Laboratorio A.E.D.

Lars-Åke Fredlund
lfredlund@fi.upm.es
Tonghong Li
tonghong@fi.upm.es
Manuel Carro Liñares
mcarro@fi.upm.es
Germán Puebla Sánchez
german@fi.upm.es
Pablo Nogueira
pnogueira@fi.upm.es

Viernes 11:00-13:00

Entrega

- La fecha límite para optar a la máxima nota es
  Hoy, Viernes 19 de octubre de 2012, a las 13:00 horas
- El fichero que hay que subir es
  `IterateNodePositionList.java`
- La entrega se hace a través de la siguiente URL:
  `http://lml.ls.fi.upm.es/~entrega`
- El paquete `iterateNodePositionList` esta documentado con Javadoc en
  `http://babel.ls.fi.upm.es/~fred/courses/aed/iterateNodePositionList/`
- El proyecto debe compilar sin errores, cumplir la especificación y pasar el `Tester`.

Configuración

- Arrancad Eclipse.
- Cread un paquete `iterateNodePositionList` en el proyecto `aed`, dentro de `src`.
- Aula Virtual → AED → Sesiones de laboratorio → Laboratorio 4 → codigo_lab4.zip (formato zip).
- Importad al paquete `iterateNodePositionList` las fuentes que habéis descargado
- Ejecutad `Tester`. Veréis que lanza una excepción:

```
Testing nth...
Exception in thread "main" java.lang.NullPointerException
at iterateNodePositionList.Tester.doTest(Tester.java:45)
at iterateNodePositionList.Tester.main(Tester.java:31)
```

Consejos

- Haced un primer diseño del algoritmo en papel, abstrayendo los detalles menores
- Es muy útil dibujar las estructuras de datos que uséis
- Simulad el algoritmo en papel:
  - ¿qué pasa con los variables durante los bucles?
  - ¿qué pasa con las estructuras de datos?
- Si falla algo:
  - Para depurar vuestro código os será **muy** útil imprimir los valores de las variables del programa:

    ```
    System.out.println("The current element is "+currPos.element());
    ```

  - o usar el "debugger" de Eclipse
  - y, obviamente, revisar el codigo

Tareas para hoy

- Hoy trabajaremos otra vez con el API `PositionList` del paquete `net.datastructures`, pero esta vez vamos a usar "iteradores" para recorrer las listas.
- **¡Es obligatorio usar iteradores para solucionar las tareas de este laboratorio!**
- No se puede llamar `first()`,`last()`, `next(p)` o `prev(p)` del API `PositionList`. Ni tampoco se puede usar los atributos de la clase `NodePositionList`, es decir `header`, `numElts` o `trailer`. Tampoco se puede trabajar con instancias de la clase `DNode`.
- En cambio, es **obligatorio** llamar al método `iterator()` para devolver un iterador, y usar estos iteradores para recorrer la listas.
- Se pide completar los métodos `nth(int n)` y `unique()` de la clase `IterateNodePositionList`. Esta permitido añadir nuevos métodos o variables locales.

Tareas para hoy (2)

- El método `E nth(int n)` debe devolver el elemento dentro el nodo `n`-ésimo de la lista, siempre que `n` sea menor o igual que la longitud de la lista. En caso contrario debe lanzar la excepción `BoundaryViolationException`. Puede asumirse que `n` será siempre mayor que cero.
- Asumimos que el orden de elementos dentro la lista es dado por el método `iterator()`. Es decir, el primer elemento devuelto por el iterador es considerado el primer elemento de la lista, etc.
- Ejemplo: Si `l` es una lista con los elementos $10, 20, 15, 4, 5$:
  - `l.nth(1)` debe devolver el nodo con el elemento 10.
  - `l.nth(3)` debe devolver el nodo con el elemento 15.
  - `l.nth(6)` debe lanzar la excepción `BoundaryViolationException`.
- Es **obligatorio** llamar al método `iterator()` para devolver un iterador, y usar este iterador para recorrer la lista.

Tareas para hoy (3)

- `d = l.unique()` devuelve en `d` una lista sin los elementos duplicados que pudiesen aparecer en `l`. Para los elementos duplicados, `unique()` debe preservar la primera ocurrencia del elemento (la más cercana al principio de la lista).
- Ejemplos:
  - Si `l` es 0,3,1,2,3 entonces la llamada `l.unique()` debe devolver una **nueva** lista con los elementos 0,3,1,2.
  - Si `l` es una lista vacía, `l.unique()` debe devolver una **nueva** lista vacía.
  - Si `l` es 0,3,9,0,2,9,0 la llamada `l.unique()` debe devolver una **nueva** lista 0,3,9,2.
- Para comparar dos elementos `e1` y `e2` se puede usar la comparación `e1.equals(e2)` que devuelve `true` si son iguales y `false` si no son iguales.
- Es **obligatorio** llamar al método `iterator()` para devolver iteradores, y usar estos iteradores para recorrer la listas.
- El método no debe cambiar la lista sobre la que se invoca (accesible usando `this`).