CSCI 136
Data Structures & Advanced Programming

Fall 2018

Instructors

Bill Lenhart & Bill Jannen

Administrative Details

- Lab 1 handout is online
- Prelab (should be completed before lab):
  - Lab 1 design doc
    - Use Die Design Doc as model - no pseudo-code needed this time!
- TA hours start tonight
  - See TA hour schedule on course website

Last Time

Basic Java elements so far

- Primitive and array types
- Variable declaration and assignment

Some basic unix commands

- Compile (javac), run (java) cycle

Today

- Further examples
- Discussion: Lab 1
- Operators & operator precedence
- Expressions
- Control structures
  - Branching: if – else, switch, break, continue
  - Looping: while, do – while, for, for – each
- Object-Oriented Program (OOP) Design
  - Basic concepts and Java-specific features

Sample Programs

- Sum0-5.java
  - Programs that adds two integers
- Of Note:
  - System.in is of type ReadStream
  - Scanner class provides parsing of text streams (terminal input, files, Strings, etc)
  - args[] is passed to `main` from the OS environment
    - args[] contains command-line arguments held as Strings
  - Integer.valueOf(...) converts String to int
  - Static values/methods: in, out, valueOf, main

Lab 1

- Purpose
- Coinstrip Game
- Demo of solution
- Die Design Doc

Operators

Java provides a number of built-in *operators* including

- Arithmetic operators: +, -, *, /, %
- Relational operators: ==, !=, <, **≤, >, ≥**
- Logical operators &&, || (don't use &, |)
- Assignment operators =, +=, -=, *=, /=, ...

Common unary operators include

- Arithmetic: - (prefix); ++, -- (prefix and postfix)
- Logical: ! (not)

Operator Precedence in Java

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Operator Gotchas!

- There is no exponentiation operator in Java.
  - The symbol ^ is the *bitwise or* operator in Java.
- The *remainder* operator % is the same as the mathematical 'mod' function for *positive* arguments,
  - For **negative** arguments **it is not**: -8 % 3 = -2
- The logical operators && and || use *short-circuit evaluation*:
  - Once the value of the logical expression can be determined, no further evaluation takes place.
  - E.g.: If n = 0, then (n != 0) && (k/n > 3), will yield false without evaluating k/n. Very useful!

Expressions

Expressions are either:

- literals, variables, invocations of non-void methods, or
- statements formed by applying operators to them

An expression returns a value

- `3+2*5 - 7/4      // returns 12`
- `x + y*z – q/w`
- `(- b + Math.sqrt(b*b – 4 * a * c) )/( 2* a)`
- `( n > 0) && (k / n > 2) // computes a` *`boolean`*

Expressions

Assignment operator also forms an expression

- x = 3; // assigns x the value 3 and returns 3
- So y = 4 * (x = 3) sets x = 3 and y = 12 (and returns 12)

Boolean expressions let us control program *flow of execution* when combined with *control structures*

Example

- `if ( (x < 5) && (y !=0 ) ) {...}`
- `while (! loggedIn) { ... }`

Control Structures

Select next statement to execute based on value of a boolean expression. Two flavors:

- Looping structures: `while, do/while, for`
  - Repeatedly execute same statement (block)
- Branching structures: `if, if/else, switch`
  - Select one of several possible statements (blocks)
  - Special: break/continue: exit a looping structure
    - break: exits loop completely
    - continue: proceeds to next iteration of loop

while & do-while

Consider this code to flip coin until heads up...

```
Random rng = new Random();
int flip = rng.nextInt(2), count = 0;
while (flip == 0) {   // count flips until "heads"
    count++;
    flip = rng.nextInt(2);
}
```

...and compare it to this

```
int flip, count = 0;
do {                  // count flips until "heads"
    count++;
    flip = rng.nextInt(2);
} while (flip == 0) ;
```

For & for-each

Here's a typical **for** loop example

```
int[] grades = { 100, 78, 92, 87, 89, 90 };
int sum = 0;
for( int i = 0; i < grades.length; i++ )
    sum += grades[i];
```

This **for** construct is equivalent to

```
int i = 0;
while ( i < grades.length ) {
    sum += grades[i];
    i++;
}
```

Can also write

```
for (int g : grades ) sum += g;
// called for-each construct
```

Loop Construct Notes

- The body of a **while** loop may not ever be executed
- The body of a **do – while** loop always executes at least once
- **For** loops are typically used when number of iterations desired is known in advance. E.g.
  - Execute loop exactly 100 times
  - Execute loop for each element of an array
- The **for-each** construct is often used to access array (and other collection type) values when *no updating* of the array is required
  - We'll explore this construct more later in the course