C.F.G.S. DESARROLLO DE APLICACIONES MULTIPLATAFORMA

MODULO PROGRAMACIÓN

Tutorial Básico de la Clase ArrayList

Tutorial Básico de la Clase ArrayList

1. Introducción

La clase `ArrayList` forma parte del *Java Collections Framework*, representando una buena alternativa al uso de arrays cuando se desconoce a priori el número de elementos que se van a almacenar. Internamente, la clase `ArrayList` se implementa mediante arrays, creando un array inicial de una capacidad determinada. Si se llena, crea un nuevo array de mayor capacidad, copia los elementos del array original al nuevo array y desecha el primero.

En la práctica, un `ArrayList` es similar a un array unidimensional cuya capacidad crece de forma dinámica a medida que va siendo necesario. De hecho, cada posición ocupada también se identifica mediante un índice. El índice de la primera posición tiene el valor `0`, el de la segunda el valor `1` y así sucesivamente hasta el índice de la última posición que tendrá el valor `N-1`, siendo `N` el número de elementos almacenados.

Para obtener el número de elementos almacenados se define el método `size()`.

2. Creación

La clase `ArrayList` define tres constructores:

- `ArrayList()`: crea un `ArrayList` con una capacidad inicial para 10 elementos.
- `ArrayList(int initialCapacity)`: crea un `ArrayList` con la capacidad inicial especificada en el parámetro `initialCapacity`.
- `ArrayList(Collection c)`: crea un `ArrayList` añadiendo los elementos almacenados en la colección `c`.

La clase `ArrayList` se define como un tipo genérico que se parametriza con el tipo de los elementos que se van a almacenar. Ejemplos:

```
ArrayList<String> lista1 = new ArrayList<>();
ArrayList<Integer> lista2 = new ArrayList<>(20);
ArrayList<String> lista3 = new ArrayList<>(lista1);
```

3. Añadir elementos

Para añadir elementos a un `ArrayList` se definen los métodos `add` y `addAll` para añadir uno o varios elementos respectivamente. En ambos casos se trata de métodos sobrecargados. Ejemplos:

```
// Creación de un ArrayList
ArrayList<String> nombres1 = new ArrayList<>();
// Añadiendo elementos al final
nombres1.add("Fernando");
nombres1.add("Lara");
nombres1.add("Elena");
// Contenido de nombres1 después de añadir: [Fernando, Lara, Elena]
System.out.println(nombres1);

// Añadiendo un elemento en la posición 2
nombres1.add(2, "Carlos");
// Contenido de nombres1 después de añadir: [Fernando, Lara, Carlos, Elena]
System.out.println(nombres1);

// Añadiendo los elementos de un ArrayList al final de otro ArrayList
ArrayList<String> nombres2 = new ArrayList<>();
nombres2.add("Paula");
```

```
nombres2.add("Bruno");
nombres2.add("Carmen");
nombres1.addAll(nombres2);
// Contenido de nombres1 después de añadir: [Fernando, Lara, Carlos, Elena, Paula, Bruno, Carmen]
System.out.println(nombres1);

// Añadiendo los elementos de un ArrayList en la posición 2 de otro ArrayList
ArrayList<String> nombres3 = new ArrayList<>();
nombres3.add("Olivia");
nombres3.add("Valentina");
nombres3.add("Ismael");
nombres3.addAll(2, nombres2);
//Contenido de nombres3 después de añadir: [Olivia, Valentina, Paula, Bruno, Carmen, Ismael]
System.out.println(nombres3);
```

4. Sustituir elementos

Para sustituir el elemento almacenado en una posición específica se define el método `set`. Por ejemplo, para sustituir el elemento `"Carmen"` por el elemento `"Hugo"` en `nombres3` invocamos al método `set` de la forma siguiente:

```
nombres3.set(4, "Hugo");
//Contenido de nombres3 después de sustituir: [Olivia, Valentina, Paula, Bruno, Hugo, Ismael]
System.out.println(nombres3);
```

5. Eliminar elementos

Para eliminar elementos de un `ArrayList` se definen los métodos siguientes:

- `remove(int index)`: elimina el elemento almacenado en la posición `index`. Ejemplo:

```
// Contenido de nombres1 antes de eliminar: [Fernando, Lara, Carlos, Elena, Paula, Bruno, Carmen]
nombres1.remove(1);
// Contenido de nombres1 después de eliminar: [Fernando, Carlos, Elena, Paula, Bruno, Carmen]
System.out.println(nombres1);
```

- `remove(Object o)`: elimina la primera ocurrencia del elemento `o`. Ejemplo:

```
// Contenido de nombres1 antes de eliminar: [Fernando, Carlos, Elena, Paula, Bruno, Carmen]
nombres1.remove("Carmen");
// Contenido de nombres1 después de eliminar: [Fernando, Carlos, Elena, Paula, Bruno]
System.out.println(nombres3);
```

- `removeAll(Collection c)`: elimina todos los elementos que se encuentren almacenados en `c`. Ejemplo:

```
// Contenido de nombres1: [Fernando, Carlos, Elena, Paula, Bruno]
// Contenido de nombres3 antes de eliminar: [Olivia, Valentina, Paula, Bruno, Hugo, Ismael]
nombres3.removeAll(nombres1);
//Contenido de nombres3 después de eliminar: [Olivia, Valentina, Hugo, Ismael]
System.out.println(nombres3);
```

6. Obtener elementos e índices

Para obtener el elemento almacenado en una posición específica se define el método `get(int index)`. Ejemplo:

```
// Contenido de nombres1: [Fernando, Carlos, Elena, Paula, Bruno]
String nombre = nombres1.get(3);
// En nombre se almacena "Paula"
System.out.println(nombre);
```

Para obtener el índice correspondiente a la posición en la que se encuentra almacenado un elemento se define el método `indexOf(Object o)`. En caso de que el elemento se encuentre repetido, se retorna el indice correspondiente a la primer ocurrencia. Ejemplo:

```
// Contenido de nombres1: [Fernando, Carlos, Elena, Paula, Bruno]
int i = nombres1.indexOf("Carlos");
// En i se almacena el valor 1
System.out.println(i);
```

Para obtener el índice correspondiente a la posición en la que se encuentra almacenada la última ocurrencia de un elemento se define el método `lastIndexOf(Object o)`. Ejemplo

```
nombres1.add("Carlos");
nombres1.add("Valentina");
// Contenido de nombres1: [Fernando, Carlos, Elena, Paula, Bruno, Carlos, Valentina]
int li = nombres1.lastIndexOf("Carlos");
/ En li se almacena el valor 5
System.out.println(li);
```

7. Recorrido de una ArrayList

Existen tres formas de recorrer un `ArrayList` sin contar la utilización de bucles regulares:

1. Usando el bucle `for` mejorado. Ejemplo:

```
for (String nombre: nombres1)
    System.out.println(nombre);
```

2. Usando un objeto `Iterator` o un objeto `ListIterator`. En el segundo caso es posible recorrerlo en ambas direcciones. Ejemplo:

```
Iterator<String> i = nombres1.iterator();
While(i.hasNext())
    System.out.println(i.next());
```

3. Usando el método `forEach` con la correspondiente expresión lambda. Ejemplo:

```
Nombres1.forEach(nombre -> System.out.println(nombre));
```

Se ha de tener en cuenta que cualquier intento de modificar el contenido de un `ArrayList` mientras se está realizando el recorrido del mismo, provocará el lanzamiento de la `ConcurrentModificationException`.

Existe una excepción a esta regla que consiste en la posibilidad de eliminar el elemento retornado por método `next` de la clase `Iterator` mediante el método `remove` de la misma. Ejemplo:

```
// Contenido de nombres1 antes de iterar: [Fernando, Carlos, Elena, Paula, Bruno, Carlos, Valentina]
Iterator i = nombres1.iterator();
while(i.hasNext()) {
    String nombre = i.next();
    if (nombre.toLowerCase().charAt(0) == 'c')
        iterator.remove();
    else
        System.out.println(nombre);
}
// Contenido de nombres1 después de iterar: [Fernando, Elena, Paula, Bruno, Valentina]
```