Ejercicio Tipos Parametrizados.

Objetivos del ejercicio

Comprender y practicar los siguientes conceptos:

- Interface
- **Tipos parametrizados**

Tarea

Tenemos la clase `ContenedorInteger` que almacena números enteros en un vector de forma ordenada ascendentemente.

```
import java.util.Vector;

public class ContenedorInteger {
     private Vector<Integer> datos;

     /** Construye un Contenedor de datos por defecto
      * de 10 elementos
      *
      */
     public ContenedorInteger() {
       datos= new Vector<Integer>();
     }

     /**
      * Construye un Contenedor de datos con parámetros de entrada:
      * Capacidad int
      * @return
      */
     public ContenedorInteger(int cap){
       datos= new Vector<Integer>(cap);
     }

     /**
      * Nos indica si el contenedor esta o no lleno
      * @return boolean
      */
     private boolean contenedorLleno(){
       if (datos.size()==datos.capacity())
              return true;
       else
              return false;
     }
     /**
      * Anyade un dato ordenado que le pasamos como parametro al contenedor
      * @param dato
      * @return booleano: Si se puede anyadir o no el dato al contenedor
      */
     public boolean anyadeDatoOrdenado(int dato){
       boolean ok=false;


       int i=0;
       while (i<datos.size() && (dato>datos.elementAt(i)) ){
              i++;
       }
       datos.add(i,dato);
       ok= true;
```

```
        return ok;
    }
    /**
     * Metodo para obtener el dato que está en una determinada posición
     * @param pos posición del elemento que queremos obtener
     * @return el elemento que está en esa posición
     */
    public int getDatoPos(int pos){
        int res;

                res= datos.elementAt(pos);

        return res;
    }
    /**
     * Metodo para obtener el dato que está en una determinada posición
     * @param pos posición del elemento que queremos obtener
     * @return el elemento que está en esa posición
     */
    public int obtenerDatoPos(int pos){
        int res;

        res= datos.elementAt(pos);
        return res;
    }
    /**
     * Método para eliminar un dato del contenedor
     * @param dato que queremos eliminar del contenedor
     */
    public void eliminarDato(int dato){
        datos.removeElement(dato);
    }

    /**
     * Método que devuelve el número de datos actuales
     * existentes en el contenedor
     * @return número de datos del contenedor
     */
    public int numElementos(){
        return datos.size();
    }
    /* (non-Javadoc)
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {
        return "Contenedor [datos=" + datos + "]";
    }


    /**
     * Método para buscar un dato en el contenedor
     * @param dato a buscar
     * @return booleano indicando si el dato se encuentra o no en el contenedor.
     */
    public boolean buscarDato(int dato){
        boolean esta= false;

        int i=0;
        while ((i<datos.size())&&(dato!=datos.elementAt(i)))
                i++;

        //Elemento esta en el vector
        if (i<datos.size())
                esta= true;
```

```
        return esta;
      }

      /**
       * Método para buscar un dato en el contenedor
       * @param dato a buscar
       * @return entero indicando si la posicion del dato en el contenedor, -1 si no
esta
       */
      public int buscarDatoPos(int dato){
        int pos=-1;

        int i=0;
        while ((i<datos.size())&&(dato!=datos.elementAt(i)))
              i++;

        //Elemento esta en el vector
        if (i<datos.size())
              pos=i;

        return pos;
      }
}
```

Cuestión 1.

Realiza los cambios necesarios sobre el código para que esta clase esté parametrizada y funcione correctamente con cualquier tipo de dato. Llama esta nueva clase **`ContenedorOrdenado`** y define una interfaz si es necesario.

NOTA: Crear el fichero fuente **`ContenedorInteger.java`** con el código previo. El fichero fuente **`Prueba.java`** está disponibles en el aula virtual.