CPS109 Course Notes 6

Alexander Ferworn

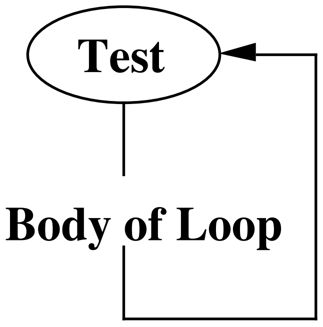
Unrelated Facts Worth Remembering

- Use metaphors to understand issues and explain them to others.
- Look up what metaphor means.

Table of Contents

1 Iteration

Previously we have seen how to make a decision in java using the if or the switch. Often we want to make a decision about doing a task many times. In effect you hit the decision again and again and again. This repetition is called iteration or looping, where we decide to do a repetitive task based on a decision made before the task begins, or after the task has been done at least once. This is the topic of this document. The most basic loop appears as the diagram below. A test is performed before the body of a loop is executed. If the test evaluates to true then the body of the loop is executed. At the end of the body the loop returns control back to the test which is reevaluated for truth. This continues until the test turns false and the loop is exited.



2 while

A while loop is used to repeat a given statement over and over. Of course, its not likely that you would want to repeat forever. That would be an infinite loop, which is generally a bad thing. (There is an old story about computer pioneer Grace Murray Hopper, who read instructions on a bottle of shampoo telling her to " lather, rinse, repeat." As the story goes, she claims that she tried to follow the directions but that she ran out of shampoo. (In case you don't get it, this is a joke about the way that computers mindlessly follow instructions.))

Definition:

- A while loop will repeat a statement over and over, but only so long as a specified condition remains true.
- A while loop can repeat zero or more times.

Syntax:

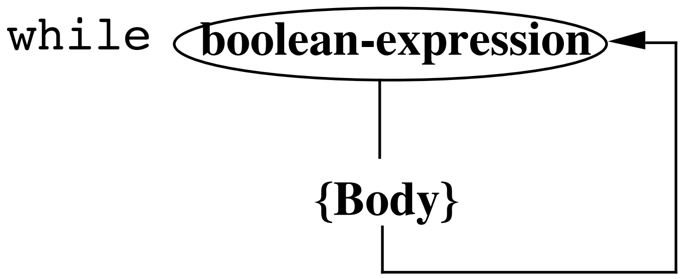
A while loop has the form:

```
     while (<boolean-expression>)
          <statement>
```

Since the statement can be, and usually is, a block, many while loops have the form:

```
     while (<boolean-expression>)
     {
          <statements>
     }
```

When the computer comes to a while statement, it evaluates the boolean-expression, which yields either true or false as the value. If the value is false, the computer skips over the rest of the while loop and proceeds to the next command in the program. If the value of the expression is true, the computer executes the statements inside the loop. Then it returns to the beginning of the while loop and repeats the process. That is, it re-evaluates the boolean-expression, ends the loop if the value is false and continues it if the value is true. This will continue over and over until the value of the expression is false; if that never happens, then there will be an infinite loop.



Here is an example of a while loop that simply prints out the numbers 1, 2, 3, 4, 5:

```
     int number = 1;
     while ( number < 5 )
     {
          System.out.println(number);
          number = number + 1;
```

```
    }
    System.out.println("Done!");
```

The variable number is initialized with the value 1. So the first time through the while loop, when the computer evaluates the expression "number < 5", it is asking whether 1 is less than 5, which is true. The computer therefore proceeds to execute the two statements inside the loop. The first statement prints out "1". The second statement adds 1 to number and stores the result back into the variable number; the value of number has been changed to 2. The computer has reached the end of the loop, so it returns to the beginning and asks again whether number is less than 5. Once again this is true, so the computer executes the loop again, this time printing out 2 as the value of number and then changing the value of number to 3. It continues in this way until eventually number becomes equal to 5. At that point, the expression "number < 5" evaluates to false. So, the computer jumps past the end of the loop to the next statement and prints out the message "Done!"

By the way, you should remember that you'll never see a while loop standing by itself in a real program. It will always be inside a subroutine which is itself defined inside some class. As an example of a while loop used inside a complete program, here is a little program that computes the interest on an investment over several years:

```
public class Interest2
{
      /*
      This class implements a simple program that
      will compute the amount of interest that is
      earned on $17,000 invested at an interest
      rate of 0.07 for 5 years.  The value of
      the investment at the end of each year
      is printed to standard output.
      */
      public static void main(String[] args)
      {
            double principal = 17000;
            // the initial value of the investment
            double rate = 0.07;
            // the annual interest rate

            int years = 0;
            // counts the number of years that
            // have passed

            while (years < 5)
            {
                  double interest = principal * rate;
                  // compute this year's interest
                  principal = principal + interest;
                  // add it to principal
                  years = years + 1;
```

```
                    // count the current year.
                    System.out.print
                    ("The value of the invmnt after ");
                    System.out.print(years);
                    System.out.print(" years is $");
                    System.out.println(principal);
               } // end of while loop
          } // end of main()
     } // end of class Interest2
```

3 do-while

A previous section introduced the while statement, in which the computer tests a condition at the beginning of the loop, to determine whether it should continue looping:

The do loop is a variation of this in which the test comes at the end. It takes the form:

```
do
     <statement>
while ( <boolean-expression> );
```

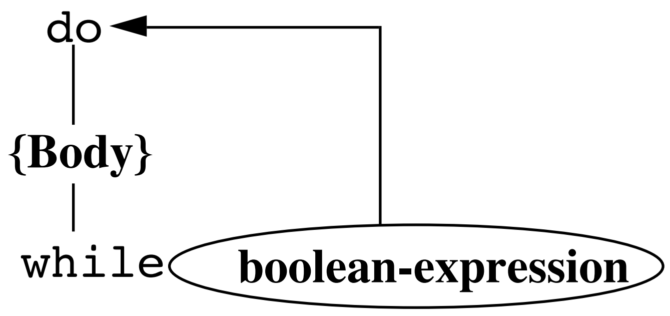
or, since as usual the statement can be a block,

```
do
{
     <statements>
} while ( <boolean-expression> );
```

(Note the semicolon, ';', at the end. This semicolon is part of the statement, just as the semicolon at the end of an assignment statement or declaration is part of the statement. More generally, every statement in Java ends either with a semicolon or a right brace, '}'.)

Note:

- In a Do loop, the body of the loop is executed 1 or more times (at least once).



To execute a do loop, the computer first executes the body of the loop -- that is, the statement or statements inside the loop -- and then evaluates the boolean expression. If the value of the expression is true, the computer returns to the beginning of the do loop and repeats the process; if the value is false, it ends the loop and continues with the next part of the program.

The main difference between the do loop and the while loop is that the body of a do loop is executed at least once, before the boolean expression is ever evaluated. In a while loop, if the boolean expression is false when the computer first checks it, then the body of the loop will never be executed at all.

For example, consider the following pseudo code for a game-playing program. The do loop makes sense here instead of a while loop because with the do loop, you know there will be at least one game. Also, the test that is used at the end of the loop wouldn't even make sense at the beginning:

```
do
{
      <Play a Game>
      <Ask user if he wants to play another game>
} while ( <the user's response is yes> );
```

4 for

The for loop exists to make a common type of while loop easier to write. Many while loops have the same general form:

```
<initialization>
while (<continuation-condition> )
{
      <statements>
      <update>
}
```

For example, consider

```
int years = 0;  // initialize the variable years
while ( years < 5 )
{
      // condition for continuing loop
      interest = principal * rate;
      // do some statements
      principal += interest;
      System.out.println(principal);
      years++;
      // update the value of the variable years
}
```

This loop can be written as the following for statement:

```
for ( int years = 0; years < 5; years++ )
{
      interest = principal * rate;
      principal += interest;
      System.out.println(principal);
}
```

The initialization, continuation condition, and updating have all been combined in the first line of the for loop. This keeps everything involved in the "control" of the loop in one place, which helps makes the loop easier to read and understand. In general, a for loop takes the form:

```
for(<initialization>;<continuation-condition>;<update>)
      <statement>
```

or, using a block statement,:

```
for(<initialization>;<continuation-condition>;<update>)
{
      <statement>
}
```

The continuation-condition must be a boolean-valued expression. The initialization can be any expression, as can the update. Any of the three can be empty. Usually, the initialization is an assignment or a declaration, and the update is an assignment or an increment or decrement operation. The official syntax actually allows the initialization and the update to consist of several expressions, separated by commas. If you declare a variable in the initialization section, the scope of that variable is limited to the for statement; that is, it is no longer valid after the for statement is ended.

Here are a few examples of for loops:

```
// print out the alphabet on one line of
output
for ( char ch='A'; ch <= 'Z';  ch++ )
      System.out.print(ch);
System.out.println();

// count up to 10 and down from 10 at the same time
for ( int i=0,j=10;  i < 10;  i++,j-- )
{
      System.out.println(i + " " + j);
}

// compute the number 1 * 2 * 3 * ... * 20
```

```
      long factorial = 1;
      for (int num=2; num <= 20; num++)
            factorial *= num;
      System.out.println("20! = " + factorial);
```

5 A Bit More About Break

The syntax of the while, do, and for loops allows you to make a test at either the beginning or at the end of the loop to determine whether or not to continue executing the loop. Sometimes, it is more natural to have the test in the middle of the loop, or to have several tests at different places in the same loop. Java provides a general method for breaking out of the middle of any loop. It's called the break statement, which takes the form

```
      break;
```

When the computer executes a break statement, it will immediately jump out of the loop (or other control structure) that contains the break. It then continues on to whatever follows the loop in the program. Consider for example:

```
      while (true)
      {  // looks like it will run forever!
            console.put("Enter a positive number: ");
            N = console.getlnt();
            if (N > 0)   // input is OK; jump out of loop
                  break;
            console.putln("Your answer must be > 0.");
      }
      // continue here after break
```

A break statement terminates the loop that immediately encloses the break statement. It is possible to have nested loops, where one loop statement is contained inside another. If you use a break statement inside a nested loop, it will only break out of that loop, not out of the loop that contains the nested loop. There is something called a "labeled break" statement that allows you to specify which loop you want to break. I won't give the details here; you can look them up if you ever need them.