Nombre y Apellido: ..... N° Legajo: .....

Segundo Parcial de Programación Orientada a Objetos (72.33)

07/06/2018

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*	*Firma Docente*

- **Condición mínima de aprobación: Tener BIEN o BIEN- dos de los tres ejercicios.**
- **Las soluciones que no se ajusten al paradigma OO, no serán aceptadas.**
- **Las soluciones que no se ajusten estrictamente al enunciado, no serán aceptadas.**
- **Puede entregarse en lápiz.**
- **No es necesario escribir las sentencias `import`.**
- **Además de las clases solicitadas se pueden agregar las que consideren necesarias.**
- **Escribir en cada hoja Nombre, Apellido, Legajo, Número de Hoja y Total Hojas entregadas.**

Ejercicio 1

Un **iterador cíclico** es un iterador que, a partir de una colección, permite recorrerla infinitamente. Al igual que un iterador convencional permite recorrer todos los elementos de la colección, pero la diferencia es que, una vez consumido todos los elementos de la colección, vuelve a recorrer los mismos elementos.

Se desea implementar un iterador cíclico que recibe un entero **N** en su constructor y, ante cada invocación a **`next`**, retorne **N elementos de la colección**.

El método `next` lanza una `NoSuchElementException` cuando una invocación a `hasNext` hubiera retornado `false`.

Implementar todo lo necesario para que, con el siguiente programa:

```
import java.util.Arrays;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class CyclicIteratorTester {

   public static void main(String[] args) {
       List<String> list = Arrays.asList("hola", "que", "tal", "todo", "bien");
       CyclicIterator<String> listIterator = new CyclicIterator<>(3, list);
       for(int i = 0; listIterator.hasNext() && i < 4; i++) {
           for(String element : listIterator.next()) {
               System.out.print(" :" + element);
           }
           System.out.println();
       }
       System.out.println("----------");
       Set<Integer> set = new HashSet<>();
       CyclicIterator<Integer> setIterator = new CyclicIterator<>(3, set);
       System.out.println(setIterator.hasNext());
       setIterator.next();
   }

}
```

se obtenga la siguiente salida

```
 :hola :que :tal
 :todo :bien :hola
 :que :tal :todo
 :bien :hola :que
----------
false
Exception in thread "main" java.util.NoSuchElementException
```

Ejercicio 2

Una **cola de prioridad** es una colección ordenada. El orden de los elementos depende del orden de inserción, pero además de un número que establece la prioridad de ese elemento sobre el resto.

Dados dos elementos de la colección, si tienen la misma prioridad, el que se insertó primero está antes que el segundo. Si la prioridad de ambos difiere quien tenga menor prioridad está primero. Al sacar un elemento siempre se saca el primero.

Se cuenta con la interfaz `PriorityQueue<E>`

```
public interface PriorityQueue<E> {

   /** Agrega el elemento */
   void enqueue(E element, int priority);

   /** Remueve y retorna el elemento con mayor prioridad */
   E dequeue();

   /** Indica si la cola de prioridad está vacía */
   boolean isEmpty();

   /** Devuelve la cantidad de elementos encolados */
   int size();

   /** Devuelve la cantidad de elementos encolados con la prioridad indicada */
   int size(int priority);

}
```

Implementar todo lo necesario para que, con el siguiente programa:

```
public class PriorityQueueTester {

   public static void main(String[] args) {
       PriorityQueue<String> pq = new PriorityQueueImpl<>();
       pq.enqueue("Hola", 2);
       pq.enqueue("Chau", 5);
       pq.enqueue("Mundo", 2);
       pq.enqueue("ZZZZ", 1);
       pq.enqueue("Java", 1);
       pq.enqueue("Java", 5);
       pq.enqueue("Algo", 1);
       System.out.println(pq.dequeue());
       pq.enqueue("Otro", 1);
       System.out.println(pq.size());
       System.out.println(pq.size(2));
       while(!pq.isEmpty()) {
           System.out.println(pq.dequeue());
       }
   }

}
```

se obtenga la siguiente salida

```
ZZZZ
7
2
Java
Algo
Otro
Hola
Mundo
Chau
Java
```

Ejercicio 3

Se quiere modelar el menú de una pizzería.

Se tienen dos tipos de pizza: al horno y a la parrilla y varios agregados (*toppings*) que se le pueden poner encima a la pizza: extra queso, tomate y cebolla.

Los precios de los tipos de pizza son:

- **Pizza al horno:** $100
- **Pizza a la parrilla:** $150

Los precios de los *toppings*:

- **Extra queso:** $20
- **Tomate:** $30
- **Cebolla:** $10

Completar los … e implementar todo lo necesario para que, con el siguiente programa de prueba

```
public class PizzaTester {

  public static void main(String[] args) {
     // Pizza a la parrilla con Tomate
     Pizza pizza1 = ...................................................;
     // Pizza a la parrilla con Tomate con Cebolla con Extra queso
     Pizza pizza2 = ...................................................;
     // Pizza al horno con Cebolla con Extra queso
     Pizza pizza3 = ...................................................;
     System.out.println(pizza1);
     System.out.println(pizza2);
     System.out.println(pizza3);
  }

}
```

se obtenga la siguiente salida:

```
Pizza a la parrilla con Tomate: $180.00
Pizza a la parrilla con Tomate con Cebolla con Extra queso: $210.00
Pizza al horno con Cebolla con Extra queso: $130.00
```