Java Programming AP Edition
U3C8 N-D Arrays and ArrayLists

ARRAYLIST

ERIC Y. CHOU, PH.D. IEEE SENIOR MEMBER

Why ArrayList?

Memory allocation. Programming convenience.

Arrays have the disadvantage that, once they have been created, their lengths can never be changed. We are often faced with situations, however, where the natural thing to do is to delete or insert array elements. One way to achieve this is to create a new array into which we then copy all the elements we want to keep together with any additional elements we want to insert.

Therefore, we need a different data structure that can behave like array but allows us to grow or shrink the ***"array"*** without copying all the elements. For this need, ArrayList serves this purpose very well.

Why ArrayList?

To overcome this difficulty, the Java language provides a class called ArrayList. An instance of ArrayList can be used to store data just like a regular Java array. Individual elements of arrays and ArrayList objects are accessed in similar ways, using an index. Unlike arrays, however, an ArrayList object allows us to change its length by deleting elements or inserting elements (at any point, not just at the end).

Declaration of ArrayList

ArrayList like a train. The datatype it carrys is like the cargo.

In declaring a variable of type **ArrayList** we use a statement like this:

```
ArrayList<String> aList;
```

The word between the *angle brackets*, **<…>**, indicates the data type of the elements that the **ArrayList** will store. In this case, **aList** is declared to be an **ArrayList** each of whose elements will be a **String**.

The following statement creates an **ArrayList** of **String**s and then assigns it to **aList**:

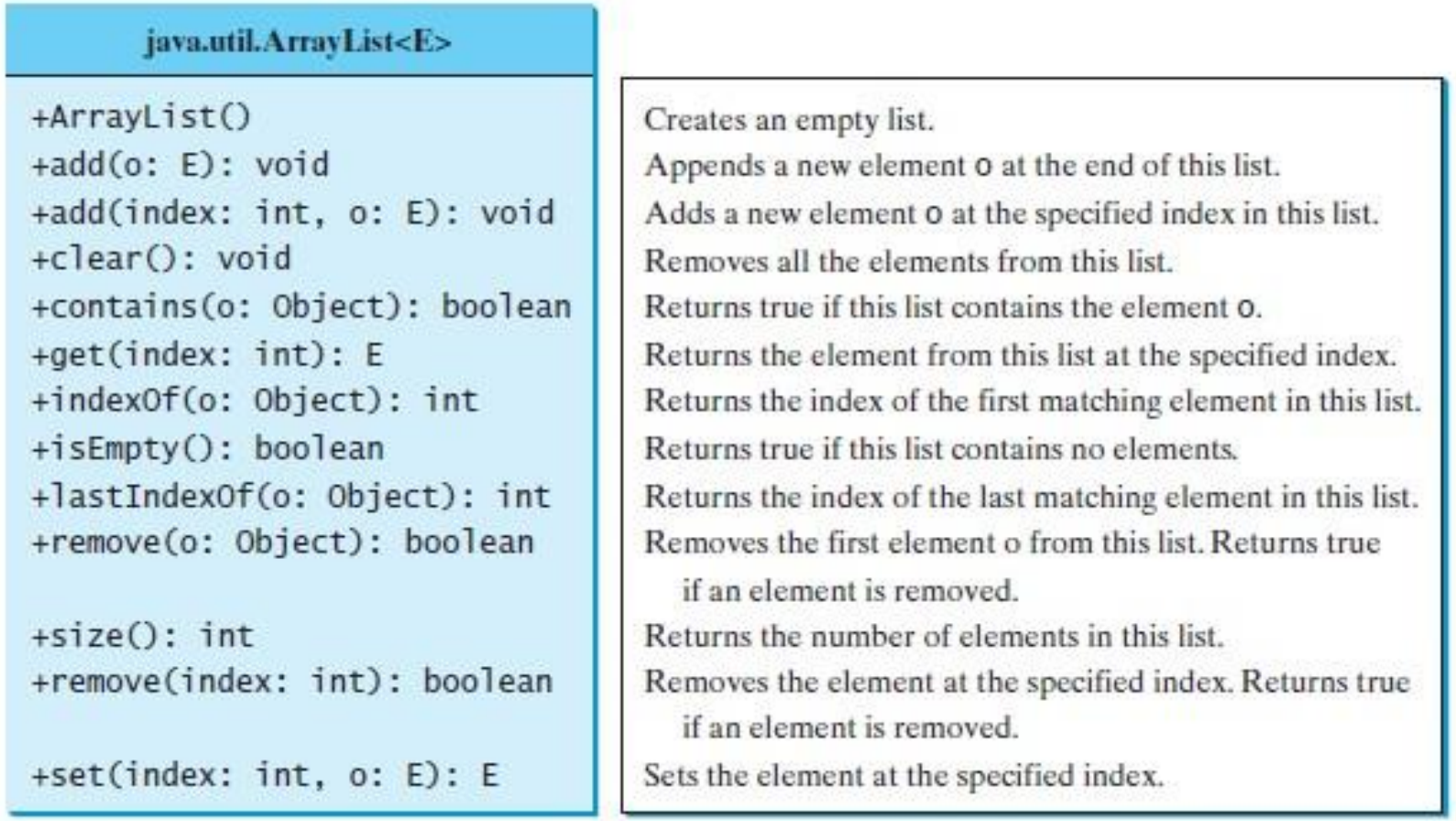
```
aList = new ArrayList<String>();
```

We can also declare and assign to the variable in a single statement:

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

The ArrayList Class

You can create an array to store objects. But the array's size is fixed once the array is created. Java provides the ArrayList class that can be used to store an unlimited number of objects.



Generic Data Type <E>

Can be String, Integer, Double, … (but not primitive data types)

If `E` is the name of a data type, the `ArrayList<E>` class is an example of a so-called *generic class* with *type parameter* whatever the replacement for `E` is. (Before we replace `E` by the name of an actual data type, it is called a *formal type parameter*.) It is also possible to declare an **`ArrayList`** without using a type parameter. Such a usage treats **`ArrayList`** as a so-called *raw class*. Raw **`ArrayLists`** require careful handling. However, they are not included in the AP subset and we have very little to do with them in this course.

NO ArrayList<int>

Generics in Java is not applicable to primitive types as in **int**. You should use the **wrapper** types as in **Integer**:

```
int a = 3;

ArrayList<Integer> aList = new ArrayList<Integer>();

// ArrayList<Integer>(); is a constructor call like

//   Scanner(System.in);


aList.add(a);   // put data of int type here is OK, because of auto-boxing
```

Differences and Similarities between Arrays and ArrayList

Operation	*Array*	*ArrayList*
Creating an array/ArrayList	`String[] a = new String[10]`	`ArrayList<String> list = new ArrayList<>();`
Accessing an element	`a[index]`	`list.get(index);`
Updating an element	`a[index] = "London";`	`list.set(index, "London");`
Returning size	`a.length`	`list.size();`
Adding a new element		`list.add("London");`
Inserting a new element		`list.add(index, "London");`
Removing an element		`list.remove(index);`
Removing an element		`list.remove(Object);`
Removing all elements		`list.clear();`