Conditionals

CSE 142, Summer 2003

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/03su/

Readings and References

- Reading
  - » Chapter 6, *Intro to Programming and Object-Oriented Design Using Java*, Niño and Hosch
- References
  - » "Language Basics", Java tutorial
    http://java.sun.com/docs/books/tutorial/java/nutsandbolts/index.html

Implementing Interesting Behavior

- We need to be able to make decisions in order to have objects behave in interesting ways
  - » Has this Acrobat been asked to do anything yet?
  - » Did the user supply any arguments to the program?
  - » Is the display window visible?
  - » Is **`myPet`**'s name the same as **`yourPet`**'s name?
- The **`if`** statement is our primary tool for changing the flow of control in the program

Sequences and Blocks

```
// Simple sequence of statements
statement1;
statement2;
// Block - can replace a single statement anywhere
{
   statement1;
   statement2;
}
```

The `if` statement

```
if (condition) {
    this block is executed if the condition is true
} else {
    this block is executed if the condition is false
}
```

- The condition is a logical expression that is evaluated to be **`true`** or **`false`**, depending on the values in the expression and the operators

operators that produce boolean results

- All of the normal arithmetic comparison operators are available

`>`	:	greater than
`<`	:	less than
`>=`	:	greater than or equal
`<=`	:	less than or equal
`==`	:	equal
`!=`	:	not equal

examples

- numeric comparisons are extremely common

```
if (count == limit) {
   messageDialog.warn("count has reached limit");
}
```

```
public void twirl(int k) {
   System.out.println(familyName+" twirled "+k+" times.");
   twirlCount = twirlCount + k;
   if (twirlCount > twirlTarget) {
      System.out.println("I'm getting tired of twirling!");
   }
}
```

see Acrobat in ex8

Compound expressions

- We can combine various logical expressions together to make one larger expression

```
if (arg != null && arg.equals("begin")) {
   process the beginning of something …
}
```

- There are operators for "and", "or" and "not"

`&&`	:	**`and`**
`||`	:	**`or`**
`!`	:	**`not`**

examples

- the “not” operator can be handy for clarity in some cases, but it can also be confusing, so use carefully

```
if (!ready) {
    messageDialog.warn(“system not ready”);
}
```

- the **&&** and **||** operators are “shortcut” operators
  - » they stop evaluation as soon as the logical condition is satisfied

```
if (arg != null && arg.equals(“green”)) {
   myColor = Color.green;
}
```


Use braces and parentheses liberally

- Better safe than sorry
  - » Parentheses surround parts of an expression
  - » Braces surround a block of code, even one line

```
if ((a==b) && ((c+d) == e)) {
  state.advance(a);
} else {
  state.retreat(e);
}
```


multiple cases

- You can chain **if** statements together to select one of several possibilities

```
if (arg.equals(“green”)) {
   myColor = Color.green;
} else if (arg.equals(“blue”)) {
   myColor = Color.blue;
} else {
   myColor = defaultColor;
}
```


boolean expressions and variables

- If you find yourself doing something like this

```
if (pageNumber == lastPage) {
  allDone = true;
} else {
  allDone = false;
}
```

- there is an easier way

```
allDone = (pageNumber == lastPage);
```


boolean variable


boolean expression

conditional operator (3 operands)

- If you find yourself doing something like this

```
if (score < 0) {
  color = Color.red;
} else {
  color = Color.black;
}
```

- there is an easier way

```
color = (score < 0) ? Color.red : Color.black;
```

variable

boolean expression

use this value if expression is true

use this value if expression is false

Expression using a returned boolean

- methods can return boolean values too

```
if (arg.equals("green")) {
   myColor = Color.green;
} else {
   myColor = defaultColor;
}
```


returning a boolean value

- It is often convenient to return a boolean expression from a method

```
public boolean isEmpty() {
  return (this.itemCount == 0);
}
```

itemCount is an instance variable in this example

comparing floating point numbers

- Never, never test for exact equality of two floating point numbers using ==
  - » double and float values are approximate values which may vary slightly way out to the right of the decimal point
  - » 1.00000000000000000000000001
  - » 1.00000000000000000000000002
  - » Are they equal?
    - NO. But probably close enough for our purposes ...

floating point compare

- check for exceeding a limit

```
if (xVal >= maxX) { …
if (yVal < 0.0) { …
```

- check for difference less than some small amount

```
double epsilon = 0.00001;
if (Math.abs(xVal-xGoal) < epsilon) {...
```


switch statement

```
switch (integral type) {

   case value1 : {
      statement1;
      break; //Break out of switch
   }
   case value2 : {
      statement2;
      break;
   }
   default : {
      statement3;
   }

}
```

there are lots of limitations and potential bugs in using this, so be careful!