SIENAcollege
Computer Science

Computer Science 120
Introduction to Programming
Siena College
Spring 2011

Topic Notes: The `ArrayList`

Java's array constructs provide a powerful mechanism for us to manage much larger collections of data than we could if each item needed to have its own unique name.

But we have also seen how arrays can be quite inconvenient to use. When we do not know ahead of time how many items we will need to store in our array, we need to keep track not only of the size of the array (though that's easy using the array's `.length`), but also the number of array slots that are actually in use. If the array becomes full, we may need to reconstruct our array and copy over the contents to the new array. We also saw that adding to or removing from the middle of an array takes some care to do correctly.

However, these are very common operations on collections of data like those we would store in an array. Those of you who will go on to take data structures will learn about a variety of ways that collections of data can be stored that vary in complexity, flexibility, and efficiency. We will consider just one of those structures here: the `ArrayList`.

`ArrayList` is a class that implements an *abstract data type* provided by the standard Java utility library. It has very similar capabilities to an array, but has somewhat higher-level methods and functionality.

Let's see how to use them through an example. If we replace the array in our program that could draw any number of `RoadSegment` objects on our canvas, and drag them around.

See Example: DragAllRoadsArrayList

This program has the same functionality, but some of the complexity of dealing with arrays has been removed – those details are taken care of by the `ArrayList` implementation.

- A named constant (the initial array size), the array declaration, and the declaration of the variable that tracked the number of elements in the array have all been replaced by one instance variable:

  ```
  private ArrayList<RoadSegment> segments;
  ```

  The syntax is similar to what we have seen, but we do need to tell the `ArrayList` what kind of values it will hold: in this case, `RoadSegment`s.

- In the `begin` method, we replace the construction of the array and the initialization of the number of elements variable with:

  ```
  segments = new ArrayList<RoadSegment>();
  ```

Note that we do not need to tell the `ArrayList` how big to be – it will choose an initial size and will grow as needed if we add more elements than that size.

- The `for` loop in `onMousePress` now needs to query the `ArrayList` for the number of elements it contains, which is available through the `size` method.

```
for (int segmentNum = 0; segmentNum < segments.size(); segmentNum++
```

- The actual array references where we look up values in the array are replaced by a call to the `ArrayList`'s `get` method:

```
if (segments.get(segmentNum).contains(point)) {
    selectedSegment = segments.get(segmentNum);
```

- And the array assignment and update of the number of elements are replaced by a call to the `ArrayList`'s `add` method:

```
segments.add(new RoadSegment(point, canvas));
```

The big win here (for us as programmers) is the fact that all of the "check capacity" code – where we check to make sure the array is large enough then create the larger array, and copy over all of the array's contents – goes away completely! That functionality is all provided by the `ArrayList` implementation!

Other methods

The example above demonstrated just a few of the capabilities of the `ArrayList` class: construction, `add`, `get`, and `size`.

The full documentation for the `ArrayList` can be found at `http://download.oracle.com/javase/1.5.0/docs/api/java/util/ArrayList.html`

We'll look at just a couple of additional methods, some of which will come up in later examples.

- `clear` – remove all elements from the list.
- `contains` – determine if a given object is in the list
- `indexOf` – search for first occurrence of a given object in the list and return its index
- `remove` – remove object either by index or value
- `set` – replace the contents at an index with a new element