EDA (ETS de Ingeniería Informática). Curso 2020-2021

Práctica 1. La Lista Con Punto de Interés de una aplicación de apuestas de La Primitiva: implementación y uso (1 sesión)

Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València

1. Objetivos

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, al acabar esta práctica el alumno deberá ser capaz de implementar y utilizar eficazmente la jerarquía Java de la *Lista Con PI* (*Punto de Interés*) que usa una aplicación de apuestas de La Primitiva.

Al mismo tiempo, al realizar esta práctica, el alumno deberá ser capaz de 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

La Lotería Primitiva es un juego de azar regulado por Loterías y Apuestas del Estado en el que cada apuesta consiste en elegir seis números distintos entre el 1 y el 49 (combinación); básicamente, una apuesta resultará premiada si coincide con la combinación ganadora del sorteo correspondiente.

En la actualidad existen distintas aplicaciones para jugar *online* a La Primitiva aunque, obviamente, todas ellas comparten una misma funcionalidad básica: hacer una apuesta. Precisamente por ello, en esta práctica el alumno abordará la implementación de una sencilla aplicación de apuestas de La Primitiva que tan solo permite realizar una apuesta aleatoria, i.e. seleccionar de forma aleatoria seis números distintos entre el 1 y el 49, y almacenarlos en una *Lista Con PI* a modo de resguardo virtual; a petición del usuario, los números de la apuesta pueden almacenarse, bien en orden de generación, bien en orden Ascendente.

Las clases de la aplicación

Siguiendo las anteriores indicaciones, las dos clases de la aplicación a implementar son:

- `NumeroPrimitiva`, que representa un número seleccionado aleatoriamente entre el 1 y el 49. Por ello, un `NumeroPrimitiva TIENE UN int numero`.
  En cuanto a los métodos de esta clase, cabe señalar ahora que...
  - Como un `NumeroPrimitiva` puede figurar en una lista ordenada Ascendentemente, la clase debe sobrescribir el método `compareTo` de la interfaz `Comparable` como sigue: devuelve un `int` negativo si `this NumeroPrimitiva` tiene un valor menor que el de `otro`, un `int` positivo si `this` tiene un valor mayor que el de `otro` y `0` si los valores de `this` y `otro` coinciden.
  - Como los números de una combinación de La Primitiva tienen que ser distintos por definición, la clase debe sobrescribir el método `equals` de la clase `Object` como sigue: devuelve `true` si `this NumeroPrimitiva` tiene el mismo valor que `otro`, o `false` en caso contrario.
- `ApuestaPrimitiva`, que representa una apuesta aleatoria de La Primitiva; por ello, una `ApuestaPrimitiva TIENE UNA ListaConPI combinacion` de seis `NumeroPrimitiva`, distintos y generados aleatoriamente.

  Cabe destacar ahora que el método constructor de esta clase tiene un parámetro `boolean ordenada` para determinar si los números de la `ApuestaPrimitiva` a crear deben ser almacenados en orden Ascendente (`ordenada == true`) o no (`ordenada == false`); para conseguirlo, la `ListaConPI combinacion` se implementará mediante, respectivamente, una `LEGListaConPIOrdenada` o una `LEGListaConPI`.

  Así mismo, dado que los números de una apuesta deben ser distintos, cada vez que se genera aleatoriamente un nuevo número de la apuesta se debe comprobar que aún no figura en la combinación actual. Para ello, se tiene que invocar a un método auxiliar de la clase denominado `posicionDe`; en concreto, y asumiendo

que el primer elemento de una combinación está en su posición 0 y el último en la 5, este método devuelve la posición de un `NumeroPrimitiva n` dado en una `ApuestaPrimitiva`, o -1 si `n` no forma parte de la combinación actual.

3. Actividades a realizar

Al tratarse de la primera sesión de prácticas, antes de llevar a cabo las actividades que se proponen en este apartado es necesario que el alumno cree y organice tal y como se muestra en la Figura 1 el proyecto *BlueJ* que usará (**exclusivamente**) para las prácticas de la asignatura.

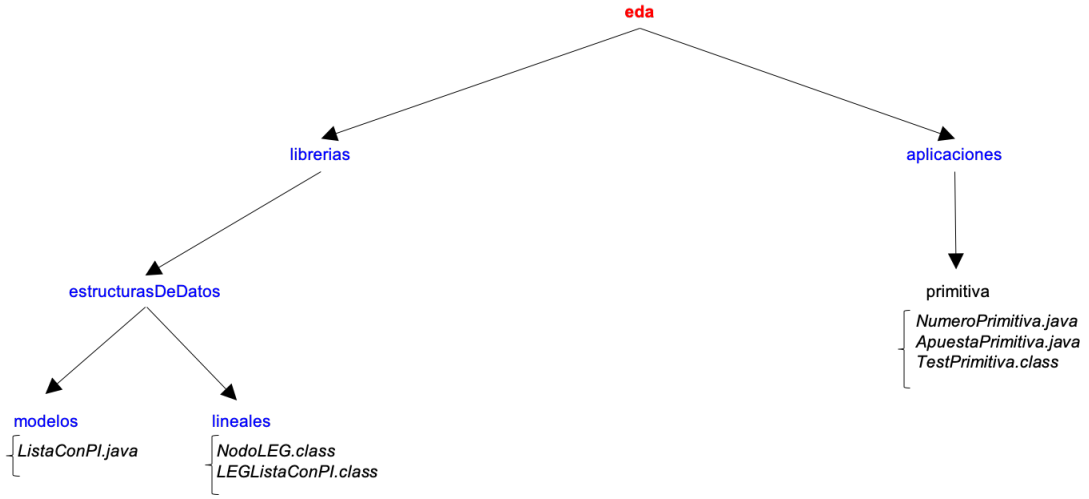

Figura 1: Estructura del proyecto *BlueJ* de prácticas del alumno

Para ello, debe llevar a cabo los siguientes pasos, en el orden en el que se indican:

- Tras comprobar que la versión de *BlueJ* instalada en su ordenador es la 4.2.2, crear un proyecto *BlueJ* denominado *eda* en una carpeta de un disco local de su ordenador.
- Dentro del proyecto *eda*, crear dos nuevos paquetes denominados *librerias* y *aplicaciones*. En adelante, en *librerias* se situarán los paquetes que contienen las clases de utilidades y estructuras de datos que se usarán en las prácticas de la asignatura, mientras que en *aplicaciones* estarán los paquetes con las clases de las aplicaciones específicas que las usan.
- Abrir el paquete *aplicaciones* y crear en él un nuevo paquete de nombre *primitiva*, que contendrá las clases de la aplicación a desarrollar en esta práctica.
- Abrir el paquete *librerias* y crear en él un nuevo paquete de nombre *estructurasDeDatos*.
- Abrir el paquete *estructurasDeDatos* y crear en él dos nuevos paquetes: *modelos* y *lineales*. En adelante, en estos paquetes se ubicarán, respectivamente, los modelos Java (interfaces) y las Implementaciones Lineales que se realicen de ellos en las prácticas de la asignatura.
- Salir de *BlueJ* **seleccionando la opción *Salir* de la pestaña *Proyecto***.
- Descargar los ficheros disponibles en *PoliformaT* en sus correspondientes directorios, tal como se indica en la Figura 1: `ListaConPI.java` en el directorio *librerias.estructurasDeDatos.modelos*; `NodoLEG.class` y `LEGListaConPI.class` en *librerias.estructurasDeDatos.lineales*; etc.
- Entrar en el proyecto *BlueJ eda* y compilar la única clase de su paquete *librerias.estructurasDeDatos.modelos*.
- Cerrar el paquete *modelos*, preferiblemente seleccionando la opción *Cerrar* de la pestaña *Proyecto*.
- Abrir el paquete *librerias.estructurasDeDatos.lineales*.

3.1. Implementar la clase `LEGListaConPIOrdenada`

En esta actividad el alumno debe diseñar en el subpaquete *lineales* la clase `LEGListaConPIOrdenada`, una nueva Implementación Enlazada de la interfaz `ListaConPI` que mantiene ordenados Ascendentemente los elementos de la lista. Como la diferencia fundamental entre una lista ordenada y una no ordenada es la operación de inserción, la forma más efectiva de implementar esta nueva clase es definirla como una Derivada de la clase `LEGListaConPI` existente y sobrescribir su método `insertar` para que los elementos de la lista se inserten en orden (ascendente), en lugar de antes de su PI.

Es importante señalar ahora las dos cuestiones básicas que el alumno debe tener (muy) en cuenta a la hora de implementar la clase `LEGListaConPIOrdenada`:

- Los elementos de una lista solo se pueden ordenar si son comparables. Por tanto, el tipo genérico de los elementos de la clase debe restringirse a aquellos que implementan la interfaz `Comparable` y, consecuentemente, se debe usar el método `compareTo` para comparar este tipo de elementos.
- La sobrescritura del método `insertar` heredado de la clase base `LEGListaConPI` se debe realizar usando única y exclusivamente los métodos de la interfaz `ListaConPI`, pues solo así se puede hacer independiente de los detalles de implementación que "se ocultan" en la clase base.

3.2. Completar la implementación de las clases de la aplicación de apuestas

Para llevar a cabo esta actividad, en primer lugar el alumno debe abrir el paquete *aplicaciones.primitiva* del proyecto *BlueJ eda*. Luego, siguiendo los comentarios que aparecen en el código de las clases de este paquete, así como la descripción que de ellas se ha realizado en el apartado 2 de este boletín, debe: completar la cabecera de la clase `NumeroPrimitiva` y los cuerpos de sus métodos `equals` y `compareTo`; completar los cuerpos de los tres métodos de la clase `ApuestaPrimitiva`.

3.3. Validar el código desarrollado en la práctica

Para comprobar la corrección del código que ha implementado durante la sesión, el alumno debe realizar dos pruebas usando las clases del paquete *aplicaciones.primitiva* de su proyecto *BlueJ eda*. En la primera de ellas, debe crear dos objetos de tipo `ApuestaPrimitiva` en el *Object Bench* de *BlueJ*, `apuesta` y `apuestaOrd`, ejecutando el constructor de la clase con argumentos `false` y `true` respectivamente. Luego, usando el *Code Pad* de *BlueJ*, debe aplicar a cada uno de ellos el método `toString` para comprobar que, tal como muestra la Figura 2, si bien ambos objetos contienen seis números distintos entre el 1 y el 49, los de `apuestaOrd` están ordenados Ascendentemente -pues ha sido creado usando una `LEGListaConPIOrdenada`- y los de `apuesta` no -pues ha sido creado usando una `LEGListaConPI`.

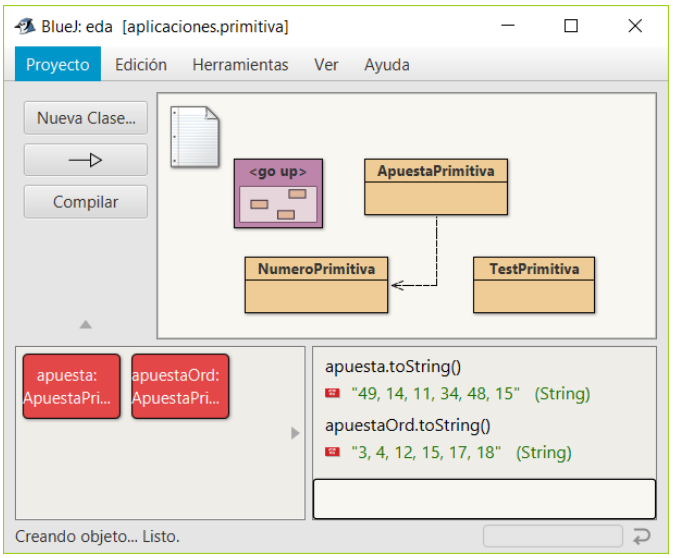

Figura 2: Generación de dos apuestas de La Primitiva

La segunda prueba que debe realizar el alumno es ejecutar el método `comprobar` de la clase `TestPrimitiva`; al hacerlo, se comprueba mediante una batería de tests el correcto funcionamiento de la clase `LEGListaConPIOrdenada`, los métodos `equals` y `compareTo` de `NumeroPrimitiva` y, finalmente, los métodos constructor, `posicionDe` y `toString` de `ApuestaPrimitiva`.