Useless Fact of the Day

- I just saved a bunch of money on my car insurance by switching to Geico.

- (I hate that little gecko thing)

Topics

- More Basics About Java
- Coding Conventions
- Anatomy of a Class (again)
- Data Types
- Operations
- Tricky Ramifications
CPUs, and Machine Code

- The CPU is a computer’s processor -- it executes instructions (calculations on data).
- Everything is data in a computer (a picture is really just one really really big number).
- Many types of CPU (Intel Pentium, PowerPC, ARM processors in cell phones, etc.)
- Each has a different set of instructions it is capable of working with.
- Code with these instructions is machine code (or assembly).
- Programming in straight machine code is horrible.

Java

- Java is a higher-level programming language.
- It’s more human-readable than machine code.
- It lets you think more abstractly about your program instead of worrying about every little detail (although you still have to worry about many little details).
- A single statement in a high-level language often represents several machine code instructions.

Compiling

- A compiler must be used to convert (compile) code into machine code.
- Happens when you click the little running man in Eclipse.
- Many high-level languages are compiled: C, C++, Fortran, Pascal, etc.
- ...but for most, they must be re-compiled to suit whatever processor they are on.
- Compiled Java code runs on Macs, PCs, and Linux computers!
- This is because it runs on a virtual machine which is running in your real machine.
- The virtual machine uses the same machine code instructions no matter what processor (Intel, PowerPC) and OS (Mac, Windows) it is running on.

Java Compiling -- .class

- When you code, you write “.java” files
- These are compiled into corresponding “.class” files
- (you may have noticed that the game engine is full of .class files -- you can’t edit these, which keeps you from accidentally messing anything up when you’re looking around at them)
- These are sometimes called Java bytecode files.
Syntax

- **Syntax** is a set of rules to which a program must conform to be valid (like grammar, in English).
- Java is **case sensitive** -- “sprite” is not the same as “Sprite”
- Every statement (line) ends with a **semi-colon**, “;”, except for statements which have a body afterwards enclosed in squiggly braces, “{ }”
- White space (spaces, returns, tabs) is ignored -- you could write the entire program on one long line, but this would not be readable at all!
- When you make a **syntax error**, Eclipse will underline it for you in red (hover over the error to see what it is).

Coding Conventions

- Not necessary for the computer, but important if any human is to understand your code
- **Capitalization**
  - Class identifier: first letter (“Sprite”)
  - Variable and method identifiers: start lowercase (“myVariable,” or “my_variable”)
  - Constants: all capitals (“MY_CONSTANT,” “Math.PI,” “Color.BLUE”)
- **Indentation**
  - Indent all code in between a set of two matching braces (a code block)
  - (and if there’s another set of two matching braces inside, indent again... etc.)
- **Class/Method Organization**
  - Usually, declare instance variables and local variables first (at the top of a code block)
  - **Comment Often!!!!!!!!!!!!!!!!!!**

Creating a Variable

- Two steps:
  1. **Declare** the variable (if it’s a local variable, declare it at the top of a method; if it’s an instance variable, then declare it at the top of the class body): (only needed if it is an instance variable)
     ```java
     public Coordinate myPoint;
     ```
  2. **Initialize** (or **define**) the variable to some value (do this in a method, usually):
     ```java
     myPoint = new Coordinate(3, 7);
     ```
     - Can optionally do both steps in one line:
       ```java
       Coordinate myPoint = new Coordinate(3, 7);
       ```

Anatomy of a Class

- **public class Rabbit**
  ```java
  public class Rabbit {
  public int weight;
  public int hungriness;
  public Rabbit(int w, int h) {
    weight = w;
    hungriness = h;
  }
  public void eat(Grass g) {
    weight = weight + g.amount;
    hungriness = hungriness - g.amount;
  }
  }
  ```
  - **class name (identifier)**
  - **class declaration**
  - **class definition**
  - **data members (instance variables)**
  - **constructor**
  - **class body**
  - **a method**
Data Types

- **A data type** defines the structure of information
- Can be existing (built-in to Java) or user-defined (by writing a class)
- **A primitive** only stores a single value (int, double)
- **A reference** (class) type can store many values and/or methods (Rectangle, Rabbit)

Operations

- **Operators (+, -, etc.)**
  - Most are binary (require two operands)
  - Some are unary
- **Methods** (Math.sin(double x), Rabbit.eat(Grass g), etc.)
  - A **static** method does not require an instance (object) of the class to exist in order to be used (Example: Math.sin(double x))
  - An **instance** method does require an instance of the class to exist (Example: The eat(Grass g) function in the Rabbit class must be run on a particular Rabbit -- if we had a Rabbit called "r" and a Grass called "g", we could say "r.eat(g)", but we could never say "Rabbit.eat(g)"
  - May or may not compute a value
  - Can have zero or more parameters
Operations

- Methods:
  - Math.sin(double), Math.cos(double) sine and cosine (static)
  - Math.abs(double) absolute value (static)
  - Math.random() random number between 0 and 1 (static)
  - Math.sqrt(double) square root (static)
  - System.out.println(String or int or double or ...) prints argument to the console (static)
  - Integer.parseInt(String) turns a string (say, “123”) into an integer (123)
  - [Double or Integer or Rectangle or ...].toString() returns the string representation of an object (instance)
  - canvas.addSprite(Sprite) adds a sprite to the canvas (instance)
  - Tiger.eat(Rabbit) eats a rabbit (instance)

Tricky Ramifications

- Integer truncation:
  - If you attempt to store 5.9 in an int variable, it will be truncated and stored as 5

- Integer vs. double division:
  - 19 / 10 → 1
  - 19.0 / 10.0 → 1.9
  - 19 / 10.0 → 1.9
  - (double) 19 / 10 → 1.9
  - int x = 3; int y = 2; x / y → 1
  - int y = 2; double x = 3; int y = 2; x / y → 1.5