Java Basics

Session 3

Overall Schedule

- Session 1: Java installation, key programming concepts, overview of Java, structure of a Java program
- Session 2: Eclipse IDE,Variables, data types, mathematical operations, Strings
- Session 3: Decisions and control flow
- Session 4: Arrays and methods

Overall Schedule

- Session 5: Classes, objects, and inheritance
- Session 6: Interfaces, the List and Comparable Interfaces

Session 3

Session Topics

- Comparison and Logical Operators
- Scope
- Decisions
  - if-else statements
- Loops
  - while
  - for
  - do-while
- The switch statement

Comparison and Logical Operators

C&L Operators

- All of these operators evaluate to booleans
- Comparison operators work on all numeric/char types in any mix. Promotion occurs as in mathematical operators
- Equality operators are legal on *all* variables of compatible types
- Logical operators only work on booleans
- Bitwise operators are a trick, and are only appropriate on booleans in this context
- Precedence is between Mathematical and Assignment operators, except ! which is prefix unary

C&L Operators

Operators	Function
!	Logical not. Flips true and false
< > <= >=	Comparison. Less Than, Greater Than, etc
== !=	Equals and Not Equals
&	Bitwise And
^	Bitwise Exclusive Or (NOT mathematical power)
|	Bitwise Inclusive Or
&&	Logical And
||	Logical Or

Logical Operators

true && true	true
true && false	false
false && true	false
false && false	false

true & true	true
true & false	false
false & true	false
false & false	false

true ^ true	false
true ^ false	true
false ^ true	true
false ^ false	false

true || true	true
true || false	true
false || true	true
false || false	false

true | true	true
true | false	true
false | true	true
false | false	false

!true	false
!false	true

Logical vs Bitwise

- Logical operators in Java *short circuit*
  - If the left side of an && is false, the right is ignored
  - If the left side of an || is true, the right is ignored
- Bitwise operators fully evaluate
  - a && (b = x > y) vs
  - a & (b = x > y)
  - with &&, b is only assigned x > y if a is true
  - with &, be is assigned x > y in either case.
- Bitwise ^
  - a ^ b vs
  - a || b && !(a && b)

Scope

Scope

- Scope refers to where in your code a particular name is visible.
  - It is solely a feature of your code and does not necessarily indicate when memory is or is not allocated
  - General Rule (so far): A variable is in scope from the line on which it is declared until the end of the code block it's declared in.
  - There are other rules and cases we'll look at as they come up.

Decisions

If - else

Basic structure:

- *italics* indicate a placeholder for code
- []s indicate an optional part

if(*condition*)
    statement
[else
    statement]

Important note

- A *statement* can be
  - A single semi-colon terminated statement
  - Another if statement (and others we'll see later)
  - A block of multiple statements in {}s

Example

```
if (score >= 90)
        grade = 'A';
else
        if (score >= 80)
                grade = 'B';
        else
                if (score >= 70)
                        grade = 'C';
                else
                        if (score >= 60)
                                grade = 'D';
                        else
                                grade = 'F';
```

Man, that was cumbersome

Good thing white space isn't meaningful!

```
if (score >= 90)
        grade = 'A';
else if (score >= 80)
        grade = 'B';
else if (score >= 70)
        grade = 'C';
else if (score >= 60)
        grade = 'D';
else
        grade = 'F';
```

With braces (optional in this case)

```
if (score >= 90) {
        grade = 'A';
}
else if (score >= 80) {
        grade = 'B';
}
else if (score >= 70) {
        grade = 'C';
}
else if (score >= 60) {
        grade = 'D';
{
else {
        grade = 'F';
}
```

Or even more compact

```
if (score >= 90) grade = 'A';
else if (score >= 80) grade = 'B';
else if (score >= 70) grade = 'C';
else if (score >= 60) grade = 'D';
else grade = 'F';
```

Note on layout and whitespace

- Only a few rules
  - Be consistent with your team/project/product.
  - Avoid the need for scrolling right on a normal screen.
  - { can go at the end of the line starting the block or on the next line. Be consistent. People have strong beliefs on this.
  - } can ONLY go on the next line.
  - if else if is the only place to not indent nested blocks.

Let's practice this… um… stuff!

- https://codingbat.com/java
  - From Warmup-1
    - sleepIn
    - hasTeen
    - loneTeen
  - From Logic-1
    - cigarParty

Loops

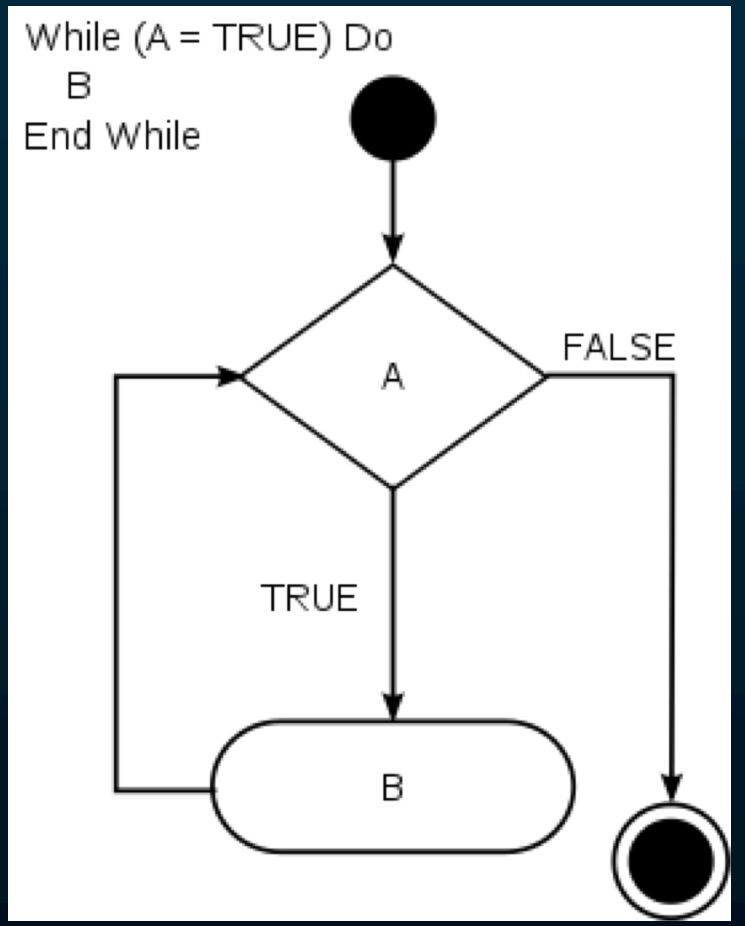
Loops

- Loops allow for the repetition of code.
  - Work essentially like a repeating if statement, checking the same condition repeatedly
  - Three types
    - while
    - for
    - do-while

The while loop

while(*condition*)
    statement

- The while loop is a *pretest* loop. If the condition is initially false, the code block does not execute.



Example

```
int num = 1;
while (num <= 10) {
    System.out.println(num + " x 2 = " + num * 2);
    num = num + 1;
}
```

For Loop

for([*initialization*]; [*condition*]; [*update*])
statement

- initialization allows us to set or create values prior to the run of the loop.
  - initialization only happens once. Values are not reset between executions of the loop body.
  - A variable declared in the initialization is only in scope for the loop.
  - The initialization can contain multiple expressions separated by commas.

Condition and Update

- The condition is the same as in an if or while
  - The for loop is a pretest loop as the while loop is
- The update is an expression or comma-separated list of expressions that occur *between* runs of the loop, before the condition is checked
  - The update does NOT occur before he first check of the condition

Components are optional

- All three components of a for loop are optional
  - Naturally, you may not need to initialize any values
  - Likewise, any needed changes to values may occur in the body, as with a while loop
  - If the condition is omitted, it is the same is putting *true* for the condition.
  - The semicolons stay, so a for loop may look like:
  - for(; x<10 ;) this is identical to while(x<10)
  - for(;;)

Same Example, with a for

```
for(int num = 1; num <= 10; num++) {
    System.out.println(num + " x 2 = " + num * 2);
}
```

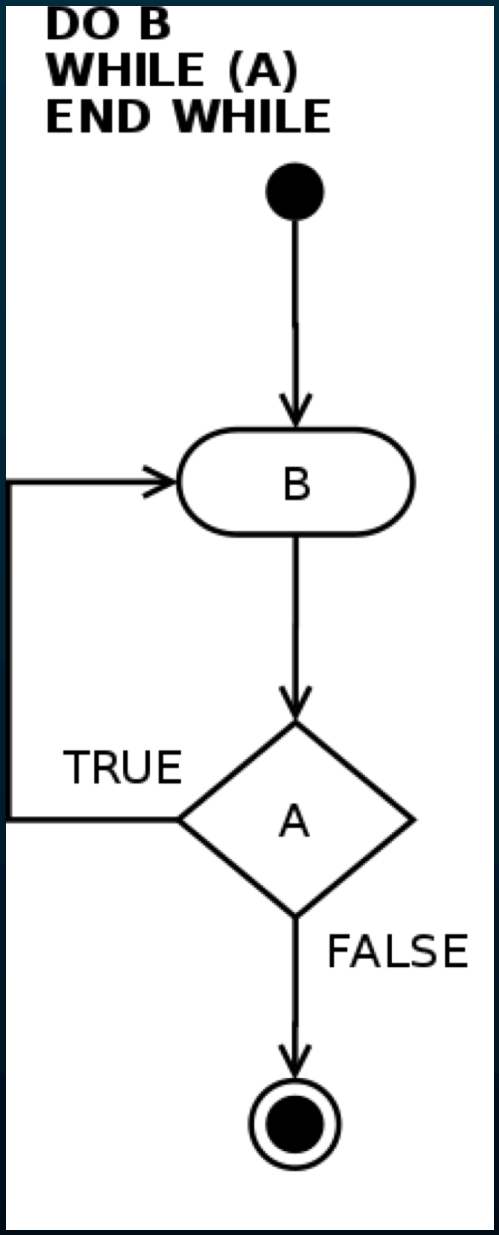
Most for loops look something similar to this.

(As with the if statement, the {}s here are unnecessary as there is only one statement in the body.)

The do-while loop

```
do {
    statement
}while(condition);
```

- This is a *posttest* loop
  - It is identical to a while loop except that the body happens once before the condition is true.
  - No one really knows why the semi-colon is there. It's required.



break and continue

- There are two keywords that allow us to alter the normal flow of a loop.
  - break; exits the loop immediately, going to the next line after the loop.
    - Thus, we can get out of while(true) or for(;;)
  - continue jumps directly to the condition check, skipping the rest of the body
    - In a for loop, the update *is* evaluated prior to the condition check after a continue

MOAR practice!

- Write a loop to add up the values from 1 to 10 (inclusive)
- Write a loop to add up all of the *even* values from 1 to 10 (inclusive)

In both cases, if you have extra time, do so with more than one loop type.

Quick look – the switch statement

switch (*int, char, or String expression*) {
    case *literal or final**:
        statements
        [break;]
    case *literal or final**:
        statements
        [break;]
    [default:]
        statements
}

Let's quick look at https://www.w3schools.com/java/java_switch.asp

Heads Up – Input!

- We will bring in console input on Monday.
- This will use the Scanner class
- We'll only look at simple input that avoids getting into ugly stuff with type conversion and input buffer issues.