Lecture 15

Arrays (and For Loops)

For Loops

```
for (initiating statement; conditional statement; next statement)
                                                    // usually incremental
{
    body statement(s);
}
```

The **for** statement provides a compact way to iterate over a range of values. Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied.
The general form of the for statement can be expressed as above.

Yay! You can use this instead of a counting-while-loop!!

For or While?

```
class WhileDemo {
    public static void main(String[] args){
        int count = 1;
        while (count < 11) {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

```
class ForLoopDemo {
    public static void main(String[] args {
        for(int count = 1; count < 11; count++){
            System.out.println("Count is: " + count);
        }
    }
}
```

Introduction to Arrays

So far, you have been working with **variables** that hold only **one value**. The **integer** variables you have set up have held only **one number** (and *later* in the quarter we will see how **string** variables will hold **one long string** of text).

An **array** is a collection to hold *more than one value at a time*. It's like a **list** of items.

Think of an array as like the columns in a spreadsheet. You can have a spreadsheet with only one column, or several columns.

The data held in a single-list (one-dimensional) array might look like this:

	grades
0	100
1	89
2	96
3	100
4	98

Putting Values into Arrays

	grades
0	100
1	89
2	96
3	100
4	98

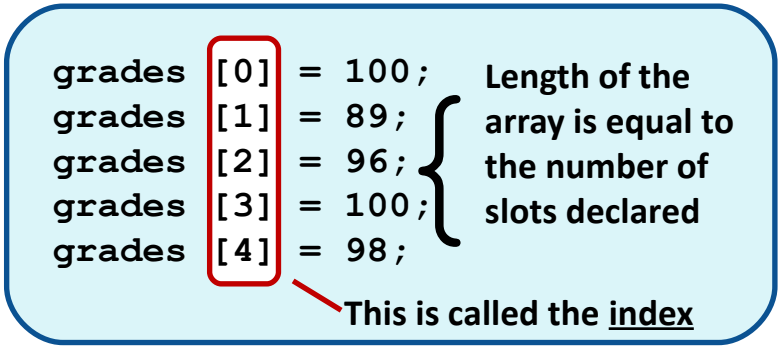
You can also declare the **int** separately and call it by its given name, like this:

```
int summer2012 = 5;
int[ ] grades = new int[summer2012];
```

We are telling Java to set up an array with 5 positions in it.

After this line is executed, Java will assign default values for the array (o for an integer array). To assign values to the various positions in an array, you do it in the normal way:

```
grades[0] = 100;
grades[1] = 89;
grades[2] = 96;
grades[3] = 100;
grades[4] = 98;
```



If you know what values are going to be in the array, you can set them up like this instead:

```
int[ ] grades = { 100, 89, 96, 100, 98 }; // Java treats as a
                                          // new instance
```

Arrays

- Non-array variables
  - Have a datatype and a name
  - Contain ONE value
  - Are accessed by name alone
- Array variables
  - Have a datatype and a name
  - Contain MANY values of that type
  - Also have a length
  - Are accessed by name and index

Array notes

- Arrays have a built in value length:
  - Int []myArray = new int [5];
  - MyArray.length returns 5
- Constructing an array with new makes space for length * dataType memory
- In Java (and most other modern languages), indices are zero based – the first value is in myArray[0], the last in myArray[length-1].
  - This means you can write loops with just a <:
  - for (int j=0; j < myArray.length; j++) {

Tracing an array

float[] f = new float[length]

- ? what is the size of the array? f.length == ?
- ? what is the datatype of the array?

- f[0] = PI*r*r;
- f[i] = 2.2;
- f[(3i)] = 3.3;
  - ? what is the index? What if the index is a variable, or an equation?
  - ? what is the new value?

Passing arrays around

- You can treat arrays like other variables with type <dataType> []

```
int summer2012 = 5;                                  <dataType> int
int[ ] grades = new int[summer2012];                 Specify any integer length

public void methodRequestsArray (int [] arrayIn)    // Passing an array
public void methodRequestsInt (int valIn)
public int[] methodReturnsArray ()                   // Returning an array

methodRequestArray (grades);                         // calling those methods.
methodRequestsInt (grades[0]);
grades = methodReturnsArray();
```

Arrays of strings

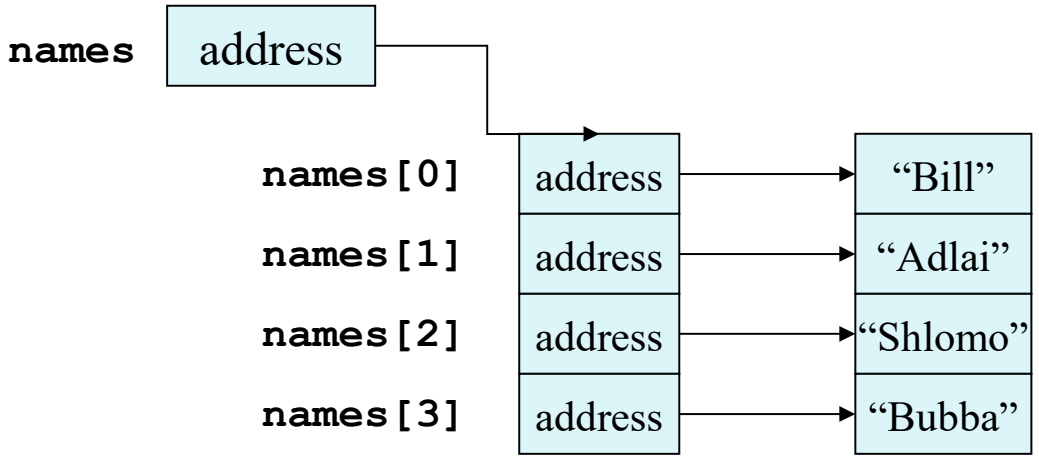
Java allows us to create arrays of String objects. Here is a statement that creates a String array initialized with values:

```
String[] names = {"Bill", "Adlai", "Shlomo", "Bubba"};
```

In memory, an array of String objects is arranged differently than an array of a primitive data type. To use a String object, you must have a reference to the string object. So, an array of String objects is really an array of references to String objects.

The **names** variable holds the address to the array.

A **String** array is an array of references to **String** objects.

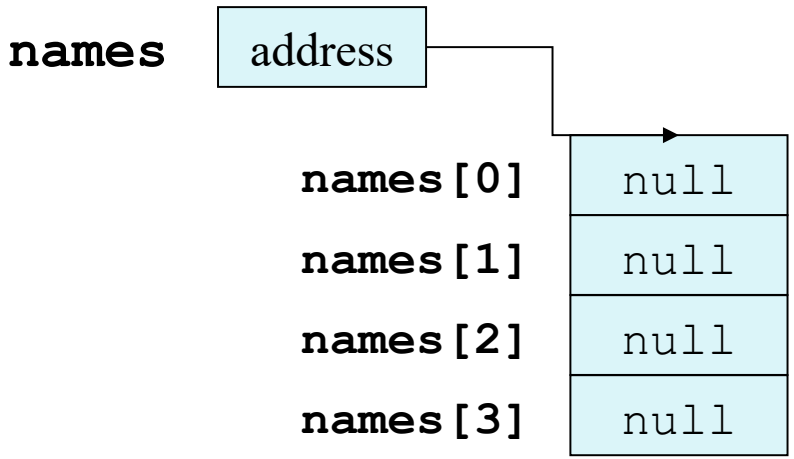


Initializing String arrays

If an initialization list is not provided, the new keyword must be used to create the array:

```
String[] names = new String[4];
```

The **names** variable holds the address to the array.

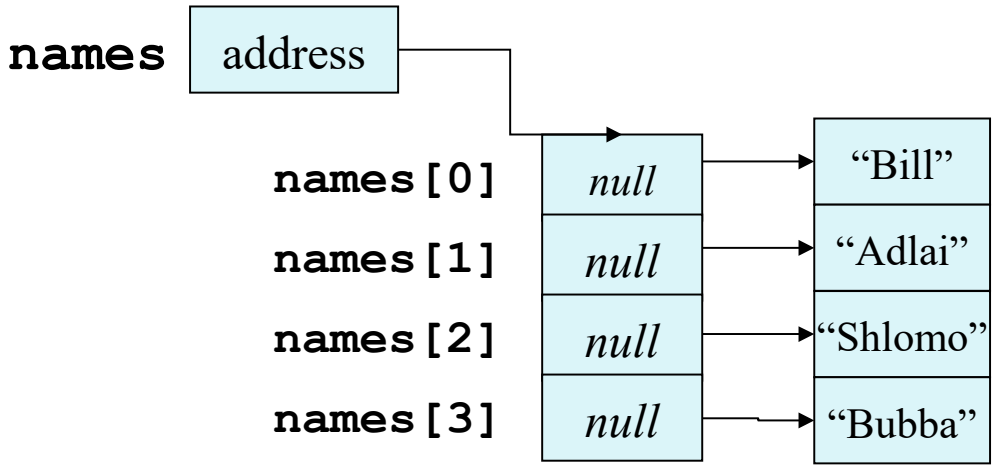


Putting contents in String arrays

When an array is created in this manner, each element of the array must be initialized:

```
String[] names = new String[4];

names[0] = "Bill";
names[1] = "Adlai";
names[2] = "Shlomo";
names[3] = "Bubba";
```



String methods in Arrays

- **`String`** objects have several methods, including:
  - **`toUpperCase`**
  - **`compareTo`**
  - **`equals`**
  - **`charAt`**
- Each element of a **`String`** array is a **`String`** object.
- Methods can be used by using the *array name* and *index* as before.

```
System.out.println(names[0].toUpperCase());
char letter = names[3].charAt(0);
```