Mathematical Operations, Static Methods

Lecture 9, Thu Feb 2 2006

based on slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/cpsc111-06-spr

Reading

- Re-read Chapter 4.3-4.5 (today)
- Next week: Chapter 6 all (6.1-6.4)

News

- Weekly Questions due today
- Midterm reminder: Tue Feb 7, 18:30 - 20:00
  - Geography 100 & 200
- Discovery Forum – here, right after class
  - Computer Science And Medicine: Where Technology Meets Biology
  - you can see demos of what I do when I’m not teaching!

Recap: Commenting Code

- Conventions
  - explain what classes and methods do
  - plus anywhere that you've done something nonobvious
    - often better to say why than what
      - not useful
        int wishes = 3; // set wishes to 3
      - useful
        int wishes = 3; // follow fairy tale convention

Recap: javadoc Comments

- Specific format for method and class header comments
  - running javadoc program will automatically generate HTML documentation
- Rules
  - /** to start, first sentence used for method summary
  - @param tag for parameter name and explanation
  - @return tag for return value explanation
  - other tags: @author, @version
  - */ to end
- Running
  % javadoc Die.java
  % javadoc *.java

Recap: Cleanup Pass

- Would we hand in our code as it stands?
  - good use of whitespace?
  - well commented?
    - every class, method, parameter, return value
    - clear, descriptive variable naming conventions?
    - constants vs. variables or magic numbers?
    - fields initialized?
    - good structure?
- ideal: do as you go
  - commenting first is a great idea!
- acceptable: clean up before declaring victory
### Finishing Point and PointTest

### Formal vs. Actual Parameters
- **formal** parameter: in declaration of class
- **actual** parameter: passed in when method is called
  - variable names may or may not match
  - if parameter is primitive type
    - **call by value**: value of actual parameter copied into formal parameter when method is called
  - changes made to formal parameter inside method body will not be reflected in actual parameter value outside of method
  - if parameter is object: covered later

### Scope
- Fields of class are have **class scope**: accessible to any class member
  - in `Die` and `Point` class implementation, fields accessed by all class methods
- Parameters of method and any variables declared within body of method have **local scope**: accessible only to that method
  - not to any other part of your code
- In general, scope of a variable is block of code within which it is declared
  - **block** of code is defined by braces `{ }`

### Objectives
- Understand how to use mathematical shorthand operators
- Understand when values will be implicitly converted
- Understand how to use static variables and methods

### Increment and Decrement
- Often want to increment or decrement by 1
  - obvious way to increment
    - `count = count + 1;`
  - assignment statement breakdown
    - retrieve value stored with variable `count`
    - add 1 to that value
    - store new sum back into same variable `count`
  - obvious way to decrement
    - `count = count - 1;`

### Shorthand Operators
- **Java shorthand**
  - `count++; // same as count = count + 1;`
  - `count--; // same as count = count - 1;`
  - note no whitespace between variable name and operator
- **Similar shorthand for assignment**
  - `tigers += 5; // like tigers=tigers+5;`
  - `lions -= 3; // like lions=lions-3;`
  - `bunnies *= 2; // like bunnies=bunnies*2;`
  - `dinos /= 100; // like dinos=dinos/100;`
Shorthand Assignment Operators

- What value ends up assigned to `total`?
  ```java
  int total = 5;
  int current = 4;
  total *= current + 3;
  ```

- Remember that Java evaluates right before left of `=`
  - First right side is evaluated: result is 7
  ```java
  total *= 7;
  ```

- Result is 7
  ```java
  total = total * 7;
  ```

- `total` is 35

Data Conversion

- Math in your head
  - 1/3 same as .33333333333333333....
- Math in Java: it depends!
  ```java
  int a = 1 / 3;
  double b = 1 / 3;
  int c = 1.0 / 3.0;
  double d = 1.0 / 3.0;
  ```

- Consider each case
  ```java
  int a = 1 / 3;       // a is 0
  double b = 1 / 3;    // b is 0.0
  int c = 1.0 / 3.0;   // Java's not happy
  double d = 1.0 / 3.0; // d is 0.333333333
  ```

- Literals 1 and 3 are integers
- Arithmetic with integers results in integer
  - Fractional part truncated (discarded)
- So 0 is value assigned to `a`

- Arithmetic with doubles results in double
  - Results is 0.3333333....
- Left side expects int not double
  - Fractional part would have to be truncated
- Java wants to make sure you know you'd lose fractional information
  - Could be explicit with cast

  ```java
  int c = (int) (1.0 / 3.0); // cast placates Java
  ```
Data Conversion
- Consider each case
  ```java
double d = 1.0 / 3.0;  // d is 0.33333333
```
- Literals 1.0 and 3.0 are doubles
- Arithmetic with doubles results in double
  - results is 0.3333333...
- Right side double can hold value
  - well... just approximation of repeating value!
    - finite number of bits to hold infinite sequence
    - roundoff errors can be major problem
  - CPSC 302, 303 cover in more detail

Casting: explicit data conversion
- Widening: conversion from one data type to another type with equal or greater amount of space to store value
  - widening conversions safer because don't lose information (except for roundoff)
- Narrowing: conversion from one type to another type with less space to store value
  - important information may be lost
  - avoid narrowing conversions!

Assignment Conversion
- Assignment conversion: value of one type assigned to variable of other type, so must be converted to new type
  - implicit, happens automatically
  - Java allows widening but not narrowing through assignment

Which of these is not a conversion?
- widening conversion?
- narrowing conversion?
  ```java
int a = 1 / 3;         // a is 0
double b = 1 / 3;      // b is 0.0
int c = 1.0 / 3.0;     // Java's not happy
double d = 1.0 / 3.0;  // d is 0.3333333333333333
```

Promotion
- Second kind of data conversion
  - happens when expression contains mixed data types
  - example:
    ```java
    int hours_worked = 40;
double pay_rate = 5.25;
double total_pay = hours_worked * pay_rate;
    ```
- To perform multiplication, Java promotes value assigned to `hours_worked` to floating point value
  - produces floating point result
  - implicit, widening

Data Conversion
- No such thing as automatic demoting
  - would be narrowing!
  ```java
  int hours_worked = 40;
double pay_rate = 5.25;
double total_pay = hours_worked * pay_rate; // error
  ```
  ```java
  int total_pay = hours_worked * (int) pay_rate;
  ```
Modulus Operator
- computes remainder when second operand divided into first
- sign of result is sign of numerator
- if both operands integer, returns integer
- if both operands floating point, returns floating point
- operator is %

```java
int num1 = 8, num2 = 13;
double num3 = 3.7;
System.out.println( num1 % 3 );
System.out.println( num2 % -13 );
System.out.println( num3 % 3.2 );
System.out.println( -num3 % 3 );
```

Questions?

Static Variables
```java
public class Giraffe {
    private double neckLength;
    public Giraffe(double neckLength) {
        this.necklength = neckLength;
    }
    public void sayHowTall() {
        System.out.println("Neck is "+ neckLength);
    }
}
```

- how would we keep track of how many giraffes we've made?
- need a way to declare variable that "belongs" to class definition itself
- as opposed to variable included with every instance (object) of the class

Static Variables
```java
public class Giraffe {
    private static int numGiraffes;
    private double neckLength;
    public Giraffe(double neckLength) {
        this.necklength = neckLength;
        numGiraffes++;
    }
    public void sayHowTall() {
        System.out.println("Neck is "+ neckLength);
    }
}
```

- updating static variable is straightforward
- increment in constructor

Static Variables
```java
public class Giraffe {
    private static int numGiraffes;
    private double neckLength;
    public Giraffe(double neckLength) {
        this.necklength = neckLength;
        numGiraffes++;
    }
    public void sayHowTall() {
        System.out.println("Neck is "+ neckLength);
    }
}
```

- static variable: variable shared among all instances of class
- aka class variable
- use "static" as modifier in variable declaration
Static Methods

- Static methods do not operate in context of particular object
  - cannot reference instance variables because they exist only in an instance of a class
  - compiler will give error if static method attempts to use nonstatic variable
- Static method *can* reference static variables
  - because static variables exist independent of specific objects
- Therefore, the main method can access only static or local variables.

Static Variables

- Static variable shared among all instances of class
  - Only one copy of static variable for all objects of class
  - Thus changing value of static variable in one object changes it for all others objects too!
- Memory space for a static variable established first time containing class is referenced in program

Static Methods

- Static method "belongs" to the class itself
  - not to objects that are instances of class
  - aka class method
- Do not have to instantiate object of class in order to invoke static method of that class
  - Can use class name instead of object name to invoke static method

Calling Static Method Example

```java
public class UseGiraffes
{
    public static void main (String[] args)
    {
        System.out.println("Total Giraffes: "+
            Giraffe.getGiraffeCount());
        Giraffe fred = new Giraffe(200);
        Giraffe bobby = new Giraffe(220);
        Giraffe ethel = new Giraffe(190);
        Giraffe hortense = new Giraffe(250);
        System.out.println("Total Giraffes: "+
            Giraffe.getGiraffeCount());
    }
}
```

Note that Giraffe is class name, not object name!
- at first line haven’t created any Giraffe objects yet

Static Methods

- Static method "belongs" to the class itself
  - not to objects that are instances of class
  - aka class method
- Do not have to instantiate object of class in order to invoke static method of that class
  - Can use class name instead of object name to invoke static method

Calling Static Method Example

```java
public class UseGiraffes
{
    public static void main (String[] args)
    {
        System.out.println("Total Giraffes: "+
            Giraffe.getGiraffeCount());
        Giraffe fred = new Giraffe(200);
        Giraffe bobby = new Giraffe(220);
        Giraffe ethel = new Giraffe(190);
        Giraffe hortense = new Giraffe(250);
        System.out.println("Total Giraffes: "+
            Giraffe.getGiraffeCount());
    }
}
```

Now you know what all these words mean
- main method can access only static or local variables
Static Methods in `java.Math`:

- Java provides you with many pre-existing static methods.
- Package `java.lang.Math` is part of the basic Java environment.
- You can use static methods provided by the Math class.

Examples:

- `Math.sqrt(36)`
  - 6.0
- `Math.sin(90)`
  - 0.8939966636005579
- `Math.sin(Math.toRadians(90))`
  - 1.0
- `Math.max(54, 70)`
  - 70
- `Math.round(3.14159)`
  - 3
- `Math.random()`
  - 0.7843919693319797
  - 0.4253202368928023
- `Math.pow(2, 3)`
  - 8.0
- `Math.pow(3, 2)`
  - 9.0
- `Math.log(1000)`
  - 6.907755278888577
- `Math.log10(1000)`
  - 3.0