Trabajo práctico n.º 2

Programación II

Segundo cuatrimestre 2016

La fecha de entrega de este TP es el **17 de noviembre de 2016**. Se puede realizar de manera individual, o en grupos de a lo sumo dos personas.

El TP consta de dos partes, que se entregarán de manera conjunta en un único ZIP con el código Java según las instrucciones especificadas en al final de la consigna.

El código de apoyo para los ejercicios, más la versión más reciente de esta consigna, se pueden descargar aquí:

`https://github.com/ungs-prog2/codigo_aula/archive/tp2_2016_2.zip`

Contenidos

1. Generics e interfaces

1.1. Consigna

La interfaz *Deque<E>* de Java representa una cola doblemente terminada. Se pide **implementar la interfaz Deque** usando nodos doblemente enlazados.

Lineamientos de la entrega:

- La única estructura auxiliar que se permite es *Nodo*.
- El método `removeLast()` debe tener complejidad computacional $\mathcal{O}(1)$.
- El nombre de la clase debe ser *DequeEnlazada* y debe implementar correctamente la interfaz *Deque*[1] de Java.
- Se excluyen de la implementación los siguientes métodos, que simplemente devolverán *null* o *false* según corresponda:
  - `iterator()`
  - `descendingIterator()`
  - `toArray()`
  - `removeAll()`
  - `retainAll()`

Las secciones que siguen explican esta consigna en detalle.

1.2. Cola doblemente terminada

Una cola doblemente terminada[2] es una estructura de datos normalmente enlazada con primitivas de inserción y borrado para cada extremo (cabeza y cola).

Las primitivas resultantes son:

	Inserción	Examen	Borrado
Cabeza	`insertarPrimero`	`verPrimero`	`eliminarPrimero`
Cola	`insertarUltimo`	`verUltimo`	`eliminarUltimo`

De estas seis primitivas, las primeras cinco son $\mathcal{O}(1)$ de manera trivial al combinar una estructura enlaza simple con una referencia al último elemento. Por el contrario, para implementar `eliminarUltimo()` en tiempo constante es imprescindible usar nodos doblemente enlazados.

[1] `https://docs.oracle.com/javase/7/docs/api/java/util/Deque.html`
[2] `https://es.wikipedia.org/wiki/Cola_doblemente_terminada`

1.3. La interfaz *Deque* de Java

La interfaz *Deque* de Java incluye muchas más primitivas de las que arriba listadas. Los motivos se explican aquí:[3]

1. Las primitivas de examen y borrado ofrecen dos variantes según su comportamiento cuando la cola está vacía:
   - `removeFirst` y `getFirst` lanzan *NoSuchElementException*.
   - `pollFirst` y `peekFirst` simplemente devuelven *null*.
2. Debido a interfaces antiguas como *Stack* y *Queue*, *Deque* ofrece más de un nombre para algunos métodos:

Alias	Canónico
`push()`	`addFirst()`
`add()`	`offerLast()`
`offer()`	`offerLast()`
`pop()`	`removeFirst()`
`remove()`	`removeFirst()`
`poll()`	`pollFirst()`
`peek()`	`peekFirst()`
`element()`	`getFirst()`

3. *Deque* hereda de *Collection*, por lo que incluye (entre otros):
   - `clear()`
   - `addAll()`
   - `contains()`
   - `containsAll()`
4. *Deque* viola la interfaz estándar de la cola doblemente terminada ofreciendo borrado en nodos intermedios:
   - `removeFirstOccurrence()`
   - `removeLastOccurrence()`

1.4. Materiales y ayuda para la implementación[4]

En el ZIP adjunto se incluye una clase abstracta *DequeBase* que especifica las primitivas a implementar. El resto de primitivas de la interfaz están ya implementadas en la clase abstracta.

De usarse esta clase base, la declaración final de *DequeEnlazada* quedaría:

[3] Este listado es meramente informativo. En la sección 1.4 se incluye código auxiliar para hacer manejable la extensión de la interfaz.

[4] Esta sección resuelve para el alumno el "trabajo burocrático" que conlleva implementar la interfaz al completo. Su uso, aunque recomendado, es enteramente opcional; verdaderamente, los lineamientos de la entrega al principio de la consigna son suficientes para implementar el ejercicio.

```
public class DequeEnlazada<T>
               extends DequeBase<T>
               implements Deque<T> { ... }
```

Otras sugerencias:

- se puede arrancar a partir de las clases *PilaInt* y *ColaInt* vistas en la cursada y/o en materias anteriores. También se incluye su código en el adjunto.
- se incluye asimismo una clase de test con la que comprobar el correcto funcionamiento de la implementación a entregar.

2. Herencia y polimorfismo

2.1. Introducción

2.2. Consigna

2.3. Recomendaciones/guía

2.4. Informe

Instrucciones para la entrega