COMP10062: Week 6 Guide

Sam Scott, Mohawk College, 2017

0 Reading for this Week

For this week, you should **read sections 7.1, 7.2, and parts of 7.3** (see details below)**.** This guide also contains sections on **Arrays of Objects** and **Multiple Association**. This important topic is not covered explicitly in Chapter 7 of the textbook.

1. Array Basics (Read Section 7.1, 7.2)

a. Comparison to Python lists

Arrays are a bit like Python lists…

- Arrays are objects
- Arrays store indexed elements, starting at element 0.
- Arrays can be processed easily using a special kind of loop.

…but there are some important differences.

- Arrays are fixed in length.
- Arrays are fixed in type.
- Arrays do not support the slice operator.
- Arrays do not have methods.

b. Declaring Array Variables (p. 480)

Use `[]` beside the type to indicate this is an array.

```
int[] a;      double[] b;      String[] c;
```

c. Creating and Initializing Arrays (pp. 480-483)

Use the `new` keyword, but instead of calling a constructor with round brackets, specify the size of the array using square brackets. When you create an array, the contents are initialized to the "0" element for the type (0 for numbers, `false` for boolean, or `null` for objects).

```
a = new int[50];                        ← create
for (int i=0; i<a.length; i++)          ← initialize
     a[i] = -1;

b = new double[maxSize];                ← create
for (int i=0; i<b.length; i++)          ← initialize
     b[i] = sc.nextDouble();
```

You can also use an **array literal**. This is a comma-separated list of values enclosed in curly brackets.

```
String[] c = {"I", "am", "the", "walrus"};
```

The above only works when you declare and assign at the same time. But if you combine it with the `new` keyword, you can do it any time.

```
String[] c;                                          // c = null
c = {"I", "am", "the", "walrus"};                    // syntax error
c = new String[]{"I", "am", "the", "walrus"};        // this is ok
```

d. Processing with a For Loop (p. 488)

Use the `.length` field of an array to find out how many elements the array contains.

```
double sum = 0;
for (int i=0; i<b.length; i++)
      sum += b[i];
```

e. Processing with an Enhanced For Loop (p. 488)

The **enhanced `for` loop** (a.k.a. the "for each" loop) assigns one element at a time from an array into a temporary variable and then executes the loop body for each value. This loop is great when you want to visit the elements of an array in increasing order, and when you do not plan to change the array contents in any way.

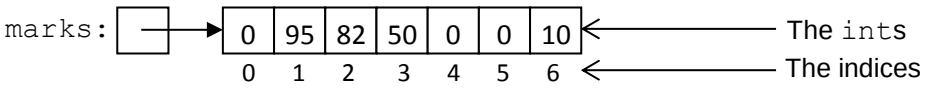
```
double product = 1;   // what is "product" after this loop?
for (int element: a)
      product *= element;

for (int element: a)  // what does "a" contain after this loop?
      element = 100;
```

f. Arrays as Parameters and Return Values (pp. 505, 510-513)

Arrays are a special kind of object in Java. Array variables are reference variables.

```
int[] marks = {0, 95, 82, 50, 0, 0, 10};
```



Like object types, Array references can be passed as parameters and returned as return values.

```
public void printArray(int[] array) {
      for(int item: array)
            System.out.println(item);
}

public int[] getLittleArray() {
      int [] d = {1, 3, 5};
      return d;
}
```

If you change an array in a method, the changes will be reflected in the method that called it.

```
public void changeThisArray(int x, int[] y) {
      y[x] = 100;
      x = x + 1;
}

…
int index = 0;
changeThisArray(index, marks);
System.out.println(index+" "+marks[0]);  // what will this print?
```

Arrays as Instance Variables (listing 7.9, pp 519-521)

You can use an array type as an instance variable. But don't forget that if you only *declare* the array, it will be `null` initially (see **ArrayInstanceVariable1.java**). You still have to *create* and possibly *initialize* an array or have an existing array passed as a parameter to the constructor and then assign it to the instance variable.

If you're accepting an array from a constructor, you should worry about privacy leaks and think about whether or not you should make your own private copy of the array. If you make your own copy (see **ArrayInstanceVariable3.java**), then you can completely control access to it, but if you use the array reference that was passed to the constructor (see **ArrayInstanceVariable2.java**), some other parts of the code may also be able to change your variable.

2. Arrays of Objects

a. Declaring Variables for Arrays of Objects

You are not restricted to just primitive types and Strings when you declare an array variable.

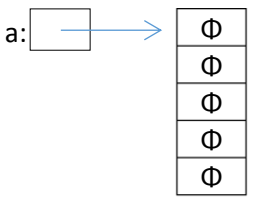
```
Circle[] a;        Customer[] b;      GradeCounter[] c;
```

a: Φ ← initialized to null reference (Φ = phi = null).

b. Creating Arrays of Objects

Create arrays of objects in the same way as any other array type.

```
a = new Circle[5];
b = new Customer[numCustomers];
```



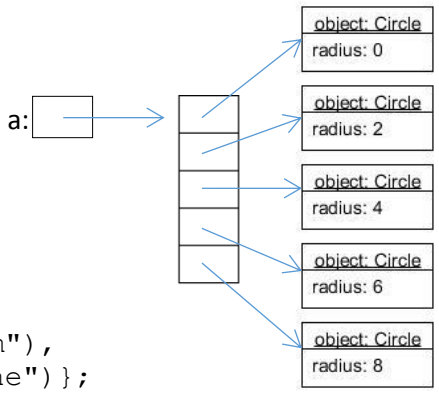
← note that this is an array of null references

Now you have an array of reference variables, as shown on the right.

c. Initializing Arrays of Objects

Initializing an array of objects means creating a new object to assign to each element in the array.

```
for (int i=0; i<a.length; i++)
     a[i] = new Circle(i*2);
```



The above code fills each array element with a different `Circle` object. See picture at right.

```
b = new Customer[]{new Customer("Sam"),
                   new Customer("Anne")};
```

The above code creates a new array with two new `Customer` objects. This is not usually the most practical way to create an array of objects.

d. Accessing Objects in an Array

You can use the dereference operator "." directly on an array element.

```
Circle c = a[0];
System.out.println(c.getRadius());
```

← these two lines…

do…

```
System.out.println(a[0].getRadius());
```

← the same thing…

*** Watch out for null pointer exceptions from uninitialized arrays! ***

e. Processing Arrays of Objects

To process an array of objects, use a `for` loop like this:

```
double totalArea = 0;
for (int i=0; i<a.length; i++)
      totalArea += a[i].getArea();
System.out.println(totalArea);
```

Or you can use an enhanced `for` loop like this:

```
double totalArea = 0;
for (Circle circle : a)
     totalArea += circle.getArea();
System.out.println(totalArea);
```

f. Enhanced For Gotchas

If you use an enhanced for loop, you can make some changes to the individual objects in the array. Take a look at the examples below. Why does one work, but not the other two? Explain with the aid of a diagram…

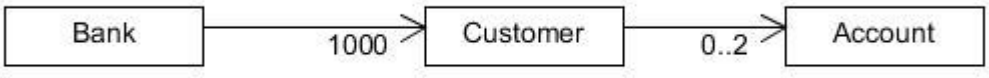
```
for (Circle circle : a)
     circle.setRadius(100);
              ↑
           WORKS!
```

```
for(int e : intArray)
    e = 100;
      ↑
  DOESN'T WORK!
```

```
for (Circle circle : a)
      circle = new Circle(5);   ←——  DOESN'T WORK!
```

Explain with the aid of a diagram…

g. Arrays of Associations



Arrays can be used to implement UML class diagrams that call for associations with high multiplicities. In the example above, `Bank` holds 1000 `Customer` objects. It would be impractical to declare 1000 instance variables, so we use an array instead. The array variable must be *declared* along with other instance variables and the array could also be *created* when declared. But the `Customer` objects required to *initialize* the array need to be created elsewhere – possibly in the `Bank` constructor.

```
private Customer[] customers = new Customer[1000];
public Bank() {
      // loop to create new customers goes here…
}
```

Another option is be to *declare* the instance variable and allow the array to be *created* and *initialized* elsewhere and passed into the constructor, like this:

```
private Customer[] customers;
public Bank(Customer[] customers) {
      // copy or assign to this.customers here
}
```

But beware of **privacy leaks**!