Classes, Data Types, and Operations
Lecture 3 (7/5/2006)

Useless Fact of the Day
• Nursing blue whale calves consume 100 gallons of mother’s milk, gain 200 pounds, and grow 1.5 inches in length every day

Topics
• Anatomy of a Class (again)
• Coding Conventions
• Data Types
• Operations
• Tricky Ramifications

Anatomy of a Class
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;
}
Coding Conventions

• Not necessary for the computer, but important if any human is to understand your code

• Capitalization
  • Class identifier: first letter ("Rectangle")
  • Variable and method identifiers: start lowercase ("myVariable," or "my_variable")
  • Constants: all capitals ("MYCONSTANT," "Math.PI," "Color.BLUE")

• Indentation
  • Indent all code in between a set of two matching braces

• Class Organization
  • Declare data members and local variables first (at the top of a code block)

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)

Data Types

• Primitives:
  • double real numbers
  • float less-precise real numbers
  • int integers
  • boolean true or false
  • char single character (letter, number, space, or punctuation)
  • long for extremely large (+ or -) integers

• References:
  • String text (multiple characters) (in java.lang)
  • Rectangle stores position/size of a rectangle (in java.awt)
  • Rabbit stores size/hungriness of a rabbit (user-defined)

Operations

• Operators (+, -, etc.)
  • Most are binary (require two operands)
  • Some are unary

• Methods (Math.sin, Rabbit.eat, 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

**Operators:**
- `+` add
- `-` subtract/negate
- `/` divide
- `*` multiply
- `%` modulus (remainder)
- `=` assignment
- `==` equivalence
- `!=` not equivalent
- `>` greater than
- `<` less than
- `>=` greater than or equal to
- `<=` less than or equal to

**+=**
x += y; is the same as x = x + y;

**-=**
x -= y; is the same as x = x - y;

**++**
x++; is the same as x = x + 1;

**--**
x--; is the same as x = x - 1;

**&&** logical AND

**||** logical OR

**new** creates an object

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)
- `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
  - `double x = 3;` `int y = 2;` `x / y` → 1.5