Mathematical Operators, Static Methods

Lecture 14, Fri Feb 5 2010

Shorthand Operators

- **Java shorthand**
  - `count++;` // same as `count = count + 1;`
  - `count--;` // same as `count = count - 1;`
  - `note no` operators match
  - `lions -= 3;` // like `lions = lions - 3;`
  - `bunnies *= 2;` // like `bunnies = bunnies * 2;`
  - `dinos /= 100;` // like `dinos = dinos / 100;`

Data Conversion

- **Math in your head**
  - `1/3` same as `.33333333333333333…`
  - `Math in Java: it depends!`
    - `int a = 1 / 3;` // `a` is `0` double `b = 1 / 3;` // `b` is `.0`

- **Recap:**
  - **javadoc Class Comment Example**
    ```
    /**
     * Die: simulate rolling a die
     * @author: CPSC 111, Section 206, Spring 05-06
     * ... final Die code. We started on Jan 24, tested and improved in on Jan 26, and did a final cleanup pass on Jan 31.
     */
    ```

Increment and Decrement

- **Recap: Formal vs. Actual Parameters**
  - **formal parameter:** in declaration of class
  - **actual parameter:** passed in when method is called
  - parameter 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

Midterm Format Clarification

- you do not need to memorize APIs
  - we will provide javadoc APIs for any classes or methods you need to write/debug code in the exam

Reminder: Lab Schedule Change

- no labs next week Feb 8-12
- TAs will hold office hours in labs during Monday lab times to answer pre-midterm questions
  - Mon Feb 8 11am - 3pm ICICS 008
- labs resume after break
  - staggered to ensure that even Monday morning labs have seen material in previous week's lecture

Recap: 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 {}**

Key Topic Summary

- **Generalizing from something concrete**
  - **fancy name:** abstraction
  - **Hiding the ugly guts from the outside**
    - **fancy name:** encapsulation
  - **Not letting one part ruin the other part**
    - **fancy name:** modularity
  - **Breaking down a problem**
    - **fancy name:** functional decomposition

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?
  - follows specification?
  - ideal: do as you go
    - commenting first is a great idea!
  - acceptable: clean up before declaring victory
Data Conversion

- Consider each case
  - int a = 1 / 3;  // a is 0
  - Literals 1 and 3 are integers
  - Arithmetic with integers results in integer
  - fractional part truncated (discarded)
  - So 0 is value assigned to a

- Consider each case
  - double b = 1 / 3;  // b is 0.0
  - Literals 1.0 and 3.0 are doubles
  - Arithmetic with doubles results in double
  - results is 0.3333333333333333
  - 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
  - int c = (int) (1.0 / 3.0);  // cast placates Java

- 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!

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

- No such thing as automatic demoting
  - Would be narrowing!
  - int hours_worked = 40;
  - double pay_rate = 5.25;
  - double total_pay = hours_worked * pay_rate;  // error

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 %

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

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

- Updating static variable is straightforward
  - increment in constructor

Questions?

- Second kind of data conversion
  - happens when expression contains mixed data types
  - example:
    - 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

- public class Giraffe {
  - private double neckLength;
  - public Giraffe(double neckLength) {
    - this.neckLength = neckLength;
  }

  - public void sayHowTall() {
    - System.out.println("Neck is "+ neckLength);  
  }
}
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

```
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);
  }
  public static int getGiraffeCount() {
    return numGiraffes;
  }
}
```

static method example

```
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 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.

Calling Static Method Example

```
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 basic Java environment
- you can use static methods provided by Math class
- examples:
  - Math.sqrt(36)
  - Math.sin(90)
  - Math.sin(Math.toRadians(90))
  - Math.max(54,70)
  - Math.round(3.14159)
  - Math.random()
  - Math.pow(2,3)
  - Math.pow(3,2)
  - Math.log(1000)
  - Math.log10(1000)

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.