CS 106
Introduction to Computer Science I

09 / 30 / 2015

Instructor: Michael Eckmann
Today’s Topics

• Comments and/or Questions?
• while loops
• for loops
while loops

• While loops are used to repeat lines of code until some condition is met.

e.g.

```java
while (condition) {
    // statements to do while condition is true.
}
```

• condition is evaluated and if true the code within the curly braces executes. Then condition is tested again and if true the code within the curly braces executes. This continues until the condition is false (or a break; statement is hit.)
while loops

- It's extremely important for you to know exactly what code is executed in what order for you to be able to follow what a program is doing. Let's follow the code below line by line in order. (assume that it is inside a main method of a class.)

```java
int x = 1;
System.out.println("Just before the loop");
while (x < 4)
{
    System.out.print("*");
    x++;
}
System.out.println("\nJust got out of the loop");
```
while loops

```java
int total = 0;
while (total < 100)
{
    total = total + 10;
    System.out.println("total = " + total);
}
System.out.println("finished printing totals");
```

// What would this code do?
while loops

• Sentinel controlled loops vs. counter controlled loops.
• Counter controlled loops are those loops that usually execute a set number of times every time.
• A sentinel is a value that is used to terminate a loop. For instance, if you wanted a user to enter in some numbers (but you didn't know how many s/he was going to enter) to be averaged (and assuming that they are all supposed to be positive) we can ask the user to enter in -1 after s/he enters all the numbers so we know s/he's finished.
• Let's see an example.
while loops

int input_num=0, count = 0;
double total = 0.0;
while (input_num != -1)
{
    input_str = JOptionPane(“Enter a number to be averaged”);
    input_num = Integer.parseInt(input_str);
    total += input_num;
    count++;
}
System.out.println(“average = “ + (total / count));
// any problems with this code?????  Hint: yes there's a logic error.
while loops

- Sentinel controlled loops vs. counter controlled loops.
- Counter controlled loops are those loops that usually execute a set number of times every time.
- As a programmer what value makes sense to choose as the sentinel is specific to the situation.
- Could anyone characterize what would be a bad value to choose as the sentinel?
while loops

• Order of execution of a while loop.
• First the condition is tested.
  – If it is true the statements within the curly braces execute
  – Then condition is tested again
  – If it is true the statements within the curly braces execute again
  – Then condition is tested again
  – ...
  – This continues until the condition is false when tested. At that time, program control continues with the code after the right curly brace.
• Note: It IS possible that even on the first test of the condition, that it may be false. If it is, the statements within the loop don't execute at all.
while loops

• What happens to loops that do not alter the value of a variable in the condition?
while loops

• What happens to loops that do not alter the value of variable(s) in the condition?

• If that condition is true the first time, it will stay true forever and we will have an infinite loop.

• e.g.

```java
int x = 2;
while (x > 0)
{
    System.out.println("x = " + x);
}
```
while loops

• Any other ways to get an infinite loop?
while loops

• Would this be infinite?

```java
int x = 2;
while (x > 0)
{
    System.out.println("x = " + x);
    x++;
}
```
for loops

- The *for* loop is a counter controlled repetition structure that compactly lets a programmer specify the control variable, its initial value, the condition to be checked each time through the loop and the amount to increment (or decrement) the control variable

```
for (expression1; expression2; expression3)  
    statement or compound statement
```
for loops

for (expression1; expression2; expression3)

statement or compound statement

where expression1 contains the declaration and initialization of the control variable

expression2 contains the continuation condition and the ending value of the control variable

expression3 contains the increment or decrement of the control variable (and the amount of this change in value each time through the loop)
for loops

for ( expression1; expression2; expression3 )
{
    statements
}

Order of operations:
1. expression1 executes first (and only once)
2. expression2 executes and if the result is false, program control goes to the code after the right curly brace
3. if the result is true, then the statements within the curly braces of the for loop execute in order then expression3 executes (which usually alters the value of the loop variable.) Then “goto” 2 above.
for loop example

for ( int x = 10; x >= 1; x-- )
{
    do some statements here ....
}

• note that we used x-- to subtract 1 from the counter each time through the loop, we could have used:
  
x = x - 1  or  x -= 1  or  --x
for loop

• Each of the three expressions in the for loop structure are optional.
• don’t need expression1 if control variable declared and initialized before the loop
• don’t need expression2 if you desire having an infinite loop
• don’t need expression3 if you change the value of the control variable within the loop
for loop

for ( int cntr = 1; cntr <= 20; cntr = cntr + 1 )
{
    do some statements here ....
}

• this is essentially the same thing as the following while loop

    int cntr = 1;
    while ( cntr <= 20 )
    {
        do some statements here ....
        cntr = cntr + 1;
    }

• except, the variable cntr is not available to the program after the for loop, but it is available to be referenced after the while loop

Michael Eckmann  -  Skidmore
College - CS 106 - Fall 2015
exercise

• Write an application that calculates the product of the odd integers from 1 to 15 and then displays the results in a message dialog.

• Let's write it using a while loop
• Then let's write it using a for loop
do while loops

• While loops and for loops all have the possibility of having the statements inside them execute zero or more times. Zero being the key word here.

• When would it be possible for those loops to execute zero times?

• The big difference between those kinds of loops and do-while loops are that do-while loops execute their statements at least once (that is, one or more times.)
do while loops

do
{
    statements to do
}
} while (condition);

– In this kind of loop, the condition is tested after the loop executes the first time. If the condition is true it does the statements again, and so on until the condition is false.
do while loops

do
{
    statements to do 1, 3 (if 2 was true)
}

} while (condition);

– In this kind of loop, the condition is tested **after** the loop executes the first time. If the condition is true it does the statements again, and so on until the condition is false.
do while loops

```c
int cntr = 1;
while ( cntr <= 20 )
{
    
do some statements here ....
    
    cntr = cntr + 1;
}
// the above is basically equivalent to:
int cntr = 1;
do
{
    
do some statements here ...
    
    cntr = cntr + 1;
} while (cntr <= 20);
```
break;

• We've seen how break; reacts in switch statements.

• break; acts similarly within the curly braces of a while loop, for loop or do-while loop.

• It terminates the loop and program control continues after the loop as if it ended normally.
break;

int x = 0;
while (x <= 10)
{
    if (x == 5)
    {
        break;
        break;
        System.out.println(“x = “ + x);
        x++;
    }
}

// what's this code going to do?