Escuela Superior Politécnica del Litoral
Facultad de Ingeniería en Electricidad y Computación

Estructuras de Datos

Taller ArrayList

Durante esta actividad, el acceso a Internet está restringido **exclusivamente** al Aula Virtual.

Introducción

En este taller, usted empezará a trabajar en una implementación funcional de la estructura `ArrayList`, que es una implementación estática (o continua) del TDA `List`.

La clase ArrayList utiliza un arreglo cuyo tamaño "*crece*" a medida que se necesita más espacio. Por esta razón, además de un atributo para controlar su **capacidad inicial**, la clase `ArrayList` debe tener un atributo para llevar control del **tamaño efectivo** del arreglo que utiliza. Este atributo indica cuánto espacio del arreglo reservado está siendo realmente utilizado (es decir, cuántos elementos se han insertado en el `ArrayList`). Dicho atributo debe ser actualizado cada vez que un objeto sea añadido a o removido de la lista.

El TDA Lista y la interface `List`

Para los propósitos de esta práctica, el TDA List está definido mediante la interfaz `List` de la siguiente manera:

```
public interface List<E>
```

Note que la interfaz `List` es parametrizada por tipo. El parámetro de tipo E hace referencia a que se trata de una lista de *elementos*. Esta parametrización nos permitirá instanciar listas que almacenen elementos de cualquier tipo (p. ej., listas de números enteros, listas de estudiantes, o incluso listas de listas).

La interfaz `List` contiene:

Métodos que permiten conocer el tamaño de la lista, verificar si ésta está vacía, y vaciarla:

```
public int size();
public boolean isEmpty();
public void clear();
```

Métodos para añadir elementos a (y removerlos de) la lista y para recuperarlos:

```
public boolean addFirst(E e); // inserta el elemento e al inicio
public boolean addLast(E e); // inserta el elemento e al final
public E removeFirst(); // remueve el elemento al inicio de la lista
public E removeLast(); // remueve el elemento al final de la lista
```

Métodos de inserción y eliminación de elementos en posiciones específicas:

```
public void add (int index, E element); // inserta element en la posición index
public E remove (int index); // remueve y retorna el elemento en la posición index
public E get (int index); // retorna el elemento ubicado en la posición index
public E set (int index, E element); // setea el element en la posición index
```

Finalmente, la lista tiene también un método llamado `toString`, que retorna una cadena de caracteres representando a todos los elementos almacenados en la lista:

```
public String toString();
```

Algunas observaciones sobre la lista estática

Como se especificó antes, la clase `ArrayList` corresponde a la implementación estática (o continua) del TDA lista. Por tanto, internamente, utilizará un arreglo.

Para cumplir con la especificación del TDA, esta clase también debe ser parametrizada por tipo. Sin embargo, como hemos visto antes, **Java no permite declarar arreglos con tipos parametrizados**. Por tanto, la siguiente línea de código **NO** es lícita:

```
E elements[] = new E[capacity]; ❌
```

La línea anterior genera un error en tiempo de compilación: la máquina virtual indica que no es posible crear arreglos de genéricos. Por esta razón, el arreglo de elementos de la clase `ArrayList` debe ser declarado e inicializado de la siguiente manera:

```
E elements[] = (E[]) new Object[capacity]; ✔
```

Note que el problema se soluciona usando la clase `Object` y un cast (😖). **Esta será la única ocasión que utilicemos casting en este curso**. Como lo hemos discutido antes, este tipo de operación **no es recomendable**. Este *cast*, sin embargo, está dentro de la clase `ArrayList` y quien use la clase no se enterará, necesariamente, de los detalles internos de implementación.

En el código mostrado arriba, el atributo `capacity` controla el tamaño inicial con el que "*nace*" todo `ArrayList`.

Note que, en un `ArrayList`, las operaciones de inserción y eliminación de elementos modifican su tamaño efectivo. Asimismo, la clase debe *incrementar* la capacidad del arreglo cada vez que se requiera insertar un nuevo elemento en un `ArrayList` que está lleno. La política de crecimiento que usaremos en este curso es que un `ArrayList` lleno **crecerá siempre al doble de su tamaño actual**. Este crecimiento se implementará en un método privado llamado `addCapacity()`.

Su misión

En esta práctica, usted implementará los siguientes métodos de la clase `ArrayList`:

> `public String` **`toString`**`()`: Este método proporcionar una representación en forma de cadena de los elementos almacenados en el `ArrayList`. Por ejemplo, el método `toString` retorna el `String` `"1,2,3,4"` al ser invocado por la lista `[1, 2, 3, 4]`.
>
> `public ArrayList<E>` **`subList`**`(int from, int to)`: Este método devuelve un nuevo `ArrayList` que contiene los elementos desde el índice `from` (incluido) hasta el índice `to` (incluido). Por ejemplo, si la lista que invoca el método es `[1, 2, 3, 4, 5, 6]`, `from` es 2, y `to` es 4, el método `subList` retorna la lista `[3, 4, 5]`.
>
> `public ArrayList<E>` **`removeFirstNElements`** `(int n)`: Este método remueve los primeros N elementos del ArrayList. Por ejemplo, si la lista que invoca el método es `[1, 2, 3, 4, 5, 6]`, y n es 3, la lista queda así: `[4, 5, 6]`.
>
> `public void` **`rotate`** `(int k)`: Este método rota los elementos del `ArrayList` k posiciones hacia la derecha. Los elementos que salen por la derecha vuelven a entrar por la izquierda. Por ejemplo, si la lista `[1, 2, 3, 4, 5]` invoca a `rotate` con `k = 2`, la lista cambia a `[4, 5, 1, 2, 3]`.
> Otro ejemplo, si la lista `["Alice", "Bob", "Charlie", "David", "Eve", "Frank"]` invoca al método `rotate` con `k = 3`, ésta queda así:
> `["David", "Eve", "Frank", "Alice", "Bob", "Charlie"]`.

Para probar el funcionamiento de su estructura, usted debe escribir un programa principal que instancie dos `ArrayLists`: uno de números enteros y otro de objetos de tipo `String`. Usted deberá añadir elementos a estas listas y probar el funcionamiento de sus métodos. Utilice el método **`toString`** para imprimir el contenido de cada lista luego de insertarle elementos y de invocar sus métodos.

Consideraciones Finales

Para lograr los métodos solicitados, es posible que deba también implementar algunos de los métodos que están vacíos en el código adjunto. Implemente esos métodos vacíos **únicamente si los utiliza** en sus soluciones de los métodos `toString`, `suffle`, o `rotate`.

En esta práctica, usted es dueño(a) de su estructura. Por tanto, deberá tomar algunas decisiones respecto al comportamiento de sus métodos. Cuando lo considere oportuno, incluya comentarios en su código para justificar sus decisiones de diseño.