

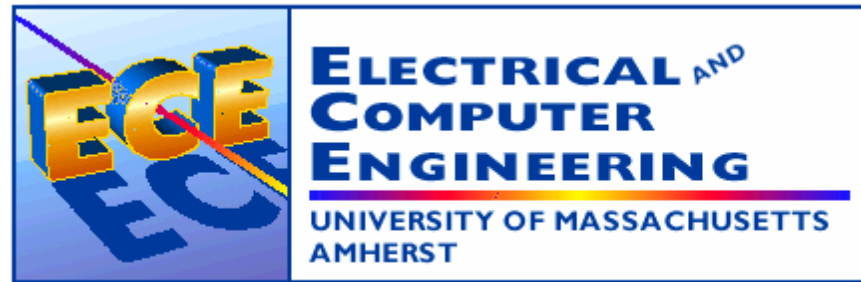
---

# ECE 122

## Engineering Problem Solving with Java

### Lecture 11

### More For Loops and Arrays



# Outline

---

- **Problem: How can I perform the same operations on a set of data values**
- **Considering “for loops”**
  - Useful for applying the same operations repetitively to arrays
- **Very common structure in programming**
- **Arrays**
  - Remember that they have a fixed size

# Indices

---

- We can use an **int** variable or any expression that evaluates to an **int** value as an index:

```
a [3]
a [k]
a [k - 2]
a [ (int) (6 * Math.random()) ]
```

# Indices

---

- In Java, an array is declared with fixed length that cannot be changed.
- Java interpreter checks the values of indices at run time
- Throws **IndexOutOfBoundsException** if an index is negative or if it is greater than the length of the array – 1.

# Why Do We Need Arrays?

---

- The power of arrays comes from the fact that the value of a subscript can be computed and updated at run time.

Before (no arrays):

```
int sum = 0;  
sum += score0;  
sum += score1;  
...  
sum += score999;
```

1000  
times!

After (with arrays):

```
int n = 1000;  
int sum = 0, k;  
for (k = 0; k < n; k++)  
    sum += scores[k];
```

# Why Arrays?

---

- **Arrays give *direct access* to any element — no need to scan the array.**

Before (no arrays):

```
if (k == 0)
    value = score0);
else if (k == 1)
    value = score1;
else
    ... // etc.
```

After (with arrays):

```
value = scores[k];
```

# Declaring Arrays

---

- The `scores` array could be declared as follows:

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

- Note the syntax, the 'reference' and the new object! It also says that there will be 10 scores referenced.
- The type of the variable `scores` is `int []` ("an array of integers" or "an array of ints.")
- Note that the array type does not specify its size, but each object of that type has a specific size
- The reference variable `scores` is set to a new array object that can hold 10 integers

# Declaring Arrays

---

- **Some other examples of array declarations:**

```
float[] prices = new float[500];
```

```
boolean[] flags;
```

```
flags = new boolean[20];
```

```
char[] codes = new char[1750];
```

**Can you provide examples of each type? Can you draw such an array as was provided earlier in these slides?**

**You saw the entries for the integer array, scores. What would a similar array of flags look like. Draw it. Of characters? Draw it. (You will see this again!)**

# Bounds Checking

---

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
  - That is, the index value must be in range 0 to N-1
- The Java interpreter throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds
- This is called automatic *bounds checking*

# Bounds Checking

---

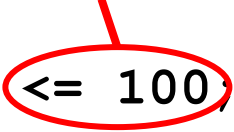
- For example, if the array `codes` can hold 100 values, it can be indexed using only the numbers 0 to 99
- If the value of `count` is 100, then the following reference will cause an exception to be thrown:

```
System.out.println (codes[count]);
```

- It's common to introduce *off-by-one errors* when using arrays

```
for (int index=0; index <= 100; index++)  
    codes[index] = index*50 + epsilon;
```

problem



# Bounds Checking

---

- Each array *object* has a public constant called `length` that stores the size of the array
- → It is referenced using the array name:  
`scores.length`
- Note that `length` holds the number of elements, not the largest index

## Alternate Array Syntax

---

- The brackets of the array type can be associated with the element type or with the name of the array
- Therefore the following two declarations are equivalent:

```
float [] prices;  
float prices[];
```

- The first format generally is more readable and should be used

## Declaration and Initialization

---

- **When an array is created, space is allocated to hold its elements.**
- **If a list of values is not given, the elements get the default values.**

```
scores = new int [10] ;  
           // length 10, all values set to 0  
  
words = new String [10000];  
           // length 10000, all values set to null
```

# Initializer Lists

---

- An *initializer list* can be used to instantiate and fill an array in one step
- The values are delimited by braces and separated by commas
- Examples:

```
int[] units = {147, 323, 89, 933, 540,  
              269, 97, 114, 298, 476};
```

```
char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```

## Initialization

---

- **An array can be declared and initialized in one statement:**

```
int scores [ ] = new int [10] ; // length 10
private double gasPrices [ ] = { 1.49, 1.69, 1.74 };
String words [ ] = new String [10000];
String cities [ ] = {"Atlanta", "Boston", "Cincinnati" };
```

# Initializer Lists

---

- **Note that when an initializer list is used:**
  - → the `new` operator is not used
  - → no size value is specified
- **The size of the array is determined by the number of items in the initializer list**
- **An initializer list can be used only in the array declaration**

# Arrays as Parameters

---

- **An entire array can be passed as a parameter to a method**
- **The reference to the array is passed,**
  - **Makes the formal and actual parameters aliases of each other**
- **Changing an array element within the method changes the original**
- **An individual array element can be passed to a method as well,**
  - **The type of the formal parameter is the same as the element type.**
  - **In this case, the call is 'by value.'**

# Arrays as Objects

---

- In Java, an array is an object. If the type of its elements is **anyType**, the type of the array object is **anyType[ ]**.
- There are two ways to declare an array:

```
anyType [ ] arrName;
```

or

```
anyType arrName [ ];
```

The difference becomes significant only when several variables are declared in one statement:

```
int [ ] a, b; // both a, b are arrays  
int a [ ], b; // a is an array, b is not
```

# Arrays as Objects

---

- As with other objects, the declaration creates only a reference, initially set to **null**.
  - An array must be created before it can be used.
- There are two ways to create an array:

```
arrName = new anyType [ length ] ;
```

Brackets,  
not parens!

or

```
arrName = new anyType [ ] { val1, val2, ..., valN } ;
```

# Array's Length

---

- The length of an array is determined when that array is created.
- The length is either given explicitly or comes from the length of the {...} initialization list.
- The length of an array **arrName** is referred to in the code as **arrName.length**.
- **length** appears like a public field (not a method) in an array object.

# Initializing Elements

---

- Unless specific values are given in a `{...}` list, all the elements are initialized to the default value: `0` for numbers, `false` for `booleans`, `null` for objects.
- If its elements are objects, the array holds references to objects, which are initially set to `null`.
- **Each object-type element must be initialized before it is used.**

# Passing Arrays to Methods

---

- **As other objects, an array is passed to a method as a reference.**
- **The elements of the original array are not copied and are accessible in the method's code.**

# Passing to Methods

---

- **Example:**

```
/**  
 *   Swaps a [ i ] and a [ j ]  
 */  
public void swap (int a[ ], int i, int j)  
{  
    int temp = a [ i ];  
    a [ i ] = a [ j ];  
    a [ j ] = temp;  
}
```

# Returning Arrays from Methods

---

- As for other objects, an array can be returned from a method.
- The returned array is usually constructed within the method or obtained from calls to other methods.
- The return type of a method that returns an array with **someType** elements is designated as **someType [ ]**.

## Returning from Methods

---

- **Example:**

```
public double[ ] solveQuadratic
    (double a, double b, double c)
{
    double d = b * b - 4 * a * c;
    if (d < 0) return null;

    d = Math.sqrt(d);

    double roots[ ] = new double[2];
    roots[0] = (-b - d) / (2*a);
    roots[1] = (-b + d) / (2*a);
    return roots;
}
```

Or simply:

```
return new double[ ]
    { (-b - d) / (2*a),
      (-b + d) / (2*a) };
```

# Summary

---

- **For loops allow for array modification**
  - Typically possible to modify one element at a time
- **For loop contains an initializer, condition, and condition modifier**
- **Arrays represent data is a series of memory locations**
  - All data has same array name
  - Specific locations located with subscripts
- **For loops and arrays go together well**
  - Fixed number of loop iterations and array size