismo tiempo, como objetivo subsidiario, la realización de esta práctica permitirá al alumno crear y manejar la estructura básica de librerías de usuario *BlueJ* en la que, de forma incremental, irá ubicando las distintas clases Java que se desarrollen durante el curso.

2. Descripción del problema

De una forma muy simplificada, el funcionamiento de un hospital se podría describir como sigue:

1. Cuando un paciente llega a un hospital entra en su sala de urgencias y espera a ser atendido.
2. En la sala de urgencias solo hay un médico que, cada cierto tiempo, sale a atender al paciente de la sala con mayor prioridad. La prioridad de un paciente viene dada por el estado de gravedad que reviste y por su edad, en base a los criterios que se especifican más adelante.
3. Tras tratarlo, el médico de urgencias da el alta a aquel paciente cuyo estado de gravedad sea leve o muy leve, pues sana; sin embargo, debe ingresarlo en el hospital si reviste un estado de gravedad mayor que leve, para que sea tratado adecuadamente.

2.1. Las clases de una aplicación de simulación de un hospital

Se ha implementado -parcialmente- una aplicación Java que modela, siguiendo las anteriores indicaciones, el funcionamiento de un hospital; las principales clases que componen tal aplicación son:

- `Paciente`, que representa a un enfermo que llega al hospital. Un paciente `TIENE UN` nombre que lo identifica, una cierta edad y un estado, un número entero que representa la gravedad que reviste mediante uno de los siguientes valores:

Valor	Estado
0	fallecido
1	crítico
2	grave
3	moderado
4	leve
5	muy leve
6	sano

Es importante resaltar que el método `compareTo` de la clase `Paciente` es el que permite establecer cuál de dos pacientes dados debe ser atendido con mayor prioridad: aquel cuyo estado revista mayor gravedad, siendo 1 el estado más grave y 5 el menos grave (no llegan pacientes fallecidos ni sanos a urgencias); en el caso en que el estado de ambos pacientes sea el mismo, entonces dará mayor prioridad al paciente que sea niño (<15 años), al más pequeño si ambos lo son, y luego al paciente que sea anciano (>65), al más mayor si ambos lo son.

- `Urgencias`, que representa la sala de urgencias de un hospital como un conjunto de sillas donde los pacientes esperan a que se les atienda, considerando los criterios de prioridad establecidos, en el menor tiempo posible.

Figura 1: Ejemplo de la sala de urgencias de un hospital

Así, la clase `Urgencias` `TIENE UN` *array* de `Paciente` donde cada posición representa una silla y su contenido al paciente que la ocupa, *null* si está vacía. Además, lo que es más importante, para gestionar la atención a los pacientes por orden de prioridad, i.e. para atender en cada momento al paciente de mayor prioridad, esta clase `TIENE UNA` *Cola de Prioridad* de `Paciente`; en la siguiente sección se recuerda la definición de esta EDA de Búsqueda Dinámica y se define su modelo Java.

- `Hospital`, que representa un hospital como un conjunto de camas donde se encuentran los pacientes para recibir el tratamiento adecuado.

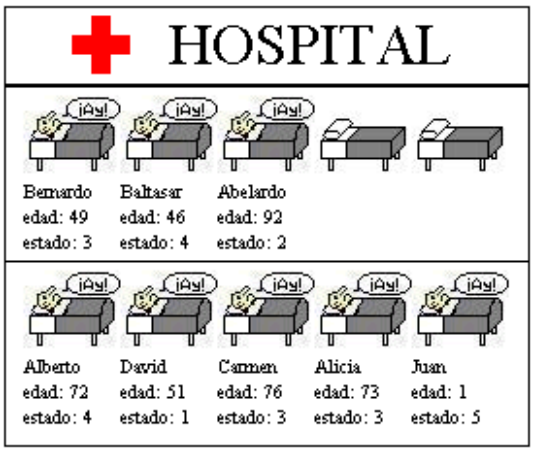

Figura 2: Ejemplo del estado de una planta de un hospital

Así, la clase `Hospital` `TIENE UN` *array* de `Paciente` donde cada posición representa una cama y su contenido al paciente que la ocupa, *null* si está vacía.

2.2. *Cola de Prioridad*: modelo Java y su papel en la simulación de las urgencias de un hospital

Una *Cola de Prioridad* es una colección homogénea de datos que solo se puede gestionar accediendo al dato que tiene la máxima prioridad. Para ello, nótese, es indispensable que cualquier dato de la colección, *d1*, sea *Comparable* con otro, *d2*, en base a su prioridad:

$$d1 < d2 \quad \texttt{SII} \quad \text{la prioridad de } d1 \text{ es mayor estricta que la de } d2$$

En base a este criterio de comparación resulta fácil deducir que el dato de mayor prioridad de la colección será el que mínimo tiempo de espera requiera para ser tratado o, equivalentemente, que acceder al dato más prioritario de una colección equivale a buscar su mínimo.

En el caso de datos con igual prioridad, en principio, se accede a ellos según un criterio *FIFO*, es decir, se accede al primero que se insertó en la *Cola de Prioridad*. Por todo ello, el comportamiento o modelo de gestión de una *Cola de Prioridad* se puede describir en Java como sigue:

```
public interface ColaPrioridad<E extends Comparable<E>> {

    /** Atendiendo a su prioridad, inserta el Elemento e en una Cola de Prioridad (CP) **/
    void insertar(E e);
    /** SII !esVacia(): obtiene y elimina el Elemento con máxima prioridad de una CP **/
    E eliminarMin();
    /** SII !esVacia(): obtiene el Elemento con máxima prioridad de una CP **/
    E recuperarMin();
    /** Comprueba si una Cola de Prioridad está vacía **/
    boolean esVacia();
}
```

Para familiarizarse con el papel que juega la EDA *Cola de Prioridad* en la simulación de las urgencias de un hospital, con el modelo de gestión de datos que representa, en la carpeta *Práctica 1* de *Recursos* de *PoliformaT* está disponible una aplicación gráfica (*simuladorClass.jar*) que simula el funcionamiento de un hospital de 20 camas y a cuyas urgencias acuden 50 pacientes durante 12 horas (de las 8:00 a las 20:00). Para ejecutar esta aplicación, en *Windows* basta un simple *click* sobre su nombre o icono, mientras que en *Linux* habrá que ejecutar el comando `java -jar simuladorClass.jar`.

3. Actividades en el laboratorio

Al tratarse de la primera sesión de prácticas, antes de llevar a cabo las actividades que se proponen en este apartado es necesario organizar el espacio de trabajo tal y como se muestra en la siguiente figura:

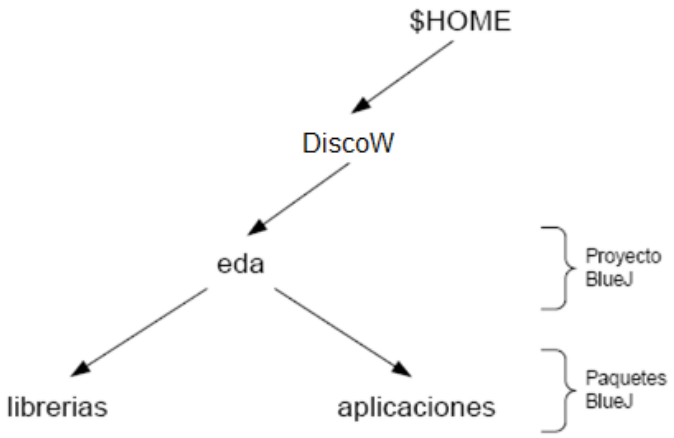

Figura 3: Estructura del espacio de trabajo

Esto es, en primer lugar se debe crear un proyecto *BlueJ* denominado *eda* en el subdirectorio *DiscoW* del *HOME* del usuario. A su vez, dentro de este proyecto se deben crear dos nuevos paquetes *BlueJ*, *librerias* y *aplicaciones*, en los que se irán ubicando las clases asociadas a las distintas prácticas que se realicen este curso. Específicamente,

- en *librerias* se situarán los paquetes *BlueJ* con las clases de utilidades, excepciones y estructuras de datos;
- en *aplicaciones* se situarán los paquetes *BlueJ* con las clases de aplicaciones específicas.

3.1. Creación de una librería para definir los modelos de las EDAs

En esta práctica se van a utilizar por primera vez dos modelos de EDAs: el de la *Cola de Prioridad*, que se usará para gestionar la atención a los pacientes de las urgencias de un hospital, y el de la *Lista Con Punto de Interés*, que se utilizará para diseñar una Implementación Lineal de una *Cola de Prioridad*. Estos modelos están definidos en Java mediante las interfaces `ColaPrioridad` y `ListaConPI` respectivamente, disponibles en *PoliformaT*. El alumno deberá añadir ambas interfaces a un nuevo paquete denominado *librerias.estructurasDeDatos.modelos* y, luego, compilarlas.

3.2. Implementación de los modelos `ColaPrioridad` y `ListaConPI`

Se puede conseguir una implementación eficiente de la interfaz `ListaConPI` utilizando una Lista Enlazada Genérica (LEG). Esta clase, que se diseñara en la próxima sesión de prácticas, recibe el nombre de `LEGListaConPI` y su correspondiente `.class` se encuentra disponible en *PoliformaT*. Para poder utilizarlo, dado que *BlueJ* solo permite añadir ficheros con extensión `.java`, el alumno deberá crear primero el paquete *BlueJ* que, en adelante, contendrá las Implementaciones Lineales de las EDAs que se diseñen: *librerias.estructurasDeDatos.lineales*; hecho esto, deberá salir de *BlueJ* y descargar el fichero `LEGListaConPI` en la carpeta o directorio del mismo nombre; finalmente, al entrar de nuevo en *BlueJ* y situarse en el paquete *librerias.estructurasDeDatos.lineales*, aparecerá el icono de la clase `LEGListaConPI` pero con el texto `(no source)` en su parte inferior para indicar que no corresponde a un fichero fuente y que, por tanto, no debe ser compilado sino solamente ejecutado.

El alumno también deberá situar y compilar el fichero `NodoLEG.java` disponible en *PoliformaT* en el paquete *librerias.estructurasDeDatos.lineales*, pues contiene la clase de los nodos de una LEG que usa `LEGListaConPI`.

Como ejercicio de implementación se plantea el diseño de dos Implementaciones de la interfaz `ColaPrioridad`, dejándose para la próxima sesión su comparación en base a criterios de eficiencia:

- La clase `LPIColaPrioridad` que implementa la interfaz haciendo uso, vía Herencia, de los métodos de la clase `ListaConPI`; la idea es mantener todos los elementos de la *Lista* ordenados de menor a mayor, de manera que el elemento más prioritario sea siempre el primero.

  Esta clase se encuentra parcialmente implementada en *PoliformaT* y, al tratarse de una Implementación Lineal de `ColaPrioridad`, se añadirá al paquete *librerias.estructurasDeDatos.lineales*.

- La clase `PriorityQColaPrioridad` que implementa la interfaz haciendo uso, vía Herencia, de los métodos de la clase `PriorityQueue` de la jerarquía `java.util.Collection` del estándar de Java. Como se puede leer en la documentación del API de Java, dicha clase representa una *Cola de Prioridad* mediante un Montículo Binario (*Heap*), un tipo particular de Árbol Binario que se estudiará en el Tema 5 de la asignatura.

  El esqueleto de la clase `PriorityQColaPrioridad` se encuentra en *PoliformaT* y deberá añadirse al nuevo paquete *librerias.estructurasDeDatos.jerarquicos*, que en adelante contendrá todas las Implementaciones Jerárquicas, basadas en un Árbol, de las EDAs que se diseñen.

3.3. Modelado de un hospital

Las tres clases que se utilizan para modelar el funcionamiento de un hospital (`Paciente`, `Urgencias` y `Hospital`) se encuentran disponibles en *PoliformaT* y se deberán incluir en un nuevo paquete *Bluej* del proyecto *eda* denominado *aplicaciones.hospital*.

El alumno deberá completar el método `compareTo` de la clase `Paciente` que, como se ha comentado anteriormente, se utiliza para establecer la prioridad de dicho paciente de acuerdo a su estado de gravedad y edad.

3.4. Validación de la práctica

El alumno dispone en *PoliformaT* del fichero `TestUrgencias.class`, que le permitirá verificar la corrección de los distintos métodos que ha implementado durante la sesión: el `compareTo` de la clase `Paciente` y los métodos de las clases `LPIColaPrioridad` y `PriorityQColaPrioridad`. Este fichero se debe descargar en la carpeta o directorio *aplicaciones.hospital*, tras salir de *BlueJ* pues se trata de un `.class`; además, antes de ejecutarlo, el alumno debe comprobar que la estructura de paquetes y ficheros que ha creado en su proyecto *BlueJ eda* es la que muestra la siguiente figura:

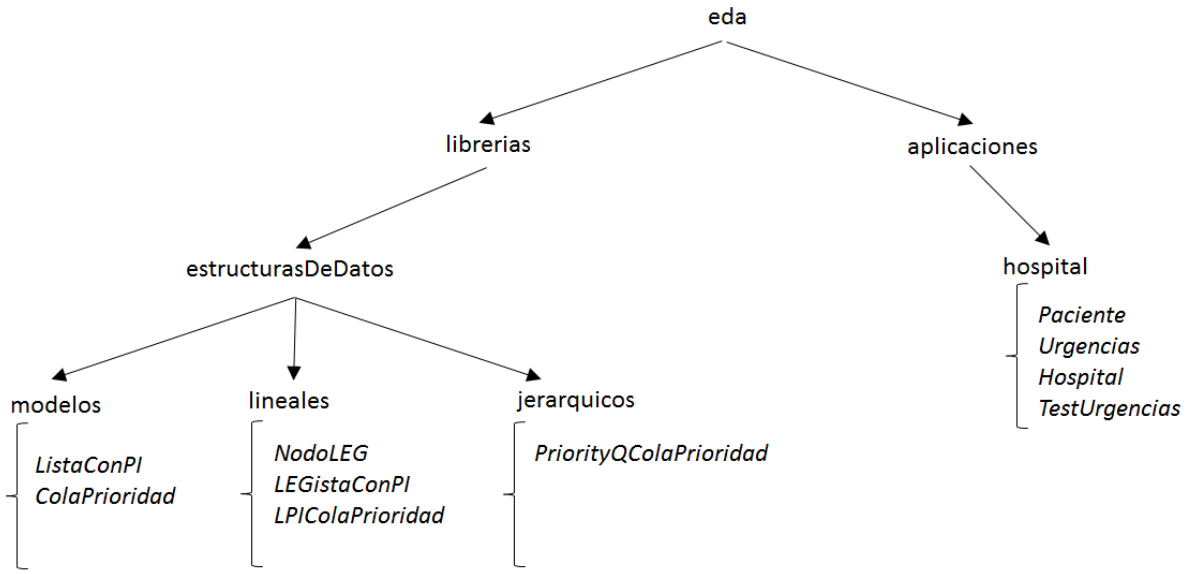

Figura 4: Estructura del proyecto *BlueJ eda* tras la primera sesión de la Práctica 1