



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

2

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:
- ```
> Math.sqrt(36) > Math.random()
6.0 0.7843919693319797
> Math.sin(90) > Math.random()
0.8939966636005579 0.4253202368928023
> Math.sin(Math.toRadians(90)) > Math.pow(2,3)
1.0 > Math.pow(2,3)
> Math.max(54,70) 8.0
70 > Math.pow(3,2)
> Math.round(3.14159) 9.0
3 > Math.log(1000)
 6.907755278982137
 > Math.log10(1000)
 3.0
```

3

Recap: Conditional Statement

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

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

4

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

5

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

6

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

7

Logical Operator Examples

- is `!(b > a)` the same as
  - `(a > b)`
  - `(a >= b)`
  - `(b < a)`

8

Objectives

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

9

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!

10

Comparing Strings

- Relational operator `==` is wrong way to compare
- ```
String name1 = "Bubba";
String name2 = "Bubba";
System.out.println(name1 == name2); // prints false
```
- `Equals` method is right way to compare Strings
- ```
String name1 = "Bubba";
String name2 = "Bubba";
System.out.println(name1.equals(name2)); // prints true
```
- why? diagrams will help

11

Comparing Strings

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

12

Comparing Strings

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

13

Short-Circuiting Evaluation

- Consider again expression
- ```
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**

14

Short-Circuiting Evaluation

- Consider different expression
- ```
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

15

If Syntax

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

16

If-Else Syntax

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

```
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

17

Block Statements

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

```
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?");
}
```

18

Block Statements

- What if we leave out block in else clause?

```
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?");
```

19

Nested If Syntax

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

```
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");
    }
}
```

20

Nested If Syntax

- Multiple `else` statements also legal

```
if (Boolean expression 1 )
{
    // statements
}
else if ( Boolean expression 2 )
{
    // statements
}
else if ( Boolean expression 3 )
{
    // statements
}
else
{
    // statements
}
```

21

Nested If Syntax

- Rewriting `NestTest` using multiple else statements

```
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");
    }
}
```

22

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
 - 0.30000000000000004
- Beware! Write tests for "dam near equal" like:

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

- where TOLERANCE is small number appropriate to problem like 0.00000001

24

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

25

Switch Syntax

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

```
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");
}
```

26

Switch Syntax

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

```
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");
}
```

27

Switch Syntax

- Case values cannot be variables

```
switch (finalMark)
{
    case f:
        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");
}
```

28

Switch Syntax

- Default statement optional, but very good idea

```
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");
}
```

29

Switch Syntax

- Break statements really important

```
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");
}
```

30