PRÁCTICA A002.- COLAS DE PRIORIDAD LIMITADAS

Una cola de prioridad es una colección en la que los elementos se organizan teniendo en cuenta su prioridad. Debe permitir al menos las siguientes operaciones:

- **insertar un elemento** : lo insertará como último elemento de todos los que tengan la misma prioridad que el que se va a insertar (como en una cola) o justo antes del primer elemento que tenga prioridad menor.
- **eliminar un elemento**: se ha de eliminar siempre el primero de entre los que tengan mayor prioridad.

Si la **cola de prioridad está limitada** quiere decir que restringe el número de elementos en la cola de prioridad a su capacidad. Cuando se inserta un elemento en una cola de prioridad llena (su número de elementos coincide con su capacidad) se insertar el nuevo elemento y posteriormente se elimina el último de la menor prioridad (de los de menor prioridad el que menor tiempo lleve en la cola).

Las colas de prioridad se pueden implementar de muy diversas formas. Por ejemplo:

- Mediante un array en el que sus elementos sean colas. Los elementos se colocarán en una u otra cola directamente según su prioridad. Como java no permite tener arrays de colas genéricas se implementará mediante un arrayList de colas.

- Mediante una lista enlazada ordenada. Se ordena la lista por la prioridad, quedando como primer elemento de la lista el que tenga mayor prioridad. Por tanto para eliminar el de mayor prioriodad bastará con tomar el primer elemento. Sin embargo, para insertar un elemento habrá que buscar su posición en la lista teniendo en cuenta su prioridad. En el caso de que ya haya en la lista elementos con prioridad igual a la del elemento que se quiere insertar, se colocará detrás del último elemento de la lista que tenga igual prioridad.

INSTRUCCIONES PARA LA RESOLUCIÓN:

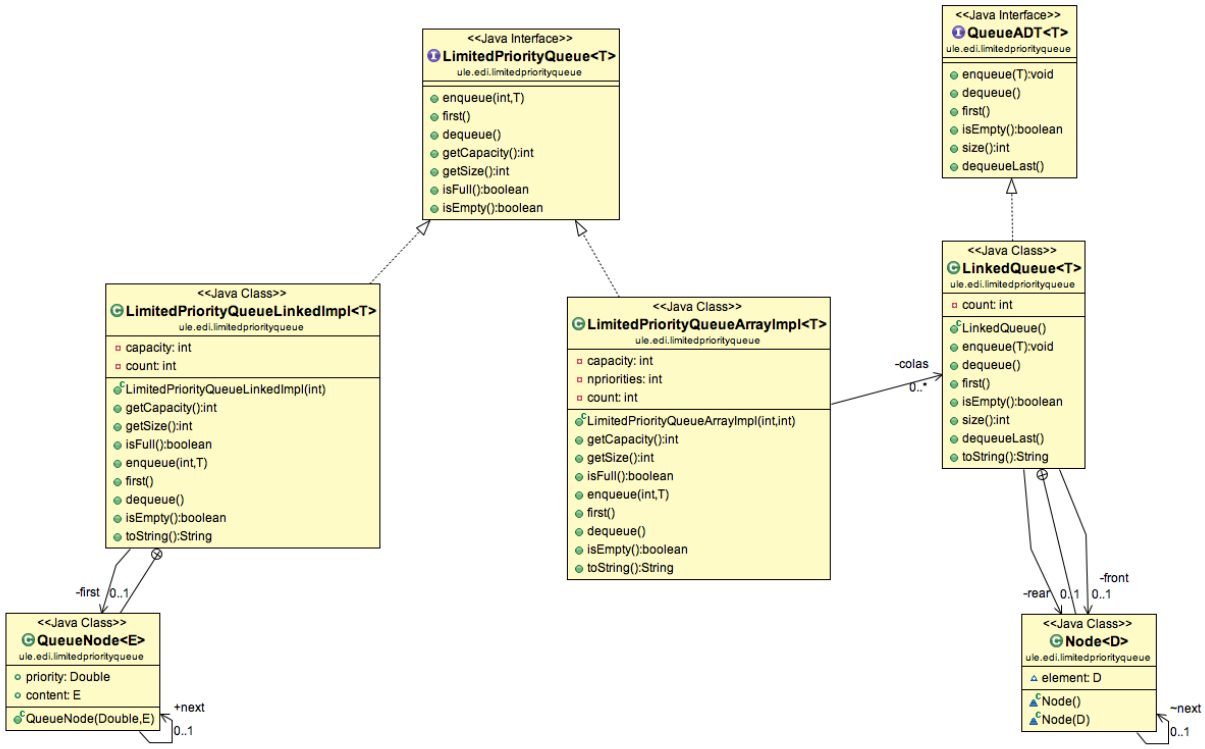
1. Descargar el proyecto edi_a002_2019 de la página de la asignatura en agora.unileon.es.
2. Importar dicho proyecto en Eclipse : Import… General…..Existing proyects into Workspace… Select archive file… (indicar el archivo ZIP descargado)
3. El proyecto edi-a002-2019 contiene el fichero **LimitedPriorityQueue<T>,** donde define el interfaz para las colas de prioridad. Este fichero NO SE PUEDE MODIFICAR.

```
package ule.edi.limitedpriorityqueue;

/**
 * Interface LimitedPriorityQueue<T> define una cola de prioridad limitada.
 *
 * Elementos de tipo T se encolan con una prioridad dada.
 *
 * Al desencolar se saca el elemento de mayor prioridad que más tiempo lleve en
 * la cola.
 *
 * El orden de los elementos con la misma prioridad es FIFO (se comporta como
 * una cola para los de la misma prioridad)
 *
 * No se aceptan objetos Null para encolar en la cola de prioridad. Los métodos
 * deben lanzar la excepción NullPointerException si se el elemento a insertar
 * es Null
```

```
 *
 */
public interface LimitedPriorityQueue<T> {

	/**
	 * Inserta un elemento en la cola.
	 *
	 * Si el número de elementos de la cola iguala su capacidad, se inserta el
	 * elemento según su prioridad y se saca el elemento de menor prioridad que
	 * lleve menos tiempo en la estructura.
	 *
	 * @param p
	 *            prioridad del elemento
	 * @param elem
	 *            elemento a encolar según su prioridad
	 *
	 * @return si la cola de prioridad estaba llena, devuelve el elemento de menor prioridad
	 *                 que lleve menos tiempo en la cola;
	 *         si no estaba llena, devuelve null
	 * @throws IllegalArgumentException  si la prioridad no es correcta
	 *         NullPointerException si el elemento es null
	 *
	 */
	public T enqueue(int p, T elem);

	/**
	 * Obtener el elemento con la prioridad mayor que más tiempo lleve en la
	 * cola de prioridad
	 *
	 * El elemento no se elimina de la cola.
	 *
	 * @return el elemento con la prioridad más alta de la cola que más tiempo
	 *         lleve en la cola de prioridad.
	 * @throws EmptyCollectionException  si la cola está vacia.
	 */
	public T first() throws EmptyCollectionException;

	/**
	 * Elimina y devuelve el elemento con la prioridad mayor que más tiempo
	 * lleve en la cola de prioridad devuelve el elemento con mayor prioridad
	 * que más tiempo lleve en la cola.
	 *
	 *
	 * @return el elemento con mayor prioridad que más tiempo lleve en la cola.
	 * @throws EmptyCollectionException  si la cola está vacia.
	 */
	public T dequeue() throws EmptyCollectionException;

	/**
	 * La capacidad total de la cola de prioridad limitada.
	 *
	 * @return número max de elementos en la cola..
	 */
	public int getCapacity();

	/**
	 * Número de elementos actualmente en la cola.
	 *
	 * @return número de elementos en la cola.
	 */
	public int getSize();

	/**
	 * Indica si la cola está al máximo de capacidad (está llena)
	 *
	 * @return 'true' si la cola está al máximo de capacidad.
	 */
	public boolean isFull();

	/**
	 * Indica si la cola está vacía
	 *
	 * @return 'true' si la cola está vacia.
	 */
	public boolean isEmpty();

}
```

4. El proyecto edi-a002-2019, contiene dos ficheros java que corresponden con las dos implementaciones de cola de prioridad descritas anteriormente:

ESTRUCTURAS DE DATOS UTILIZADAS:



a. **LimitedPriorityQueueArrayImpl.java**: donde se implementará la cola de prioridad como una lista (**ArrayList** de colas normales implementadas con referencias).

```
public class LimitedPriorityQueueArrayImpl<T> implements LimitedPriorityQueue<T> {

    private int capacity;
    private int npriorities;
    private int count;

    private ArrayList<LinkedQueue<T>> colas;

public LimitedPriorityQueueArrayImpl(int capacity, int npriorities) {

  //TODO  asignar los valores de los atributos
  // Crear el arrayList, y añadir una cola por cada una de las prioridades (1..npriorities)
  // Si capacidad <=0 disparar la excepción: IllegalArgumentException
```

LinkedQueue<T> es una cola implementada con estructuras enlazadas que implementa el interface QueueADT<T> que tiene las operaciones de cola y además la operación dequeueLast() que desencola el elemento de menor prioridad que menos tiempo lleva en la estructura.

```
public interface QueueADT<T> {
        /**
     * Inserta un elemento en la cola.
     *
     * @param elem  elemento a encolar
     *
     * @throws NullPointerException si el elemento es null
     */
        public void enqueue (T element);
```

```
        /**
         * Elimina el elemento del principio de la cola y devuelve una referencia a él.
         *
         * @return el elemento al principio de la cola
         * @throws EmptyCollectionException
         */
        public T dequeue( ) throws EmptyCollectionException;

        /**
            * Obtener sin eliminarlo el primer elemento de la cola
            *
            * @return el elemento del principio de la cola
            * @throws EmptyCollectionException   si la cola está vacia.
            */
        public T first() throws EmptyCollectionException;

        /**
            * Indica si la cola está vacía
            *
            * @return 'true' si la cola está vacia.
            */
        public boolean isEmpty();

        /**
            * Número de elementos actualmente en la cola.
            *
            * @return número de elementos en la cola.
            */
        public int size();

        /**
            * El elimina el elemento del final de la cola (el que menos tiempo lleva)
            * y devuelve una referencia a él
            *
            * @return el elemento del final de la cola
            * @throws EmptyCollectionException   si la cola está vacia.
            */
        public T dequeueLast() throws EmptyCollectionException;

}
```

b. **LimitedPriorityQueueLinkedImpl.java**: donde se implementará la cola de prioridad como una lista ordenada simplemente enlazada.

NOTA IMPORTANTE: NO SE PUEDEN UTILIZAR LAS ESTRUCTURAS DE DATOS DEL API COLLECTIONS DE JAVA (HAY QUE DEFINIR ESTRUCTURAS DE DATOS PROPIAS EN DICHAS CLASES)

```
public class LimitedPriorityQueueLinkedImpl<T> implements LimitedPriorityQueue<T> {
            private int capacity;

            private QueueNode<T> first;
            private int count;


        private static class QueueNode<E> {

                public QueueNode(Double priority, E content) {
                        this.priority = priority;
                        this.content = content;
                        this.next = null;
                }

                public Double priority;

                public E content;

                public QueueNode<E> next;
        };
```

5. A la vez que se van desarrollando las clases anteriores se deben crear las correspondientes clases de pruebas JUnit 4 (cuyo nombre debe acabar en Test) para ir comprobando su correcto funcionamiento.
6. **Se deberá entregar en agora.unileon.es la versión final de la práctica.** Para ello habrá que exportar el proyecto edi-a002-2019 como zip (Export… General… Archive File)

FECHA LIMITE DE ENTREGA DE ESTA PRÁCTICA: 24 de Marzo de 2019 23:55

- **IMPORTANTE**: Utilizar JUNIT 4.
- Se valorará la cobertura total de los test implementados, por lo que debe usarse un plugin de Eclipse para comprobar la cobertura de los test generados. Se buscará el 100% en la cobertura de los ficheros **LimitedPriorityQueueArrayImpl.java y LimitedPriorityQueueLinkedImpl.java**