PRÁCTICA A002.- QueueWithRep (Colas con elementos repetidos)

La colección utilizada en esta práctica es una cola en la que los elementos pueden estar repetidos. Cada elemento de la misma mantendrá un contador para saber cuántas instancias de dicho elemento hay en la cola.

Cola: Lista en la que la única operación de inserción es al final de la misma y la única operación de extracción de elementos es al principio de la misma. Es una estructura FIFO (First In, First Out: el primero en entrar es el primero en salir**)**

PASOS PARA LA REALIZACIÓN DE LA PRÁCTICA:

1. Descargar el proyecto de la página de la asignatura en agora.unileon.es :

   ed-a002-2020.zip

2. Importar dicho proyecto en Eclipse : Import… Existing proyects into Workspace… Select archive file…

   (indicar el archivo ZIP descargado)

Los proyectos normalmente contendrán la estructura a respetar (y probablemente muchos errores de compilación ya que todo el trabajo está por hacer).

3. El proyecto ed-a002-2020 tiene un paquete ule.ed.queuewithrep que contiene el interface QueueWithRep donde se describen las operaciones que implementará cualquier cola con repeticiones.

```
/**
 * TAD 'QueueWithRep'
 *
 * Almacena un conjunto de objetos de tipo <code>T</code>, pero cuando varios objetos iguales (según su
 * método {@link #equals(Object)}) se añaden a la colección, se mantiene una única referencia al objeto
 * y un contador de duplicados.
 *
 * Ejemplos de aplicación de un TAD como éste podría ser llevar el
 * inventario de un personaje en un juego:
 *
 *       ("Cloth"(x10), "Gold"(x5), "Silver Key"(x2))
 *
 * o almacenar información sobre una caja registradora:
 *
 *       ("5€"(2), "0.5€"(20), "20€"(1))
 *
 *
 * Se deben considerar hasta 2^31 - 1 instancias de un mismo elemento,
 * por lo que se usará un <code>int</code> para almacenar ese dato.
 *
 * El tamaño total de la cola {@link QueueWithRep#size()} será un resultado
 * de tipo <code>long</code>.
 *
 * En cualquier caso, para simplificar la práctica se supondrá que no es necesario hacer comprobaciones de
 * rangos. Si fuera necesario, se dispone de {@link Integer#MAX_VALUE} y {@link Long#MAX_VALUE}.
 *
 *
 * Excepciones
 *
 * No se permiten elementos <code>null</code>. Si a cualquier método que recibe un elemento se le pasa el
 * valor <code>null</code>, lanzará una excepción {@link NullPointerException}.
```

```
 *
 * Los valores de parámetros <code>times</code> deben ser mayores o iguales a cero, nunca negativos. Si se
 * recibe un valor negativo se lanzará {@link IllegalArgumentException}.
 *
 * Ambas son ejemplos de "unchecked exceptions"
 * {@link http://docs.oracle.com/javase/7/docs/api/java/lang/RuntimeException.html}
 *
 *
 * Constructores
 *
 * Se definirá un constructor por defecto que inicialice la instancia como cola vacía
 *
 * Métodos {@link Object#equals(Object)} y {@link Object#hashCode()}
 *
 * No se definirán estos métodos. Ver la clase QueueWithRepS en el proyecto para la igualdad.
 *
 * Método {@link Object#toString()}
 *
 * Debe reflejar qué elementos y cuántos de cada uno hay en la cola concatenando cada instancia de cada
 *    elemento. Por ejemplo: (A, A, A, B )

 * Iterador
 *
 * El iterador se desplaza por todos los elementos de la cola, uno a uno.
 *
 * Por ejemplo, con una cola.  ("A"(2), "B"(5))
 *
 * el método {@link Iterator#next()} devolverá uno tras otro
 *
 *        "A", "A", "B", "B", "B", "B", "B"
 *
 * @param <T> tipo de elementos en la cola
 */
public interface QueueWithRep<T> extends Iterable<T> {

          /**
           * Añade varias instancias de un elemento a esta cola
           *
           * Si una cola contiene ("XYZ"(1), "123"(5)), y se añaden seis instancias de "ABC", se pasará a
           * tener una cola con ("ABC"(6), "XYZ"(1), "123"(5)). Si ahora se añaden dos instancias de
           * "123", se tendrá ("ABC"(6), "XYZ"(1), "123"(7)).
           *
           * @param element el elemento a añadir
           * @param times el número de instancias
           *
           * @throws NullPointerException el elemento indicado es <code>null</code>
           * @throws IllegalArgumentException si <code>times</code> fuera negativo
           */
          public void add(T element, int times);

          /**
           * Añade una instancia de un elemento a esta cola
           *
           * @param element el elemento a añadir
           *
           * @throws NullPointerException el elemento indicado es <code>null</code>
           */
          public void add(T element);


          /**
           * Saca varias instancias de un elemento de esta cola
           *
           * Por ejemplo, al sacar dos instancias de "ABC" de una
           * COLA ("ABC"(10), "123"(2)), ésta pasará a ser
           * ("ABC"(8), "123"(2)).
           *
           * Se admite solamente un número de instancias menor del que hay en la cola;
           * e.g. si inicialmente se tiene (“ABC"(2)) y se sacan 8 instancias de "ABC",
           * se queda como estaba, y lanza la excepción IllegalArgumentException.

           * @param element elemento a sacar de esta cola
           * @param times número de instancias a sacar
           *
           * @throws NullPointerException el elemento indicado es <code>null</code>
           * @throws NoSuchElementException el elemento indicado  <code>no está en la cola</code>
           * @throws IllegalArgumentException si <code>times</code> fuera negativo, cero o mayor o igual que
```

```
                 *.         el número de apariciones del elemento
                 */
                public void remove(T element, int times);

                /**
                 * Saca el primer elemento completo de esta cola
                 *
                 *
                 * @throws EmptyCollectionException si la cola está vacía
                 * @return el numero de apariciones que había en la cola del primer elemento (que elimina)
                 */
                public int remove() throws EmptyCollectionException;


                /**
                 * Elimina todo el contenido de esta cola
                 */
                public void clear();


                /**
                 * Indica si el elemento está en esta cola
                 *
                 * Devuelve <code>true</code> si al menos existe una
                 * instancia del elemento dado en esta cola (es decir,
                 * un elemento 'x' tal que <code>x.equals(element)</code>)
                 * y <code>false</code> en caso contrario.
                 *
                 * @param element elemento a buscar en esta cola
                 * @return <code>true</code>/<code>false</code> según el resultado
                 *
                 * @throws NullPointerException el elemento indicado es <code>null</code>
                 */
                public boolean contains(T element);

                /**
                 * Indica si esta cola está vacía
                 *
                 * @return <code>true</code> si no contiene elementos
                 */
                public boolean isEmpty();

                /**
                 * Devuelve el número total de instancias en esta cola
                 *
                 * Por ejemplo, para una cola ("5€"(2), "10€"(8)) se
                 * devolverá un tamaño de 2 + 8 = 10.
                 *
                 * @return número total de instancias en esta cola
                 */
                public long size();

                /**
                 * Devuelve el número de instancias del elemento dado
                 *
                 * Es decir, el número de instancias del objeto 'x' tal que
                 * <code>x.equals(element)</code>. Por ejemplo, con una cola
                 * B1 = (AX(2), BX(8)) se indicará 8 si se pregunta por
                 * <code>B1.count(BX)</code>.
                 *
                 * Si el elemento no está, se devuelve cero.
                 *
                 * @param element el elemento a buscar en esta cola
                 * @return el número de instancias que se encuentran
                 *
                 * @throws NullPointerException el elemento indicado es <code>null</code>
                 */
                public int count(T element);

        }
```

4. El proyecto ed-a002-2020 contiene 3 paquetes:

a. ule.edi.queuewithrep **es el que hay que completar**: que contiene dos ficheros java que se corresponden con dos implementaciones diferentes del interface QueueWithRep:

   i. **ArrayQueueWithRepImpl.java**: donde se implementará el interface QueueWithRep con un array donde los elementos del mismo tienen el elemento y el número de instancias (num) que hay de ese elemento.

   ii. **LinkedQueueWithRepImpl.java**: donde se implementará el interface QueueWithRep con una lista enlazada.

   iii. A la vez que se van desarrollando las clases anteriores se deben crear los correspondientes métodos de pruebas JUnit 4 para ir comprobando su correcto funcionamiento. Para reaprovechar los tests de una implementación en la otra (ya que el comportamiento de la colección es idéntico en las dos implementaciones) solamente se escribirán los tests una vez en el fichero **AbstractQueueWithRefTests.java. (ES EN ESTE FICHERO DONDE HAY QUE ESCRIBIR LOS TESTS).**

      1. Los otros ficheros de tests simplemente inicializan los tests de la clase anterior para cada una de las implementaciones de la cola (y podrían incluir tests específicos de dicha implementación, como por ejemplo en la implementación del array, para probar que funciona el método expandCapacity si se acaba el espacio libre en el array)

5. S**e deberá entregar en agora.unileon.es la versión final de la práctica** (proyecto exportado como zip).

   NOTA: Los alumnos que descargaron el proyecto la semana pasada **deben revisar las estructuras de datos** usadas en cada una de las implementaciones y la signatura y funcionalidad de los métodos remove(T elem, int count) y remove(). **VER ARCHIVO APARTE**

FECHA LIMITE de entrega de la práctica A002-2020:

20 - Marzo -2020 a las 23:59