Static Methods, Conditionals

Lecture 15, Mon Feb 8 2010

borrowing from slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/111-10
Events this week

Schlumberger Info Session
Date: Mon., Feb 8
Time: 5:30 pm
Location: HENN Rm 201

Finding a Summer Job or Internship Info Session
Date: Wed., Feb 10
Time: 12 pm
Location: X836

Masters of Digital Media Program Info Session
Date: Thurs., Feb 11
Time: 12:30 – 1:30 pm
Location: DMP 201

Reminder: Co-op Deadline
Date: Fri., Feb 12
Submit application to Fiona at Rm X241 by 4:30 pm
Reminders

- midterm tonight, 6:30 FSC 1005
- no labs this week Feb 8-12
  - labs resume after break
Recap: 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;
Recap: Data Conversion

Math in Java: it depends!

```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 0.3333333333
```
Recap: Data Conversion

- **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)
  - Java will do widening conversions automatically

- **Narrowing**: conversion from one type to another type with less space to store value
  - Important information may be lost
  - Java will not do narrowing conversions automatically
Recap: Automatic Conversion

- Done implicitly if widening

- **Assignment conversion**: converted because value of one type assigned to variable of other type

  ```java
  double b = 1 / 3;
  ```

- **Promotion**: converted because expression contains mixed types

  ```java
  int hours_worked = 40;
  double pay_rate = 5.25;
  double total_pay = hours_worked * pay_rate;
  ```
public class Giraffe {
    private double neckLength;
    public Giraffe(double neckLength) {
        this.necklength = necklength;
    }
    public void sayHowTall() {
        System.out.println("Neck is " + neckLength);
    }
}
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;
    }
    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
public class Giraffe {
    private static int numGiraffes;
    private double neckLength;
    public Giraffe(double neckLength) {
        this.necklength = necklength;
        numGiraffes++;  // updating static variable is straightforward
    }
    public void sayHowTall() {
        System.out.println("Neck is " + neckLength);
    }
}
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
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.
Static Methods

```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 basic Java environment
  - you can use static methods provided by `Math` class
  - examples:

```java
> 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
> Math.random()
0.4253202368928023
> Math.pow(2,3)
8.0
> Math.pow(3,2)
9.0
> Math.log(1000)
6.907755278982137
> Math.log10(1000)
3.0
```
Conditional Statement

- **Boolean expression**: test that returns true or false
- **Conditional statement**: choose which statement will be executed next based on boolean expression
- **Example**

```java
if (age < 20)
    System.out.println("Really, you look like you are " + (age + 5) + ".");
```
import java.util.Scanner;

public class Feelgood
{
    public static void main (String[] args)
    {
        int age;
        Scanner scan = new Scanner (System.in);
        System.out.println("Enter your age: ");
        age = scan.nextInt();
        if (age < 20)
        {
            System.out.println("Really, you look like you "
                                + "are " + (age + 5) + ".");
            System.out.println("You don't look a day over "
                                + (age - 10) + "+!");
        }
    }
}
import java.util.Scanner;

public class Feelgood {
    public static void main (String[] args) {
        int age;
        Scanner scan = new Scanner (System.in);
        System.out.println("Enter your age: ");
        age = scan.nextInt();
        if (age < 20)
            System.out.println("Really, you look like you " + "are " + (age + 5) + ".");
        if (age >= 20)
            System.out.println("You don't look a day over " + (age - 10) + "!");
    }
}
import java.util.Scanner;

public class Feelgood
{
    public static void main (String[] args)
    {
        int age;
        Scanner scan = new Scanner (System.in);
        System.out.println("Enter your age: ");
        age = scan.nextInt();
        if (age < 20)
            System.out.println("Really, you look like you "+"are " +(age + 5) + ".");
        else
            System.out.println("You don't look a day over "+(age - 10) + "!");
    }
}
Conditional In Depth

- Within method, statements usually executed top to bottom one after the other
- Change control flow with conditional statement

```java
if (age < 20)
    System.out.println("Really, you look like you are " + (age + 5) + ".");
```

- Choice hinges on evaluation of boolean operator
Boolean Expressions

- Boolean expression: test which returns either true or false when evaluated
  - aka conditional

- Consists of operands and operators, like arithmetic expression
  - but operators only return true or false when applied to operands

- Two different kinds of operators
  - relational
    - sometime split into relational and equality
  - logical
Relational Operators

- Tests two values (operands)

- Operators
  - `==` equal
    - returns true if they are equal, false otherwise
    - note: do not confuse this with `=`
  - `!=` not equal
    - returns true if they are not equal, false otherwise
  - `<` less than
  - `<=` less than or equal to
  - `>` greater than
  - `>=` greater than or equal to
Equality Example

```java
int a = 3;
int b = 6;
int c = 10;

if (a == b)
    System.out.println("these two values are equal");

if ((b - a) == a)
    System.out.println("b is the same as a");

if (a != b)
    System.out.println("nope!");
```

- Note we can use arithmetic operator inside boolean expression
Logical Operators

- Way to combine results from relational operators into single test
- AND, OR, and NOT
  - in terms from math or philosophy class
- Operators
  - && logical AND
  - || logical OR
  - ! logical NOT
Logical AND

- Logical AND of values a and b evaluates to
  - true if both a and b are true
  - false otherwise

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a &amp;&amp; b</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
**Logical OR**

- Logical OR of values a and b evaluates to
  - true if either a or b are true
  - true if both are true
  - false otherwise

| a   | b   | a || b |
|-----|-----|------|
| false | false | false |
| false | true  | true  |
| true  | false | true  |
| true  | true  | true  |
Logical NOT

- Logical NOT of value $a$ evaluates to
  - true if $a$ is false
  - false if $a$ is true

<table>
<thead>
<tr>
<th>$a$</th>
<th>$!a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
Logical Operator Examples

```java
int a = 3;
int b = 6;
int c = 10;

if ((b > a) && (c == 10))
    System.out.println("this should print");

if (!(b > a))
    System.out.println("this should not print");

if !(b > a)
    System.out.println("what happened?");
```
Logical Operator Examples

- is \(! (b > a)\) the same as
  - \((a > b)\)
  - \((a \geq b)\)
  - \((b < a)\)