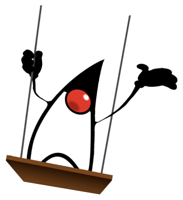
Java Programming 2 – Lecture #9 – Jeremy.Singer@glasgow.ac.uk



Java Arrays

An array is a fixed length sequence of consecutive memory locations, indexed by an integer subscript. Arrays are supported directly by the underlying Java Virtual Machine, so they are efficient to use.

Declaring Array Types

Each array has a *type*, which specifies the type of the individual elements and the dimensionality of the array. For example, `int[]` is a one-dimensional `int` array and `String[][]` is a two-dimensional `String` array. Element types may be Java primitive types or Object (reference) types.

When an array is declared (perhaps as a method parameter, a local variable or a class member) it is given a *name*. The name either comes after the type (i.e. `String [] args`) or is inserted within the type (i.e. `String args[]`). This latter form is a C-style hangover.

Initializing Array Values

An array declaration does not reserve space for the array elements, or specify the length of the array. Instead it only declares a reference to the (currently uncreated) array. This means that the uninitialized array reference is a `null` pointer value. The array may be created via a call to `new` or with an explicit initializer.

```
int [] a = new int[10];

String [] as = { "each", "peach", "pear", "plum" };
```

Subscripting Array References

Once the array has been created, array elements can be indexed via integer subscripts, e.g. `a[3]`, `as[1]`. Subscripts start from 0 (unlike Fortran, COBOL or Matlab). The maximum allowable subscript is slightly less than `Integer.MAX_VALUE`. However if a subscript is greater than or equal to the length of the array, then an `ArrayIndexOutOfBoundsException` unchecked Java exception is thrown at runtime.

The length of an array is constant, stored in a field of that name, e.g. `a.length`, `as.length`. Note that for `String` objects, `length()` is a method whereas for arrays, `length` is a field.

Iterating over Arrays

The standard idiom for iterating over an array is to use a `for` loop.

```
for (int i=0; i<a.length; i++) {

  a[i] = …;

}
```

An alternative, more concise, notation is to use the for-each loop idiom, in cases where the array indexing does not need to be explicit.

```
for (String s: as) {
  System.out.println(s);
}
```

Helper Methods for Arrays

Since an array is effectively an object in Java, it inherits all the methods from `java.lang.Object`. The `java.util.Arrays`[1] class contains a set of static helper methods for array manipulation, including `Arrays.toString()` and `Arrays.fill()`.

The ArrayList Data Structure

The major limitation of Java arrays is that they have a fixed length. The `java.util.ArrayList`[2] class is a more flexible (although less efficient) library class that implements variable length arrays. The ArrayList class is part of the Java Collections framework[3]. Creation, subscripting and other operations are now all library methods rather than built-in syntax. Further, `ArrayList` element types must be objects rather than primitive values. See the example below.

```
ArrayList<Integer> nums = new ArrayList<Integer>();
nums.add(1);
nums.add(1);
int i = 2;
int fib = 1;
while (fib < LIMIT && nums.size() < SIZE_LIMIT) {
  fib = nums.get(i-1) + nums.get(i-2);
  nums.add(fib);
}
```

Note that `ArrayList` structures can be converted to arrays, and vice versa using the `Arrays` helper methods.

[1] See http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html
[2] See http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html
[3] See http://docs.oracle.com/javase/7/docs/technotes/guides/collections/index.html