if/else & loop basics

BOOLEAN HAIR LOGIC

A  B

AND  OR  XOR

if (hunger > 0)
  FeedMe();
else
  PlayWithMe();
end
Announcements

HW1 posted, due Friday.
Highlights

- if/else statements

```java
if(booleanExpression){
    // im branching!
}
```

- loop

```java
while(x < 10)
{
    // do something!
}
```
if statement

Code inside an **if statement** is only run if the condition is true.

```java
if (isTrue)
    System.out.println("This is printed if variable isTrue is true.");
```

Need parenthesis (no semi-colon)

Indent
if statement

Braces allow multiple lines to be run. (See: BasicIf.java)

```java
if (true)
{
    System.out.println("This code is inside the if.");
    System.out.println("This code is also inside the if.");
}
```
if/else statement

Immediately after an if statement, you can make an else statement

If the “if statement” does not run, then the else statement will

(See: IfElse.java)
Short-circuit evaluation

Short-circuit evaluation is when you have a complex bool expression (&& or ||) but you don't need to compute all parts.

```java
if(false && 7/0 == 3)
{
    System.out.println("Will I crash?");
}
```

If this is false, then it will not check next

(See: ShortCircuit.java)
Short-circuit evaluation

Simple cases of short-circuit:
When you have a bunch of ORs
   if( expression || exp || exp || exp )
Once it finds any true expression, if statement will be true

When you have a bunch of ANDs
   if( expression && exp && exp && exp )
Once it finds any false expression, if statement will be false
Scope

Where a variable is visible is called its **scope**

Typically variables only live inside the block (denoted with matching `{` and `}`)

A variable lives until the block is closed, so inner blocks can see everything from the block it was created inside
Scope

```java
int x;
// can use x here
if(true)
{
    int y;
    // can use x or y here
}
// can use x here

(See: Scope.java)
```
Semicolons

The general rule for semicolons:

After every line/statement...
EXCEPT, if you start a new block

```java
int x = 2;  // line, no new block = put ;
if(x != 5)  // line, new block = no ;
{
    System.out.println("Hi");
}
```
Initialization using if

When initializing with an if-statement, you must always have an else

The computer is not smart enough to figure out these two are the same:

```java
int y;
if (x > 0) {
    y = 1;
}
else {
    y = 2;
}
```

(See: IfInitialize.java)
if/else vs loops

if/else statements makes code inside only sometimes run

Loops make code inside run more than once

Both use boolean expressions to determine if the code inside is run
while loop

A while loop tests a **boolean** expression and will run until that expression is **false**

```java
while (i < 10) {
    // looped code
    // variable i should change in here
}
```

(See: WhileLoop.java)
while loop

The **boolean** expression is tested when first entering the while loop
And!
When the end of the loop code is reached (the } to close the loop)

```java
int i = 0;
while (i < 5) {
    System.out.println("Looping, i = " + i);
    i++;
}
System.out.println("Outside loop, i = " + i);
```
while loop

It can be helpful to manually work out what loops are doing and how variables change in each loop iteration.

This will build an insight into how loops work and will be beneficial when working with more complicated loops.
while loop

3 parts to any (good) loop:

- Test variable initialized
  
  ```
  i=0;
  ```

- boolean expression
  
  ```
  while (i < 10)
  ```

- Test variable updated inside loop
  
  ```
  i++; 
  ```
while loop

As uninitialized variables and \texttt{boolean} expressions are required (syntax errors)...

The harder errors to catch are missing/bad updates, which can loop forever

(See: \texttt{InfiniteLoop.java})
Infinite loops