CMPT 225: Data Structures & Programming – Unit 11 – Array Lists

Dr. Jack Thomas

Simon Fraser University

Spring 2021

Today's Topics

- Sequences
- The Array List ADT
- The Array List in Java
- Analyzing the ArrayList class

An Array… List?

- Let’s not get ahead of ourselves.
- **Sequences** (sometimes called **lists**, but **not the lists we already met**) are the more formal designation for the kind of data storage we see in arrays, lists, stacks, and queues – a **linear series of data elements**, where each one holds a numbered position in that sequence, known as an **index**.
- If sequence elements can be **accessed by their index**, this is called an **Array List**.

So… is this an Array? A List? What?

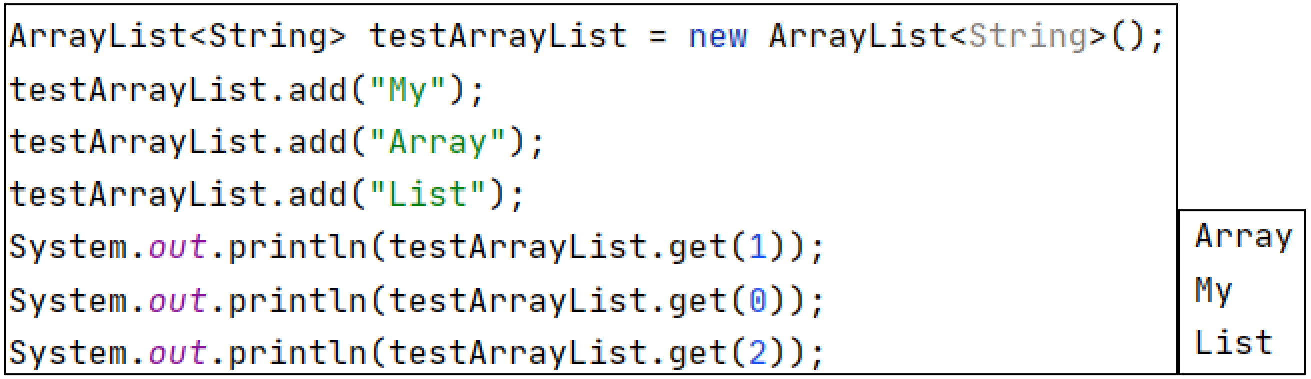
- The versions of **arrays and lists we discussed before were relatively primitive data structures**, more closely tied to the specific details of how each programming language works.
- The **Array List**, meanwhile, is the **fully-featured data structure version of the array**, like the Stack or the Queue.
- As such, it has an **ADT**, as well as a recognized set of **interface methods** and **standard class implementations** in different languages (including Java).

The Array List ADT

- A linear sequence of data elements, organized along and accessed by its index.
- Essentially the full data structure version of what arrays do.
- Standard methods include:
  - **Get**: Returns the element at a given index.
  - **Set**: Replaces the element at a given index with a given element, returns the old element.
  - **Add**: Adds a new element at the given index and increases the size.
  - **Remove**: Removes the element at a given index and decreases the size.
  - **Size**: Returns the number of elements stored in the Array List.
  - **isEmpty**: Returns whether the Array List is empty.

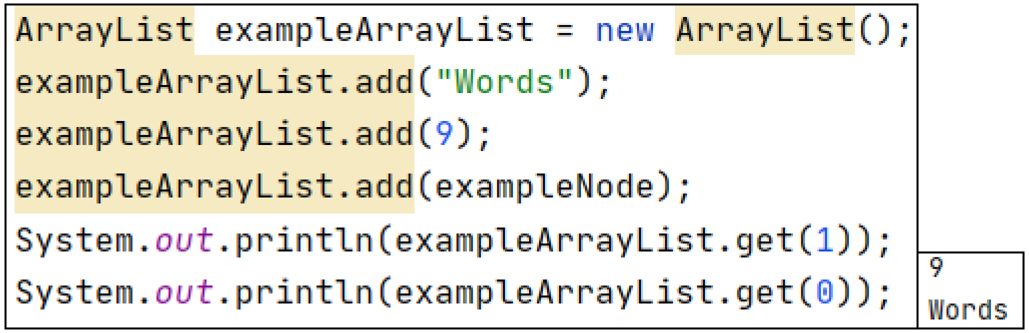
The Array List in Java

- There is a standard Java class, **ArrayList**, which is the cousin of the more list-like class, **LinkedList**.



```java
ArrayList<String> testArrayList = new ArrayList<String>();
testArrayList.add("My");
testArrayList.add("Array");
testArrayList.add("List");
System.out.println(testArrayList.get(1));
System.out.println(testArrayList.get(0));
System.out.println(testArrayList.get(2));
```

```
Array
My
List
```



```java
ArrayList exampleArrayList = new ArrayList();
exampleArrayList.add("Words");
exampleArrayList.add(9);
exampleArrayList.add(exampleNode);
System.out.println(exampleArrayList.get(1));
System.out.println(exampleArrayList.get(0));
```

```
9
Words
```

When to Use an Array List

- Sorted collections, supporting other data structures or algorithms.
- Essentially, whenever you want to use a **primitive array**, but don’t want to define a bunch of **professional-quality functions** for handling it.
- Gives you the standard set of things you might want to do with an array in an **object-friendly, interface-oriented package**.
- When deciding between a **LinkedList** or an **ArrayList**, note that searching and accessing from an ArrayList takes **O(1)** while adding and removing are **O(n)**, whereas **the reverse is true for LinkedList**.

One Final Note On Memory

- One of the defining challenges of using arrays is their **need for a fixed size** when they're declared.
- **Array Lists don't have a built-in solution** to this (ADTs don't care about internal implementations, remember), but the Java class **ArrayList does**.
- ArrayLists **double the size of their internal memory array** whenever the add() function is asked to **add an element that would exceed their capacity**.

Recap – The Final Index

- Array Lists are the full data structure version of arrays, with a formal ADT and concrete implementations in different languages.
- Java provides a standard ArrayList class, the cousin of the LinkedList class, which handles the standard operations expected of an array in an object-oriented manner.
- ArrayLists are better if we expect to do more searching and accessing than adding or removing, while the reverse is true of LinkedLists.
- ArrayLists handle the memory issue by doubling their capacity each time it reaches the maximum.