Today’s Topics

- Comments and/or Questions?
- Finish up while loops
- for loops
- do-while loops
- break; statement
- switch statements
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
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
do statements to do
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
{
    \textit{statements to do} \ 1, 3 \ (if 2 was true)
}
} while (condition);

\begin{itemize}
    \item In this kind of loop, the condition is tested \textbf{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.
\end{itemize}
do while loops

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

// the above is basically equivalent to:
```c
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.
```java
int x = 0;
while (x <= 10) {
    if (x == 5) {
        break;
    }
    System.out.println("x = "+ x);
    x++;
}

// what's this code going to do?
```
Switch

switch (some_var)
{
    case 1:
        // do stuff if some_var's value is 1
        break;
    case 2:
        // do stuff if some_var's value is 2
        break;
    default:
        // do stuff if none of the cases executed.
}
break; // this statement exits out of the switch in which it is enclosed

and program continues after the } that ends the switch

let's write some code showing an example use of switch to emulate a menu like,

1. print *'s
2. print $'s
3. print @'s
4. print !'s
Switch

- Important comments about switch.
- The value in the switch is compared in order to the cases. If it is unequal to a case, it gets compared to the case below it and so on. When/if it gets to an equal case, then the code within that case is executed and if there is no break, the code in lower cases continues to be executed until a break statement or the }.
- BE CAREFUL about putting break; statements in. Unless you're trying to do something fancy, they should be used at the end of each case.
- If the value in the switch doesn't equal any of the cases, the optional default case executes.