

## IceBlox

- o What did you think?

## Style

- o Style is not for the computer, it is for other humans to read your code
  - o Companies generally have a style you must follow.
- o It is easy to make things non-readable
  - o That means nobody else can maintain your code!

## Javadoc

- o A tool that comes with the JDK that produces HTML-based documentation from Java Source code.
- o Within a Javadoc comment, various tags can appear which allow additional information to be processed.
  - o Each tag is marked by an @ symbol and should start on a new line.
  - o Javadoc comments have this syntax: `/** comment */`
- o To run at a command line prompt: `javadoc filename.java`
- o The entire Java series of library classes/method/etc. is javadoc'd online for our class at:  
<http://doji.cs.rit.edu/jdoc1.6>

## Started to Refactor...

```
import java.util.Observable;

/**
 * The main "brains" for the IceBlox game.
 *
 * @author Karl Hörnell (javaonthebrain.com)
 * @author Jessica Bayliss (jdb@cs.rit.edu)
 */
public final class IceBlox extends Observable implements Runnable {
    //
    // implementation variables
    //
    public final int NUMBER_X_TILES = 13;
    public final int NUMBER_Y_TILES = 11;
    public final int INTRO = 0;
}
```

## Just a little more...

```
// game information
private long score=0;
private int level = 0;
private int lives = 3;
private int gameState = 0;

// each one of these should derive from character and
// be in an array of characters
// should be a character grouper that you can add and delete
// character information from to keep game code more general
// number of characters includes you
public final int NUMBER_CHARACTERS = 3;
public final int ICE_BLOX = 35;
public final int GOLD_COINS = 5;
public final int ROCKS = 5;

... and more comes after this ...
```

## Refactor

- o Refactoring is the process of rewriting written material to improve its readability or structure, with the explicit purpose of keeping its meaning or behavior.

## Goals of Programming

- o It should work
- o It should be easily readable
- o It should be maintainable
  - o Reduce redundancy
  - o Make it extensible where possible

## Software Engineering Lifecycle

1. Analysis: analyze if a solution is possible and if so come up with a requirements specification
2. Design: A program design for the requirements specification – can be a set of objects/classes
3. Coding: the actual implementation of the design in a particular programming language
4. Testing: unit and integration testing, debugging (activity to eliminate a program error)
5. Operation: software is put into actual use, includes maintenance, added features requested by the customer, cleanup of newly discovered bugs

## Assignment

- o How many different ways can you assign?

hitPoints 

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

```
int hitPoints;  
hitPoints = 2;
```

```
int hitPoints = 2;
```

## Assignment Syntax

```
int dexterity, strength, hitPoints;  
hitPoints = 2;
```

## Primitive Data Types

- o Integers (whole numbers):
  - o byte (1 byte)
  - o short (2 bytes)
  - o int (4 bytes) – default type for numbers typed out in prog
  - o long (8 bytes)
- o Floating point (real numbers):
  - o float (4 bytes)
  - o double (8 bytes) : default for floating point numbers
- o Characters: char (2 bytes)
- o Logical Values: boolean (?? bytes)
  - o boolean is NOT equivalent to 1 and 0 in the Java lang.

## What about different types?

```
double beauty = 97.7;
```

```
double beauty = 97;
```

```
float beauty = 97.7;
```

```
int beauty = 97.7;
```

## From long to byte?

```
long theSkylsTheLimit = 10000;
```

```
int theSkylsTheLimit = 10000;
```

```
short theSkylsTheLimit = 10000;
```

```
byte theSkylsTheLimit = 10000;
```

## Character Representation

```
d 0 1 1 0 0 1 0 0
```

```
char d = 'd';
```

```
char d = 100;
```

The numerical codes used for letters are most often ASCII codes. They are integers.

## Casting

- o Since basic data types are all just bytes, you can try to force them into each other with casting

- o `int weaponDamage = (int)5.5;`
    - o This will truncate 5.5 into a 5

## What Happens?

```
byte b = 2;  
byte b2 = -b;
```

## Basics: +, -, /, \*, %

- o If the / symbol is given 2 integer arguments, it will do integer (truncated!) division
  - o 5/2 will equal 2 and not 2.5
- o The modulus operator gives the integer remainder after division:
  - o 5%2 is 1
  - o 6%3 is 0
  - o 0%2 is 0
  - o 1%2 is 1

## Mod in Games

- o Keeping track of time:
  - o `totalTimeInSeconds % 60`
- o Doing an activity every n game cycles:

```
if ( currentTime % 20 == 0 ) {  
    checkForEnemies();  
}
```
- o Mod in Rome TotalWar?

## Precedence

o What order are operations done in?

o  $5+5*2+3/10\%2$

o From left to right with mult/div/mod first, then add/sub

## Precedence in Games

o What if combat for throwing a spell is calculated:

o  $(\text{decayForLevel} * \text{attackStrength}) + \text{skillBonus}$

o  $\text{decayForLevel} * (\text{attackStrength} + \text{skillBonus})$

o What may happen in either case?

## Operators and Precedence

Description	Syntax
unary postfix	[] . () ++ --
unary prefix	++ -- + - !
creation and cast	New (type)
multiplicative	* / %
additive	+ -
shift	<< >> >>> (unsigned right shift)
relational	< > >= <= instanceof
equality	== !=
and	&
xor	^
or	
boolean and	&&
boolean or	
conditional	?:
assignment	= += -= *= /= %= >>= <<= >> >>= &= ^=  =

## Test-type Question

o (10 points) What would the value of the following expressions be:

o  $22 / 7$

o  $22 \% 7$

o  $6 * 4 / 4 + 2$

o  $5 * (3 + 1) \% 3$

o  $8 / (5 \% 3) / 2$

## Test-type Answers

o (10 points) What would the value of the following expressions be:

o  $22 / 7 = 3$

o  $22 \% 7 = 1$

o  $6 * 4 / 4 + 2 = 8$

o  $5 * (3 + 1) \% 3 = 2$

o  $8 / (5 \% 3) / 2 = 2$

## Boolean And/Or

X    Y			X && Y		
X	Y	Result	X	Y	Result
True	True	True	True	True	True
True	False	True	True	False	False
False	True	True	False	True	False
False	False	False	False	False	False

## Boolean Statements

### o Examples

- o `System.out.println((true && false));`
- o `System.out.println((false || false));`
- o `System.out.println((true || ("troll" == "cute")));`

## Test Type Question

- o What is the output of the following series of statements?

```
int x = 15;
int y = 25;

System.out.println( x != y );
System.out.println( x >= y - x );
System.out.println( x == (y + x - y) );
System.out.println( !(x != 7) && (y % 2 == 0) )
```

## Logical Expressions

- o Basics: `&&`, `||`, `!`, `==`, `!=`, `<`, `>`, `<=`, `>=`
  - o Example: `notDead && notResistant && level < 21 || isARabbit`
- o You can figure out the precedence for these!

## Hello World

- o It's a good test of how much syntax you have to have in a language to get something done.
- o In the Perl language:

```
print ("Hello World\n");
```

- o In a variety of other languages:  
[http://en.wikipedia.org/wiki/Hello\\_world\\_program](http://en.wikipedia.org/wiki/Hello_world_program)

## Lots of Java Syntax

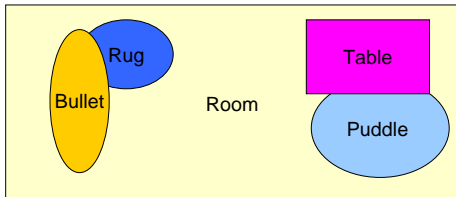
```
/**
 * Look at all this text for a simple
 * printing out of one line. Oh noes!
 */
public class Hello {
    public static void main(String[] args) {
        System.out.println( "Hello World" );
    }
}
```

## Why?

- o We started out with bits, bytes, and made it up to simple types like integers and doubles
- o These are small entities:
  - o How do I represent a graphical world in a game?
  - o Or even a picture?
- o It would be nice to represent them as entities like the simple types, only these entities are conglomerations of basic types.
  - o This is called abstraction

## Java Allows This

- o You can create conglomerations like this and they're called classes (basically your own data types)



## Classes

- o Java is object-oriented:
  - o Classes are more complex data types that allow us to represent things like: Printer, Car, Environment, Character, etc. without having to specify their pieces (arms, legs, whatnot) over and over again every time they're used in a program
  - o Object orientation helps to reduce redundancy
  - o Objects are created from classes
    - o Like cookies from cookie cutters
    - o Like buildings from blueprints

## Object Orientation

- o Class: a template for a data type that tells what it's pieces are and what it does (like a blue print for a character)
- o Object: a specific instance (instantiation) of a class (your specific and personalized character)

## Information in a Class

- o Methods: what the class does
  - o The main method is always the starting method in a Java application
- o State: Variables, constants, and other data inside a class
- o Objects personalize this information from the class template
  - o Hair color, character type, weapons holding, etc.

## Methods and State



## How Memory Works for Objects

Java declaration: `Duck evilDuck = new Duck();`

evilDuck (holds 4) 

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

4 is the starting address of this Duck object

...								
4	0	1	0	0	0	1	0	0
	1	0	0	0	0	1	0	1
	0	0	1	0	0	1	0	0
	1	0	0	0	1	1	0	1
...								

## References

- o Basic types are held in memory wherever the variable name symbolically refers to
- o References (all objects are references) have the name reference the address where the beginning of the object is stored

## Basic Types vs. References

- o Basic Type: you get a physical letter in your mailbox
- o Reference: You get a note in your mailbox that tells you to go to the post office desk to pick up a package

## Hello Revisited

```
/**
 * I changed two lines from the previous version
 * of the program
 */
public class Hello {
    public static void main(String[] args) {
        String hello = new String("Hello World");
        System.out.println( hello );
    }
}
```

## Strings are special

- o A specific string is really an object:
  - o String bird = new String("bird");
- o But: it can be referred to in a similar way to basic types:
  - o String bird = "bird";
- o These two statements are almost equivalent, but not quite. String literals share the same memory location if they have the same value whereas new Strings always have different memory locations.