Java Arrays & Collections

Jonathan Lung – CSC207H: Software Design

February 15\textsuperscript{th}, 2013
Decks of Cards & Flocks of Sheep

- We often need to track a large number of things in software.
- How do we track them?
- In Python in CSC148, we saw, wrote, and used
Decks of Cards & Flocks of Sheep

• We often need to track a large number of things in software.
• How do we track them?
• In Python in CSC148, we saw, wrote, and used
  • lists,
  • dicts,
  • trees,
  • queues,
  • linked lists,
  • and more.
Arrays

- Conceptually, an array is a list of a fixed number of items that are \textit{addressed} by an integer \textit{index}.

\begin{tabular}{|c|c|}
\hline
0 & 0x00102010 \\
1 & 0x3d91ee70 \\
2 & 0x249dfa20 \\
3 & 0x29764200 \\
\hline
\end{tabular}

- Some arrays are 0-based while others are 1-based; this depends on whether the first element of the array is 0 or 1.
- Java, like Python, is 0-based.
Arrays

Usually, consecutive items are stored in memory such that they can be efficiently accessed by index.

\[
\begin{align*}
\text{start\_address} + (0 \times 8) &= 0x00102010 \\
\text{start\_address} + (1 \times 8) &= 0x3d91ee70 \\
\text{start\_address} + (2 \times 8) &= 0x249dfa20 \\
\text{start\_address} + (3 \times 8) &= 0x29764200
\end{align*}
\]
Declaring an Array

- An array in Java is not a primitive; it is an object. We can access its length through its `length` field).
- To declare an array, we use the following syntax:
  \[
  \text{TypeOfElements}[] \text{nameOfArray} ;
  \]
- **Remember**: Declaring a variable does not create any new objects.
Initializing an Array

- To create a new array and keep a reference to it, we can write:
  
  ```java
  nameOfArray = new TypeOfElements[arrayLength];
  ```

- Note the lack of parentheses.

- For convenience, we can initialize an array by specifying each of its elements and surrounding it in curly braces. E.g.,
  
  ```java
  String[] reindeer = {"Dasher", "Prancer", "Comet", "Cupid");
  ```

- This method of specifying array contents can only be used during initialization.
Getting/Setting Array Elements

- To set or get the $n^{th}$ element (counting from 0), we write `arrayName[n]` where we would otherwise use a variable.
- Any integer value from 0 to `arrayName.length - 1`, inclusive, is valid.
Multidimensional arrays

- To declare an n-dimensional array, we write
  \[ \text{TypeOfElements} \underbrace{[]} \ldots \underbrace{[]} \]
  \( n \) sets of []s

- To create a new n-dimensional array, we write
  \( \text{new TypeOfElements} [s_1] \ldots [s_n] \)

- In Java, these are 1-dimensional arrays containing other 1-dimensional arrays. Thus, in the 2x5 two-dimensional array (can be visualized as two rows and five columns)
  \( \text{int}[][] \text{ anArray}= \text{new int}[2][5], \text{anArray}[0] \text{ is a} \)
  1-dimensional int array of length 5.
The Java Collections Framework

- Arrays do not provide much in the way of conveniences such as the ability to grow or to sort its elements.
- Java’s Collections Framework provides access to different advanced data types (ADTs) that provide these features and more.
Getting help

- The Java documentation is extremely useful. You might want to start at [http://java.sun.com/javase/7/docs/api/java/util/Collection.html](http://java.sun.com/javase/7/docs/api/java/util/Collection.html).

- You should now be familiar with the following terms which appear in the documentation:
  - interfaces,
  - abstract classes, and
  - concrete classes.

- Remember, you can only create instances of concrete classes.
Generics

• Generics are a way of extending static typing to classes when the exact type of data the classes will operate on is unknown.

• They are used extensively throughout the newest versions of the Java Collections framework.

• For example, we might want to create a `List` that contains elements of type `E`, where `E` is any class or interface; calling the `get` method on this `List` should return objects of type `E`.

• The `Map` interface is an example where two generic types need to be specified: one for the keys’ type and one for the values’ type.
Generics

A type enclosed within angle brackets in the API such as the E in List<E> means the programmer should replace E with the same data type every time it appears. For example, use List<String> strs = new ArrayList<String>(); String s = strs.get(0); to create an ArrayList of Strings and access an element.
A note about program design

In the previous example, we care that strs is a List; we happened to choose the concrete class ArrayList; by writing List<String> instead of ArrayList<String> as the type of strs, we can use a different type of List in the future if something else becomes more appropriate.