```java
import java.util.Scanner;       // trae las funciones de leer datos del teclado
import java.util.ArrayList;     // Para usar ArrayList


public class ArregloGarantizado<E> extends ArrayList<E> { // Declaración de la clase,
    parecida a la del PDF del
                                                          // pero extiende la clasa y
                                                             no implementa una interfaz
  final static int PORCENTAJE=0;                          // Por si se desea crecer por
      porcentaje
  final static int ESPACIO=20;                            // Crecimiento según
      requerimientos
  final static int INICIAL=0;                             // Arbitrariamente escogemos
      iniciar en 8 (no en 1)
  final static int FACTOR=2;                              // Crecimiento original en
      los ejemplos de los PDF
  final static boolean DEBUG=true;                        // Para depurar

  // *************************
  // * Atributos de la clase *
  // *************************

  /**
   * Arreglo que contiene los elementos de la lista (inicia con capacidad INICIAL)
   * Código tomado directamente de "Escenario 3 - Las listas como estructuras de
datos" 6.2
   */
  private Object[] arreglo=new Object[INICIAL];

  /**
   * Tamaño de la lista (inicia en 0).
   */
  private int tam=INICIAL;

  // ************************
  // * Métodos de la clase *
  // ************************

  /**
   * Este constructor no hace nada según "Escenario 3 - Las listas como estructuras
de datos" 6.2.1
   */
  public ArregloGarantizado() {
    // Nada
  } // ArregloGarantizado() constructor

  /**
```

```
   * Este destructor marca el arreglo como basura para que el GC libere
(eventualmente) la memoria
   * Código tomado directamente de "Escenario 3 - Las listas como estructuras de
datos" 6.2.2
   */
  public void clear() {
    // Liberar los apuntadores de todas las casillas ocupadas del arreglo
    for(int i=0; i<=tam; i++) {
      arreglo[i]=null;
    }
    // Poner el tam en cero para que la lista quede vacía:
    tam=0;
  } // .clear()

  /**
   * Método para consultar un elemento dado el índice, un getter
   * Código tomado directamente de "Escenario 3 - Las listas como estructuras de
datos" 6.2.3
   */
  public E get(int index) {
    // Obtener el elemento que está en la posición index del arreglo:
    Object elemento=arreglo[index];
    // Retornar el elemento obtenido, convertido al tipo E por medio de un cast:
    return (E)elemento;
  } // .get(int)

  /**
   * Método para modificar el elemento en la posición del índice dado
   * Código tomado directamente de "Escenario 3 - Las listas como estructuras de
datos" 6.2.4
   */
  public E set(int index, E element) {
    // Obtener el elemento que se encuentra en la posición index:
    E anteriorValor=get(index);
    arreglo[index]=element;
    // Retornar el elemento que anteriormente se encontraba en la posición dada:
    return anteriorValor;
  } // .set(int,element)

  /**
   * Método para garantizar que el arreglo tiene espacio para agregar un elemento
   * Código tomado de "Escenario 3 - Las listas como estructuras de datos" 6.2.5 y
modificado según la entrega
   */
  private void garantizarCapacidad(int nuevaCantidadDeElementos) {
    // Si la nueva cantidad de elementos es menor o igual que la capacidad del
    // arreglo:
    if(nuevaCantidadDeElementos<=arreglo.length) {
```

```
      // No hacer nada porque los elementos ya caben.
    }
    // si la nueva cantidad de elementos es mayor o igual que la capacidad del
    // arreglo, tocaría "crecer" la capacidad del arreglo.
    else {
      // Si porcentaje deseado > 0 y la nueva cantidad es menor que
         (actual+porcentaje), ampliamos
      // el arreglo en eso, porcentaje deseado, si no, si el espacio adicional > 0 y
         la nueva
      // cantidad es mayor que (actual+adicional), ampliamos el arreglo en eso, si
         no, si la
      // nueva cantidad es menor que el doble, duplicamos el arreglo (duplicamos
         mientras FACTOR==2)
      // Si la nueva cantidad de elementos es menoe que el doble de la capacidad
      // del arreglo, dejarla en el doble de la capacidad del arreglo:
      if(PORCENTAJE>0) {
        if(DEBUG)
          System.err.format("\t\t» Intento de redimensionado por porcentaje (usando
%d%%)%n",PORCENTAJE);
        if(nuevaCantidadDeElementos<=(arreglo.length*(100+PORCENTAJE))/100)
          nuevaCantidadDeElementos=(arreglo.length*(100+PORCENTAJE))/100; // División
              entera
        if(DEBUG)
          System.err.format("\t\t» Redimensionado a %d%n",nuevaCantidadDeElementos);
      } else if(ESPACIO>0) {
        if(DEBUG)
          System.err.format("\t\t» Intento de redimensionado por espacio adicional
(usando %d)%n",ESPACIO);
        if(nuevaCantidadDeElementos<=(arreglo.length+ESPACIO))
          nuevaCantidadDeElementos=arreglo.length+ESPACIO;
        if(DEBUG)
          System.err.format("\t\t» Redimensionado a %d%n",nuevaCantidadDeElementos);
      } else {
        if(DEBUG)
          System.err.format("\t\t» Intento de redimensionado por factor (usando
%d)%n",FACTOR);
        if(nuevaCantidadDeElementos<=(arreglo.length*FACTOR))
          nuevaCantidadDeElementos=arreglo.length*FACTOR;
        if(DEBUG)
          System.err.format("\t\t» Redimensionado a %d%n",nuevaCantidadDeElementos);
      }

      // Crear un nuevo arreglo donde quepa la nueva cantidad de elementos:
      Object[] nuevoArreglo=new Object[nuevaCantidadDeElementos];
      // Copiar todos los elementos del viejo arreglo al nuevo arreglo:
      for(int i=0; i<tam; i++) {
        nuevoArreglo[i]=(int)arreglo[i];
      }
```

```
      // Desechar el viejo arreglo, asignándole el nuevo:
      arreglo=nuevoArreglo;
    }
  } // .garantizarCapacidad(int)

  /**
   * Método para insertar un nuevo elemento en una posición específica de la lista,
dado el índice
   * Código tomado de "Escenario 3 - Las listas como estructuras de datos" 6.2.5 y
modificado según la entrega
   */
  public void add(int index, E element) {
    // Garantizar que quepan en total tam+1 elementos en el arreglo:
    garantizarCapacidad(tam+1);
    // Correr una posición hacia la derecha todos los elementos desde la
    // posición tam-1 hasta la posición index:
    for(int i=tam-1; i>=index; i--) {
      arreglo[i+1]=arreglo[i];
    }
    // Ubicar el nuevo elemento en la posición index:
    arreglo[index]=element;
    // Incrementar el tam en una unidad:
    tam++;
  } // .add(int,element)

  /**
   * Método para remover un elemento en una posición específica de la lista, dado el
índice
   * Código tomado de "Escenario 3 - Las listas como estructuras de datos" 6.2.5 y
modificado según la entrega
   */
  public E remove(int index) {
    // Obtener el elemento que se encuentra en la posición index:
    E elemento=get(index);
    // Correr una posición hacia la izquierda todos los elementos desde la posición
       index+1 hasta la posición tam-1:
    for(int i=index+1; i<tam; i++) {
      arreglo[i-1]=arreglo[i];
    }
    // Poner el valor null en la posición tam-1 del arreglo:
    arreglo[tam-1]=null;
    // Decrecer el tam en una unidad:
    tam--;
    // Retornar el elemento que antes se encontraba en la posición index:
    return elemento;
  } // .remove(int)

  public static void main(String[] args) {
```

```
    Scanner key=new Scanner(System.in);  // variable temporal para recibir de teclado
    int v;                               // variable temporal para recibir el dato a
        almacenar
    System.out.println("Cuántos elementos?");
    int Cuantos=key.nextInt();
    ArregloGarantizado Datos=new ArregloGarantizado();   // Crea el arreglo minúsculo
    // LlenarLista
    System.err.println("----- Toma de datos -----");
    for(int n=0; n<Cuantos;n++){
      System.out.format("[%d]? ",n);
      v=key.nextInt();
      Datos.add(n,(int)v);
    }
    System.err.println("----- Volcado de datos -----");
    // MostrarLista
    for(int n=0; n<Cuantos;n++) {
      System.out.format("[%d] = %d%n",n,Datos.get(n));
    }
  } // main

} // fin ArregloGarantizado
```