Lecture 12:

(Wrap up) Operator Precedence and Associativity

Begin Loops

Feb 10 2015
Operator Precedence and Associativity

Operator precedence and associativity determine the order in which operators are evaluated.

**High**
- `var++`, `var--`
- `+`, `-` (Unary plus and minus), `++var`, `--var`
- `(type)` Casting
- `!` (Not)
- `*`, `/`, `%` (Multiplication, division, and remainder)
- `+`, `-` (Binary addition and subtraction)
- `<`, `<=`, `>`, `>=` (Comparison)
- `==`, `!=` (Equality)
- `^` (Exclusive OR)
- `&&` (Conditional AND) Short-circuit AND
- `||` (Conditional OR) Short-circuit OR

**Low**
- `=`, `+=`, `-=`, `*=`, `/=`, `%=` (Assignment operator)

All binary operators are left-associative. (except assignment operators)
Loops: Motivation

Suppose that you need to print a string (e.g., "Welcome to Java!") one hundred times. It would be tedious to have to write the following statement one hundred times:

System.out.println("Welcome to Java!");

So, how do you solve this problem?
while Loops

```java
int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java");
    count++;
}
```
**while Loop Flow Chart**

**Syntax:**
```java
while (loop-continuation-condition) {
  // loop-body;
  Statement(s);
}
```

**Example:**
```java
int count = 0;
while (count < 100) {
  System.out.println("Welcome to Java");
  count++;
}
```

**Flow Chart:**

(A) Loop Continuation Condition?  
true \rightarrow Statement(s) (loop body)  
false

(B) (count < 100)?  
true \rightarrow System.out.println("Welcome to Java"); count++  
false


Guessing Numbers

Write a program that randomly generates an integer between 0 and 100, inclusive.

The program prompts the user to enter a number continuously until the number matches the randomly generated number.

For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently.
Caution: infinite loop

Do not use floating-point values for equality checking in a loop control. Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results.

Consider the following code for computing

\[ 1 + 0.9 + 0.8 + \ldots + 0.1: \]

```java
double item = 1; double sum = 0;
while (item != 0) { // No guarantee item will ever be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);
```
Caution: infinite loop

Make sure your condition terminates at some point, otherwise you have an infinite loop. Infinite loops are causing the program to run forever, so they are **big mistakes**

```java
int item = 1; int sum = 0;
while (item != 0) { // No guarantee item will ever be 0
    sum += item; // this is an infinite loop (will never terminate)
    item += 1;
}
System.out.println(sum); // nothing will ever print to the screen
```

```java
int item = 1; int sum = 0;
while (item != 10) { // not an infinite loop, condition terminates
    sum += item;
    item += 1;
}
System.out.println(sum); // output is 45
```