Uso de Colecciones: ArrayList

Descripción Básica

Java provee un framework de colecciones muy rico conformado por un conjunto de interfaces y clases que las implementan. Una de estas implementaciones es el ArrayList, cuya funcionalidad básica se describe aquí.

Una instancia de la clase java.util.ArrayList constituye una colección lineal ordenada, de crecimiento dinámico, que puede contener elementos duplicados, y ofrece acceso posicional en tiempo constante. Esta lista está indexada comenzando desde cero; es decir que el primer elemento de la lista se encuentra en esa posición.

Cada vez que se agrega un elemento, la lista por default lo ubica en la última posición; no obstante esto, pueden agregarse elementos en otras posiciones, lo cual genera el desplazamiento de una posición en sentido creciente de todos los elementos que se encuentren ubicados a partir del lugar en que se agrega el nuevo elemento.

También pueden eliminarse elementos de cualquier posición de la lista. Cuando esto ocurre, la lista genera un desplazamiento en sentido contrario al anteriormente descripto con el fin de no dejar espacios vacíos (por supuesto que este desplazamiento no ocurre cuando se elimina el último elemento de la lista).

Definición y Creación

Java permite tipar una colección especificando el tipo de los elementos que contendrá. Este tipo podrá ser una clase o una interfaz, pero no un tipo primitivo (para esto último deberían utilizarse wrappers).

El tipo de los elementos de la colección se especifica entre "<" y ">", de la siguiente manera:

List<T> items;

en donde "T" indica el nombre de una clase o interfaz.

Por lo tanto la declaración anterior indica que "items" será una instancia de ArrayList que contendrá objetos de tipo "T".

Por ejemplo, un ArrayList que contendrá instancias de la clase Employee (empleado) se define y crea de la siguiente manera:

List<Employee> employees = new ArrayList<Employee>();

Este tipado permitirá al compilador chequear que no se agregue a la lista ningún elemento que no esté tipado con la clase Employee o alguna de sus subclases, lo cual resulta en un uso más seguro.

Protocolo de uso

Entre los mensajes más utilizados de ArrayList se encuentran los siguientes:

- add(<T> o)

  Agrega el objeto “o” , de tipo “T”, a la última posición de la lista.

- add(int index, <T> o)

  Agrega el objeto “o”, de tipo “T”, en la posición indexada con index (recordar que la primera posición es la cero). Si index no denota la última posición sino una intermedia, se realiza un corrimiento de los elementos existentes para hacer lugar al nuevo elemento.

- get(int index)

  Devuelve el elemento que se encuentra en la posición indicada por index. Esto NO elimina el elemento de la lista.

- size()

  Devuelve la cantidad de elementos que contiene la lista. Este número siempre superará en uno al índice de la posición del último elemento (puesto que el índice de las listas comienza en cero).

- contanis(<T> o)

  Devuelve true si la lista contiene el elemento “o”, y false en caso contrario.

- remove(int index)

  Elimina el elemento contenido en la posición indicada por index. Esto genera un corrimiento de los elementos ubicados en las posiciones superiores, de forma de no dejar lugar vacío.

- clear()

  Elimina todos los elementos de la lista, dejándola vacía.

- isEmpty()

  Devuelve true si la lista está vacía, y false en caso contrario.

Expresiones Lambda

A partir de Java 8, se incorporó el soporte a expresiones lambda. Las expresiones lambdas son funciones anónimas que no pertenecen a ninguna clase y son utilizadas porque necesitamos utilizar una funcionalidad una única vez. Normalmente, creamos expresiones lambda con el mero propósito de enviarla como parámetro a una función de alto orden (se entiende como función de alto orden a una función que recibe como parámetro a otra función).
El potencial de las expresiones lambdas es enorme, y se utiliza en distintos escenarios a lo largo de una aplicación. En este tutorial usaremos expresiones lambda para manejar colecciones a través de bloques de código obteniendo, así, una interacción con colecciones de más alto nivel; similar al manejo de colecciones provisto por Smalltalk.
Por ejemplo, si quisiéramos aumentarle el sueldo un diez porciento a todos los empleados; deberíamos recorrer la colección de Employee y enviarle a cada uno de ellos el mensaje.

Solución tradicional:

```
Foreach(Employee employee: employees){
     Employee.increaseIncome(10.0);
}
```

Solución Lambda (en negrita, se destaca la expresión lambda):

```
employees.stream().
          forEach(employee -> employee.increaseIncome(10.0));
```

Supongamos, ahora, que necesitáramos obtener los empleados con sueldo mayor a 1500 pesos.

Solución tradicional:

```
List<Employee> highIncomeEmployees =  new
ArrayList<Employee>();
Foreach(Employee employee: employees){
     If(employee.getFinalIncome()>1500){
          highIncomeEmployees.add(employee);
     }
}
Return highIncomeEmployees;
```

Solución Lambda (en negrita, se destaca la expresión lambda):

```
return employees.stream().
               filter(employee -> employee.getFinalIncome() > 1500.0).
               collect(Collectors.toList());
```

En los ejemplos ejecutables se muestra el código necesario para realizar distintas operaciones con colecciones, podrá ver como ordenar una colección, sumarla, encontrar el objeto que maximiza o minimiza un criterio, encontrar el primer objeto que cumple con un predicado, calcular el promedio de una colección, filtrar una colección o ejecutar una misma operación para cada uno de los objetos de la colección.

Ejemplos Ejecutables en Java

Para poder observar el funcionamiento concreto de todo lo descripto hasta aquí, se proveen ejemplos ejecutables en Java. Estos ejemplos están construidos en forma de Tests de Unidad que ejecutan distintas operaciones sobre instancias de ArrayList.

El objetivo es permitirle a usted no sólo observar la ejecución de estas operaciones, sino proveerle una base que le posibilite experimentar otras operaciones y escenarios generando nuevos tests o modificando los existentes. Por ejemplo: ¿qué ocurre si intento obtener un elemento de una posición mayor a la última?, o ¿qué ocurre si intento agregar un elemento en una posición mayor a la última?

El código Java entregado tiene dos clases:

- **Employee** (empleado): es una clase cuyas instancias se utilizarán como elementos de la lista.

- **CollectionTest**: es la clase que define los Tests de Unidad. Para esto define un escenario básico de prueba (en el método setUp()) que se utilizará para todos los tests, y un conjunto de tests que ejecuta distintas operaciones de la lista y se asegura que los resultados sean los esperados.

Tanto en el método setUp() como en los métodos de tests se podrá observar el funcionamiento de la lista descripto anteriormente.

Para ejecutar estos tests se deben seguir los siguientes pasos:

1. Crear un proyecto Java en Eclipse.
2. Crear un paquete llamado “unq.collections”.
3. Importar las dos clases anteriores a ese paquete.
4. Ejecutar la clase CollectionTest para correr los tests de unidad que contiene (actualmente todos los tests son exitosos).

***