Arrays

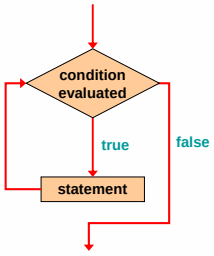
October 6, 2006

ComS 207: Programming I (in Java)
Iowa State University, FALL 2006
Instructor: Alexander Stoytchev

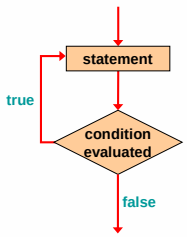
Quick review of last lecture

Comparing while and do

The while Loop



The do Loop



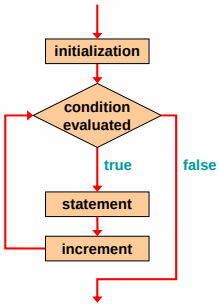
The do Statement

- **An example of a do loop:**

```
int count = 0;
do
{
   count++;
   System.out.println (count);
} while (count < 5);
```

- **The body of a do loop executes at least once**
- **See ReverseNumber.java (page 244)**

Logic of a for loop



The for Statement

- **A *for statement* has the following syntax:**

The ***initialization*** is executed once before the loop begins

The ***statement*** is executed until the ***condition*** becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

The ***increment*** portion is executed at the end of each iteration

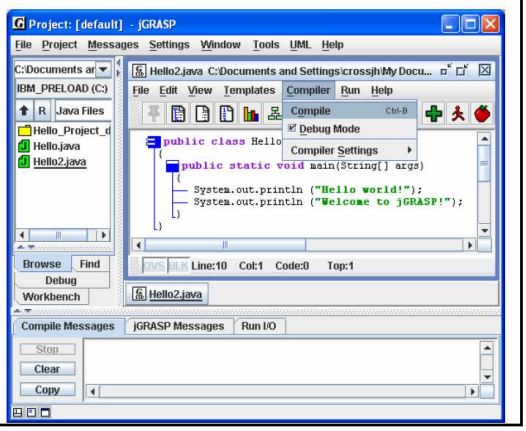
The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while ( condition )
{
   statement;
   increment;
}
```


How to use the jGRASP Debugger

The jGRASP Tutorials



Other Things About Loops

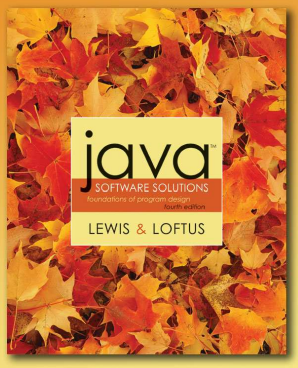
- 'break' Statement
- 'continue' Statement
- Empty Statement - ';'

The for Statement

- Each expression in the header of a `for` loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed

Chapter 7

Arrays



PEARSON Addison Wesley

Arrays

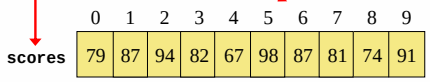
- Arrays are objects that help us organize large amounts of information

Arrays

- An *array* is an ordered list of values

The entire array has a single name

Each value has a numeric *index*



An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Arrays

- A particular value in an array is referenced using the array name followed by the index in brackets
- For example, the expression

```
scores[2]
```

refers to the value 94 (the 3rd value in the array)

- That expression represents a place to store a single integer and can be used wherever an integer variable can be used

Arrays

- For example, an array element can be assigned a value, printed, or used in a calculation:

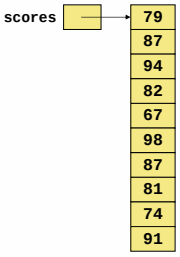
```
scores[2] = 89;

scores[first] = scores[first] + 2;

mean = (scores[0] + scores[1])/2;

System.out.println ("Top = " + scores[5]);
```


Arrays

- The values held in an array are called *array elements*
- An array stores multiple values of the same type – the *element type*
- The element type can be a primitive type or an object reference
- Therefore, we can create an array of integers, an array of characters, an array of `String` objects, an array of `Coin` objects, etc.
- In Java, the array itself is an object that must be instantiated

Arrays

- Another way to depict the `scores` array:



Declaring Arrays

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

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

- The type of the variable `scores` is `int[]` (an array of integers)
- 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];
```


Example: `BasicArray.java` (page 372)

Example: `ReverseOrder.java` (page 375)

Using Arrays

- The iterator version of the `for` loop can be used when processing array elements

```
for (int score : scores)
   System.out.println (score);
```

- This is only appropriate when processing all array elements from top (lowest index) to bottom (highest index)
- See `BasicArray.java` (page 372)

What for/in can't do

```
int[] primeNums = {2, 3, 5, 7, 11, 13, 17, 19};

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

What for/in can't do

```
String word="Test";

for (int i=0; i< word.length(); i++)
{
   if(i>0)
     System.out.print(“, “);

   System.out.print( word.charAt(i) );
}
```


Other Stuff From Chapter 5

Iterators

- An *iterator* is an object that allows you to process a collection of items one at a time
- It lets you step through each item in turn and process it as needed
- An iterator object has a `hasNext` method that returns true if there is at least one more item to process
- The `next` method returns the next item
- Iterator objects are defined using the `Iterator` interface, which is discussed further in Chapter 6

Iterators

- Several classes in the Java standard class library are iterators
- The `Scanner` class is an iterator
  - the `hasNext` method returns true if there is more data to be scanned
  - the `next` method returns the next scanned token as a string
- The `Scanner` class also has variations on the `hasNext` method for specific data types (such as `hasNextInt`)

Iterators

- The fact that a `Scanner` is an iterator is particularly helpful when reading input from a file
- Suppose we wanted to read and process a list of URLs stored in a file
- One scanner can be set up to read each line of the input until the end of the file is encountered
- Another scanner can be set up for each URL to process each part of the path
- See URLDissector.java (page 240)

Example: URLDissector.java (page 240)

Iterators and for Loops

- Recall that an iterator is an object that allows you to process each item in a collection
- A variant of the `for` loop simplifies the repetitive processing the items
- For example, if `BookList` is an iterator that manages `Book` objects, the following loop will print each book:

```
for (Book myBook : BookList)
   System.out.println (myBook);
```


Iterators and for Loops

- This style of `for` loop can be read "for each `Book` in `BookList`, ..."
- Therefore the iterator version of the `for` loop is sometimes referred to as the *foreach* loop
- It eliminates the need to call the `hasNext` and `next` methods explicitly
- It also will be helpful when processing arrays, which are discussed in Chapter 7

THE END