Ejercicios a resolver [máx: 10p.]

1. laberinto [resuelto/0p]

Dada una cuadricula con N filas y N columnas, cuyas casillas pueden estar bloqueadas o no, averiguar si es posible ir desde la casilla (0, 0) hasta la casilla (N-1, N-1) sin pasar por ninguna casilla bloqueada.

2. cercano_mas_alto

Dado un grafo *G*, un nodo $x \in G$, y un array H que indica la altura de cada nodo de G, encontrar el nodo de mayor altura de G. Si hay varios esa altura, seleccionar el más cercano de X (menos aristas a atravesar para llegar a él desde *x*).

3. laberinto_camino_salida

Dada una cuadricula con N filas y N columnas, cuyas casillas pueden estar bloqueadas o no, y dos enteros 0 <= A, B < N, averiguar si es posible ir desde la casilla (0, A) hasta la casilla (N-1, B) sin pasar por ninguna casilla bloqueada.).

4. list_iterator

Implementar una clase cuyas instancias sean iteradores sobre una *singly-linked-list*, para recorrer los nodos de la lista de principio a fin.

Conforme al contrato estándar en Java, un iterador debe tener también un método *void remove()* que retira de la lista asociada el último elemento devuelto por una llamada a *next.* Pero la implementación por defecto de ese método genera una excepción *NoSuchElementException*. En este ejercicio se pide una implementación adecuada de esa operación.

5. tree_iterator [preorder]

Implementar una clase cuyas instancias sean iteradores sobre una *árbol binario de enteros*, para recorrer los nodos del árbol en *preorden* (obsérvese que el resuelto en el aula implementaba un recorrido en *inorden).*

Observaciones

El proyecto *eG-Semana-07* incluye dos carpetas con programas *java*: *src* y *aula*. En *src* está el código correspondiente a los ejercicios. En *aula* los ejemplos y ejercicios del aula. Además también contiene:

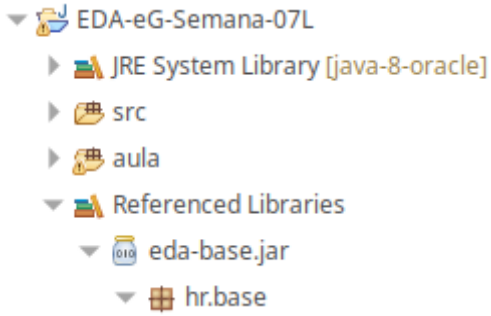
- Una carpeta de nombre *test_cases*. Dentro de ella hay otra subcarpeta por cada ejercicio, con dos subcarpetas (*input* y *output),* y dos archivos (*InputFormat* y *OutputFormat*). Los archivos de ambas están emparejados alfabéticamente. Cada pareja describe un caso de prueba: los datos a usar (input) y los resultados esperados (output)
- Un *paquete* por cada ejercicio, con el nombre del ejercicio. Dentro de cada uno hay dos clases: App y My_Code. En la segunda está el método o clase a implementar. La primera es ejecutable y contiene el código para leer los casos de prueba incluidos en la carpeta test_cases, encargándose de usar como corresponda el código de la solución propuesta.
- Una carpeta de nombre *resultados.* Puede usarse para ver con detalle la salida de cada caso de prueba.

Los casos de prueba visibles en el proyecto no cubren exhaustivamente todos los escenarios deseables. Es importante comprobar la corrección de la solución propuesta: bien analizando y revisando atentamente su código, o bien incluyendo otros casos de prueba que se consideren significativos. Para ello, habrá que revisar los casos de muestra incluidos y las descripciones en los archivos (*InputFormat* y *OutputFormat*) correspondientes a cada ejercicio.

Clases disponibles

El proyecto está configurado de manera que estén disponibles las clases usadas habitualmente en ejercicios anteriores: IntSinglyLinkedListNode, IntFIFOQueue, IntBinaryTreeNode…

Para consultar el código fuente de esas clases, en *Package Explorer* desplegar el proyecto > Referenced Libraries > eda-base.jar > hr.base



Alternativamente, declarar una variable de la clase deseada:

```
SinglyLinkedListNode x;
```

y después pinchar el nombre de la clase y pulsar F3.

El ejercicio resuelto (salida de un laberinto) hace uso de algunas clases que ineteresará/será necesario usar en otros:

- FIFOQueue<T> es similar a la cola genérica vista en el aula, pero con un nombre más sencillo
- GridCoordinates: sus instancias (objetos) representan coordenadas de casillas en una cuadrícula.
- GridIterator: iteradores para recorrer las casillas de una cuadrícula que están encima/debajo/izquierda/derecha de una casilla seleccionada.