Conditionals II

Lecture 16, Wed Feb 10 2010

borrowing from slides by Kurt Eiselt

http://www.cs.ubc.ca/~tmm/courses/111-10
Recap: 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
Recap: 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
```
Recap: Conditional Statement

- **Conditional statement**: choose which statement will be executed next based on boolean expression
  - changes control flow
- **Example**

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

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

- Understand how to compare objects and primitive data types
- Understand syntax to use for conditionals and switch statements
Comparing Strings

- How do we test for equality between Strings?
- Reminder:
  - Strings are sequences of alphanumeric characters
  - create with constructor
    - `String firstname = new String("Donald");`
  - or with shortcut
    - `String lastname = "Duck";`
  - Strings are objects, not primitive types!
Comparing Strings

- Relational operator == is wrong way to compare

```java
String name1 = "Bubba";
String name2 = "Bubba";
System.out.println(name1 == name2);  // prints false
```

- Equals method is right way to compare Strings

```java
String name1 = "Bubba";
String name2 = "Bubba";
System.out.println(name1.equals(name2));  // prints true
```

- why? diagrams will help
Comparing Strings

- these values tested for equality with test of `name1 == name2`
- two different pointers (references), so `false`
Comparing Strings

- these values tested for equality with `name1.equals(name2)`
- contents of objects are same, so true
Short-Circuiting Evaluation

■ Consider again expression

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

■ Java evaluates left to right
  ■ if \((b > a)\) is false, does value of \((c == 10)\) matter?
  ■ no! result of && must be false since one operand already evaluated to false
  ■ short-circuiting: Java does not evaluate
    ■ aka lazy evaluation
Short-Circuting Evaluation

■ Consider different expression

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

■ Java evaluates left to right
  ■ if (b>a) is true, does value of (c == 10) matter?
  ■ no! result of || must be true since one operand already evaluated to true
If Syntax

- Syntax
  - reserved word `if`
  - followed by boolean expression enclosed in parentheses
  - followed by statement

```java
if (x == y)
    System.out.println("x equals y! ");
```

- Results
  - if boolean evaluates to true, statement is executed
  - otherwise statement is skipped, execution continues with statement immediately following if statement
If-Else Syntax

- If statement may include optional else clause
  - reserved word else
  - followed by another statement

```java
if (x == y)
    System.out.println("x equals y!");
else
    System.out.println("x is not equal to y!");
```

- Results
  - if boolean evaluates to true, first statement is executed
  - otherwise (if boolean evaluates to false), statement following else is executed
Block Statements

- Often want to do many actions, not just one, based on condition
- Replace single statement with many statements surrounded by curly braces

```java
if (x == y)
{
    System.out.println("x equals y!");
    System.out.println("I'm happy");
}
else
{
    System.out.println("x is not equal to y");
    System.out.println("I'm depressed");
    System.out.println("How about you?");
}
```
What if we leave out block in else clause?

```java
if (x == y)
{
    System.out.println("x equals y!");
    System.out.println("I'm happy");
}
else
    System.out.println("x is not equal to y");
    System.out.println("I'm depressed");
    System.out.println("How about you?");
```
Nested If Syntax

- Statements within if-else statements can themselves be if-else statements

```java
public class NestTest {
    public static void main (String[] args) {
        int x = 1; int y = 3; int z = 2;

        if (x == y)
            if (y == z)
                { System.out.println("all three values the same"); }
            else
                { System.out.println("y is not equal to z"); }
        else
            System.out.println("x is not equal to y");
    }
}
```
Nested If Syntax

- Multiple `else` statements also legal

```java
if( Boolean expression 1 )
{
    // statements
}
else if( Boolean expression 2 )
{
    // statements
}
else if( Boolean expression 3 )
{
    // statements
}
else
{
    // statements
}
```
Nested If Syntax

- Rewriting `NestTest` using multiple else statements

```java
public class NestTest2 {
    public static void main (String[] args) {
        int x = 1; int y = 3; int z = 2;

        if ((x == y) && (y == z)) {
            System.out.println("all three values the same");
        } else if ((x == y) && (y != z)) {
            System.out.println("y is not equal to z");
        } else {
            System.out.println("x is not equal to y");
        }
    }
}
```
Comparing Floating Point Numbers

- Is 0.3 the same thing as
  1.0/10.0 + 1.0/10.0 + 1.0/10.0 ???

- Let’s try it out...
Comparing Floating Point Numbers

- Is 0.3 the same thing as
  1.0/10.0 + 1.0/10.0 + 1.0/10.0 ???

- No - very close, but not exactly what you expect
  - 0.3000000000000004

- Beware! Write tests for “darn near equal” like:

```java
if (Math.abs(f1 - f2) < TOLERANCE)
    System.out.println ("Essentially equal.");
```

- where TOLERANCE is small number appropriate to problem
  like 0.00000001
Comparing Characters

- You can compare character types with relational operators

  'a' < 'b'
  'a' == 'b'
  'a' < 'A'

- Remember, cannot compare Strings with relational operators
  - or any other objects!
  - must use methods like equals
Switch Syntax

- Use `switch` statement to get program to follow one of several different paths based on single value

```java
switch (finalMark)
{
    case 4:
        System.out.println("You get an A");
        break;
    case 3:
        System.out.println("You get a B");
        break;
    case 2:
        System.out.println("You get a C");
        break;
    default:
        System.out.println("See you next year");
}
```
Switch Syntax

- Expression should be int, char
  - (or enumerated type)

```java
switch (finalMark)
{
    case 4:
        System.out.println("You get an A");
        break;
    case 3:
        System.out.println("You get a B");
        break;
    case 2:
        System.out.println("You get a C");
        break;
    default:
        System.out.println("See you next year");
}
```
Switch Syntax

- Case values cannot be variables

```java
switch (finalMark) {
    case 4:
        System.out.println("You get an A");
        break;
    case 3:
        System.out.println("You get a B");
        break;
    case 2:
        System.out.println("You get a C");
        break;
    default:
        System.out.println("See you next year");
}
```
Switch Syntax

- Default statement optional, but very good idea

```java
switch (finalMark)
{
    case 4:
        System.out.println("You get an A");
        break;
    case 3:
        System.out.println("You get a B");
        break;
    case 2:
        System.out.println("You get a C");
        break;
    default:
        System.out.println("See you next year");
}
```
Switch Syntax

- Break statements really important

```java
switch (finalMark) {
    case 4:
        System.out.println("You get an A");
        break;
    case 3:
        System.out.println("You get a B");
        break;
    case 2:
        System.out.println("You get a C");
        break;
    default:
        System.out.println("See you next year");
}
```