

Arrays

Introduction to Programming and
Computational Problem Solving:
Accelerated Pace

CSE 11

Lecture 5

Arrays

- An array is a data structure that represents a collection of the same types of data

[0]	5.6
[1]	4.5
[2]	3.3
[3]	13.2
[4]	4.0
[5]	34.33
[6]	34.0
[7]	45.45
[8]	99.993
[9]	11123

← Element value

1-dimensional array

Declaring array variables

`datatype[] arrayRefVar;`

- For example

`double[] myList;`

If a variable does not contain a reference to an array, the value of the variable is `null`

`datatype arrayRefVar[];`

- For example

`double myList[];`

This style is allowed, but not preferred

Creating arrays

```
arrayRefVar = new datatype[arraySize];
```

- For example

```
myList = new double[10];
```

- `myList[0]` references the first element in the array
- `myList[9]` references the last element in the array

Declaring and creating in one step

```
datatype[] arrayRefVar = new datatype[arraySize];
```

- For example

```
double[] myList = new double[10];
```

The length of an array

- Once an array is created, its size is fixed (i.e., it cannot be changed)
- You can find its size using `arrayRefVar.length`
- For example,

```
double[] myList = new double[10];  
myList.length
```

 returns 10

Default values

- When an array is created, its elements are assigned the default value of:
 - 0 for a numeric primitive data type
 - '\u0000' for a char type
 - false for a boolean type
 - null for a reference type

Indexed variables

- The array elements are accessed through the index
- The array indices are **0-based**
 - From 0 to `arrayRefVar.length-1`
- Each element in the array is represented using the following syntax, known as an *indexed variable*

`arrayRefVar[index];`

Using indexed variables

- After an array is created, an indexed variable can be used in the same way as a regular variable

- For example

```
myList[2] = myList[0] + myList[1];
```

Array initializers

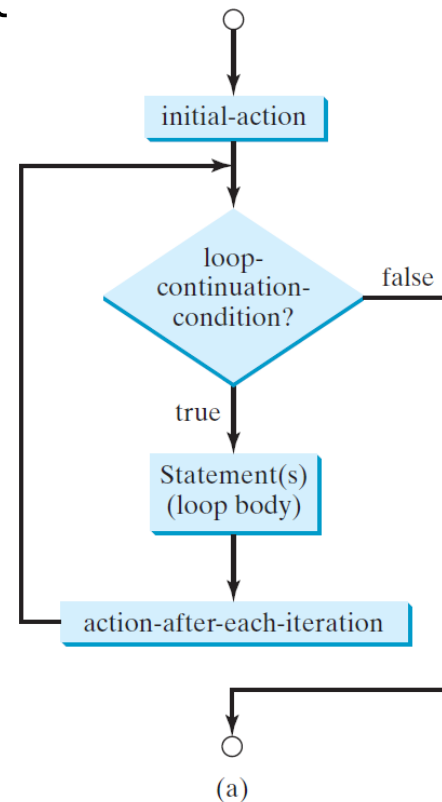
- Declaring, creating, and initializing in one step
`double[] myList = {1.9, 2.9, 3.4, 3.5};`
- This shorthand syntax must be in one statement
 - The above statement is equivalent to the following statements

```
double[] myList = new double[4];  
myList[0] = 1.9;  
myList[1] = 2.9;  
myList[2] = 3.4;  
myList[3] = 3.5;
```

for loops

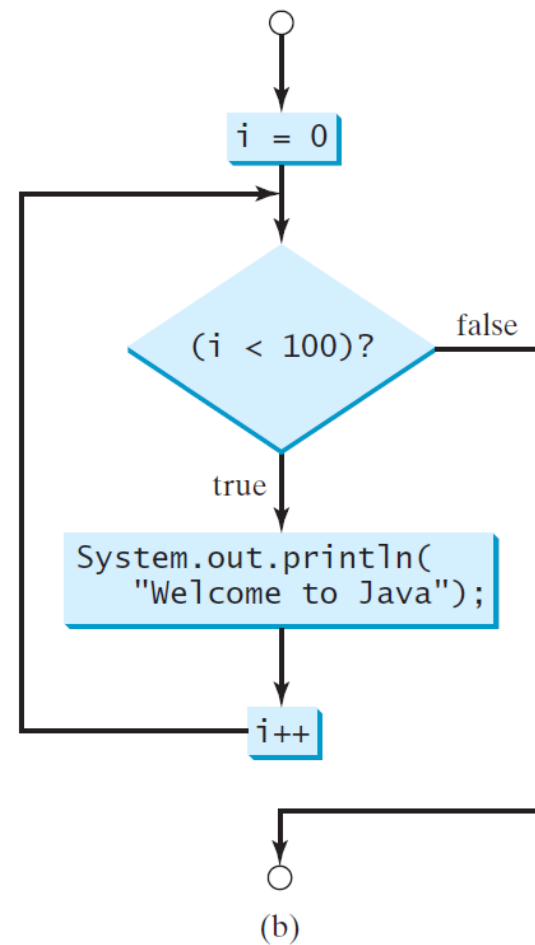
- A concise syntax for writing loops

```
for (initial-action; loop-continuation-condition;  
    action-after-each-iteration) {  
    // loop body  
    Statement(s);  
}
```



for loops

```
int i;  
for (i = 0; i < 100; i++) {  
    System.out.println(  
        "Welcome to Java!");  
}
```



for loops

- The initial-action in a for loop can be a list of zero or more comma-separated expressions
- The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements
- However, it is best practice (less error prone) **not to use comma-separated** expressions and statements

```
for (int i = 0, j = 0; (i + j < 10); i++, j++) {  
    // Do something  
}
```

Scope of local variables

- A variable declared in the initial action part of a for loop header has its scope in the entire loop
- A variable declared inside a for loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable

```
public static void method1() {  
    .  
    .  
    for (int i = 1; i < 10; i++) {  
        .  
        .  
        int j;  
        .  
        .  
        .  
    }  
}
```

The scope of i →

The scope of j →



Scope of local variables

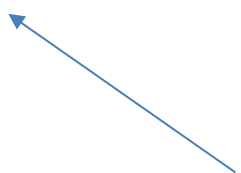
```
// Fine with no errors
public static void correctMethod() {
    int x = 1;
    int y = 1;
    // i is declared
    for (int i = 1; i < 10; i++) {
        x += i;
    }
    // i is declared again
    for (int i = 1; i < 10; i++) {
        y += i;
    }
}
```

Scope of local variables

// With errors

```
public static void incorrectMethod() {  
    int x = 1; // x is declared  
    int y = 1;  
    for (int i = 1; i < 10; i++) {  
        int x = 0;  
        x += i;  
    }  
}
```

Compile error: duplicate local variable



Processing arrays

- Summing all elements

```
double total = 0;
for (int i = 0; i < myList.length; i++) {
    total += myList[i];
}
```

- Finding the element with the maximum value

```
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) max = myList[i];
}
```

Printing arrays

```
for (int i = 0; i < myList.length; i++) {  
    System.out.print(myList[i] + " ");  
}
```

Foreach loops

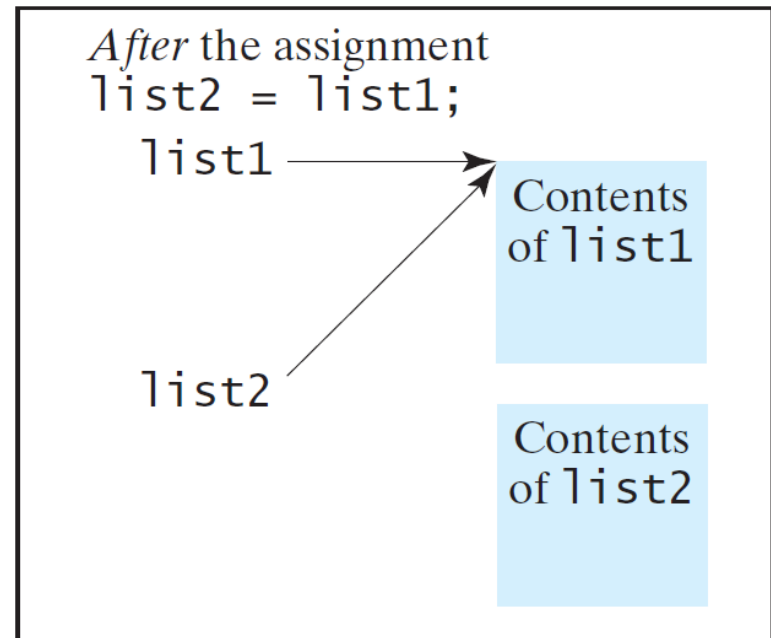
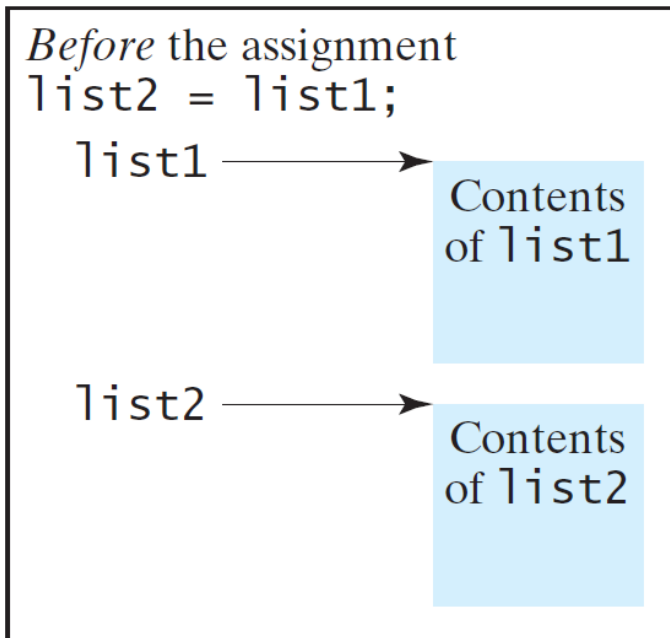
- Traverse the complete array **sequentially** without using an index variable

```
for (elementType value : arrayRefVar) {  
    // Process the value  
}
```
- For example

```
for (double value : myList)  
    System.out.println(value);
```
- You still must use an index variable if you wish to traverse the array in a different order or change the elements in the array

Copying arrays

- **The assignment statement does not copy the contents, it only copies the *reference value***
`list2 = list1;`



Copying arrays

- **To copy contents of one array to another, you must copy the array's individual elements to the other array**
- Using a loop

```
int[] srcArray = {2, 3, 1, 5, 10};  
int[] destArray = new int[srcArray.length];  
for (int i = 0; i < destArray.length; i++)  
    destArray[i] = srcArray[i];
```

Copying arrays

- Using the `System.arraycopy` method

[https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/lang/System.html#arraycopy\(java.lang.Object,int,java.lang.Object,int,int\)](https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/lang/System.html#arraycopy(java.lang.Object,int,java.lang.Object,int,int))

```
System.arraycopy(srcArray, srcPos, destArray, destPos, length);
```

— For example:

```
int[] srcArray = {2, 3, 1, 5, 10};  
int[] destArray = new int[srcArray.length];  
System.arraycopy(srcArray, 0, destArray, 0, destArray.length);
```

Passing arrays to methods

- When passing an array to a method, the **reference value** of the array is passed to the method

```
public static void printArray(int[] array) {  
    for (int i = 0; i < array.length; i++) {  
        System.out.print(array[i] + " ");  
    }  
}
```

Invoke the method, example 1:

```
int[] list = {3, 1, 2, 6, 4, 2};  
printArray(list);
```

Invoke the method, example 2:

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Anonymous array

Anonymous array

- The statement

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

creates an array using the syntax

```
new dataType[]{literal0, literal1, ..., literalk};
```

- There is no explicit reference variable for the array (i.e., it is unnamed)
- Such an array is called an *anonymous array*

Command-line parameters

```
class TestMain {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

```
java TestMain arg0 arg1 arg2 ... argn
```

- In the main method, get the arguments from `args[0]`, `args[1]`, ..., `args[n]`, which corresponds to `arg0`, `arg1`, ..., `argn` in the command line

Next Lecture

- Memory in Java