ArrayLists

A Flexible Data Structure for Use with Objects

Alwin Tareen

The `ArrayList` Data Structure

- As we have seen, the array is a very powerful data structure that has many uses. However, arrays are assigned a fixed size when they are created, and this size cannot be changed.
- Java provides a data structure called an `ArrayList` that is very similar to an array, except that it can be easily re-sized while your program is running.
- Since an `ArrayList` can grow and shrink while your program runs, it is a more efficient use of memory, and can also be simpler to use.

Declaring and Instantiating an `ArrayList`

- Note that a programmer must use methods(`add`, `get`, etc.) rather than the square bracket notation to manipulate elements in an `ArrayList`.
- The `ArrayList` is part of Java's `util` library.
- Therefore, if you want to use an `ArrayList` in your program, you must include the following `import` statement at the top of your file:

```
import java.util.*;
```

Declaring and Instantiating an ArrayList

- The following statement declares an ArrayList of type String:

```
ArrayList<String> words = new ArrayList<String>();
```

- Since the ArrayList is a class, you must use the new operator followed by a constructor call to instantiate an ArrayList object.

ArrayList Methods

ArrayList provides two methods for adding elements to a list, both of which are named add.

add(x)

- This method works like an append, by adding the element to the end of the list. This method takes one parameter, which is the element to be added.

```
words.add("pizza");
```

ArrayList Methods

add(i, x)

- This method takes two parameters and works like an insert. The first parameter represents the location in the list where the element is to be inserted.
- Starting at the given index position, all elements after this position are pushed forward by one.

```
words.add(3, "pizza");
```

ArrayList Methods

get(i)

- The get method works like the square bracket notation with an array.
- The get method takes one parameter, which is the location of the element to be retrieved.

```
String element = words.get(5);
```

ArrayList Methods

remove(i)

- The remove method takes one parameter, which is the location of the element to be removed.
- Starting at the given index position, all elements after this position are moved back by one.
- This method also returns the deleted item.

```
String element = words.remove(7);
```

ArrayList Methods

set(i, x)

- The set method replaces an element at a given location.
- This method takes two parameters.
- The first parameter is the index of the element to be replaced.
- The second parameter is the element's new value.
- This method returns the replaced element.

```
String element = words.set(6, "burger");
```

ArrayList Methods

size()

- The `size` method returns the number of elements which are currently in the list.

```
int num = words.size();
```

isEmpty()

- The `isEmpty` method returns `true` if the list contains no elements, and `false` otherwise.

```
boolean result = words.isEmpty();
```

ArrayList Methods

contains(x)

- The contains method takes one parameter, which is an object element.
- It returns true if that element is within the ArrayList, and false otherwise.

```
boolean result = words.contains("pizza");
```

The Enhanced for Loop

The for-each loop

- There is a convenient shortcut for iterating through a sequence of elements, such as an array or an ArrayList.
- It is called the enhanced for loop, or the for-each loop.
- Suppose you wanted to sum up all of the values in an array named data. The following is a typical for loop operation that can perform this task.

```
double[] data = {5.1, 8.7, 6.3, 9.2};
double total = 0.0;
for (int i = 0; i < data.length; i++)
{
    double item = data[i];
    total += item;
}
```

The Enhanced for Loop

The for-each loop

- The following code demonstrates how you would use an enhanced `for` loop to carry out the same task.

```
double[] data = {5.1, 8.7, 6.3, 9.2};
double total = 0.0;
for (double item : data)
{
    total += item;
}
```

The Enhanced `for` Loop

Explanation of the `for-each` loop

- The loop body is executed for each element in the array `data`.
- At the beginning of each loop iteration, the next element is assigned to the variable `item`.
- Then, the loop body is executed. You can read this loop as: **for each `item` in `data`**.

The Enhanced `for` Loop

An important difference

- Note that there is an important difference between the `for-each` loop and the ordinary `for` loop.
- In the `for-each` loop, the element variable `item` is assigned to the values `data[0]`, `data[1]`, in turn, all the way up to the last element.
- In the ordinary `for` loop, it is the index variable `i` which is assigned to the values `0`, `1`, `2`, etc.

Using the Enhanced `for` Loop with an `ArrayList`

- You can use the enhanced `for` loop to visit all of the elements of an `ArrayList`.
- Assume that an `ArrayList` named `accounts` has been defined, and it has been populated with `BankAccount` objects.
- Consider the following `for` loop which computes the total value of all `accounts`.

```
double total = 0.0;
for (int i = 0; i < accounts.size(); i++)
{
    BankAccount item = accounts.get(i);
    total += item.getBalance();
}
```

Using the Enhanced `for` Loop with an `ArrayList`

- The following is the equivalent `for-each` loop that would perform the same action as the previous code:

```
double total = 0.0;
for (BankAccount item : accounts)
{
    total += item.getBalance();
}
```

Conditions for Using the Enhanced `for` Loop

Note that the `for-each` loop is suitable only if the following conditions hold:

- You want to traverse **all** the elements.
- You do not want to change or update any of the elements.

- If, for instance, you want to alter the elements while looping through them, then you must use a regular `for` loop.

ArrayLists: End of Notes