Conditional Statements

CS 99 – Summer 2000
Michael Clarkson
Lecture 4

Administration

- Lab 2 due now on floppy
- Lab 3 due tomorrow via FTP
  - need Instruct account password
- Lab 4 posted this afternoon
- Prelim 1 in 1 week
  - review session TBA

Agenda

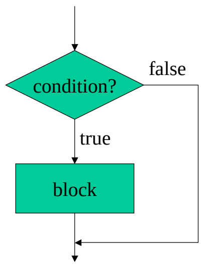
- `if` statements
- `switch` statements
- `?:` (the conditional operator)

`if`

- The `if` (also `if-then`) statement chooses whether to run code, or not:

```
if (grade > 60) {
  System.out.println("You are passing!");
}
```

Flow of control



Syntax

```
if (condition) {
  …
}
```

Example:

```
if (temperature > 80) {
  System.out.println("It's hot!");
}
```

Syntax [2]

- The condition of an `if` statement can be any Boolean expression
- There may be any number of statements in the body of the `if`
- What isn't allowed in the body?

Nested `if`s

- A *nested* statement is a statement inside another, e.g.:

```
if (month == 12) {
  if (day == 31) {
     System.out.println(“It’s New Year’s Eve”);
  }
}
```

How else could this be written?

… or `else`!

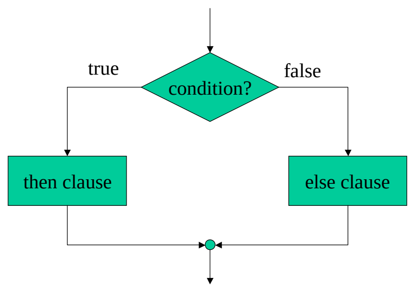
- It's also possible to make an `if` statement choose between two blocks of code.
- Syntax:

```
if (condition) {
  …
} else {
  …
}
```

`if-else`

- Semantics (meaning):
  - evaluate the condition
  - if the condition is true, execute the first block (the *then clause*)
  - if the condition is false, execute the second block (the *else clause*)
  - no matter which block was executed, resume executing with the code after both blocks

`if-else` [2]



`if-else` [3]

- Example: print the larger of two numbers

```
if (num1 > num2) {
  System.out.println(num1);
} else {
  System.out.println(num2);
}
```

How would this look in a larger program?

```
class LargerOfTwo {
     public static void main(String[] args) {
          int num1 = Console.readInt();
          int num2 = Console.readInt();

          if (num1 > num2) {
               System.out.println(num1);
          } else {
               System.out.println(num2);
          }

          System.out.println(“All done”);
     }
}
```

Robust Programs

- A *robust* program is completely protected against all possible crashes from bad data and unexpected values.
- Example:

```
System.out.print(“Enter a positive number: “);
p = Console.readInt();
if (p <= 0) {
   System.out.println(“You entered a nonpositive
   number”);
} else {
   // do whatever we expected to do, e.g., Math.sqrt(p)
}
```

Robust Programs [2]

- This kind of protection can be used many places in a program
- Requires extra code (and time!) at the tradeoff of fewer possible crashes
- Essential for professionals
- For students, can detract from programming principles
- In this class, you can assume we will always give your program valid input unless we specifically warn you otherwise

A Complicated `if`

- How could we express the following in Java?
  - If a student’s grade is:
    - 90-100, print “A”
    - 80-89, print “B”
    - 70-79, print “C”
    - 60-69, print “D”
    - 0-59, print “F”

A Complicated `if` [2]

```
if (grade >= 90) {
   System.out.println(“A”);
} else {
   if (grade >= 80) {
        System.out.println(“B”);
   } else {
        if (grade >= 70) {
                System.out.println(“C”);
        } else {
                if (grade >= 60) {
                        System.out.println(“D”);
                } else {
                        System.out.println(“F”);
                }
        }
   }
}
```

Extended `if`

- An extended `if` expresses a multiway decision – a choice of one of several alternatives
- Allows problems that would require deeply nested `if`s to be expressed more concisely

Extended `if` [2]

- Syntax:

```
if (condition1) {
  // block1
} else if (condition2) {
  // block2
…
} else if (conditionN) {
  // blockN
} else {
  // else clause
}
```

•Semantics:
- •Pick the first true condition
- •Execute its block only
- •If no conditions are true, execute the else clause

A Less Complicated `if`

```
if (grade >= 90) {
  System.out.println("A");
} else if (grade >= 80) {
  System.out.println("B");
} else if (grade >= 70) {
  System.out.println("C");
} else if (grade >= 60) {
  System.out.println("D");
} else {
  System.out.println("F");
}
```

Braces Optional

- If there's only one statement in the body of an `if`, the braces are optional:

```
if (temperature < 0)
  System.out.println("It's c-c-cold!");
```

- Strong recommendation: put them in anyway!
  - fewer bugs
  - don't have to add them later
  - avoid the "dangling-else problem"

Dangling Else Problem

```
// test for a perfect square
if (num > 0)
  if (num == Math.pow(Math.sqrt(num), 2))
     System.out.println(num);
else
  System.out.println("Non-positive number");
```

If num is 20, this prints "Non-positive number". Why?

Dangling Else [2]

```
if (num > 0) {
  if (num == Math.pow(Math.sqrt(num), 2)) {
     System.out.println(num);
  }
} else {
  System.out.println("Non-positive number");
}
```

No ambiguity when fully braced.

`switch` statements

- Similar to extended `if`s
- Evaluates a single expression, then chooses one of many paths based on that value
- Could always be replaced by a nested `if`, but is sometimes simpler and easier to read

`switch` Example

```
switch (age) {
case 18:
  System.out.println(“Legal voting age”);
  break;
case 21:
  System.out.println(“Legal drinking age”);
  break;
case 25:
  System.out.println(“Able to rent a car”);
  break;
default:
  System.out.println(“Not an interesting age”);
}
```

`switch` Syntax

```
switch (integer expression) {
case constant expression:
  //statements
  [break;]
[more cases]
[default:
  // default clause]
}
```

`switch` Example [2]

```
switch (num) {
case 1:
case 2:
case 3:
case 4:
  System.out.println(“Less than 5”);
case 5:
case 6:
case 7:
case 8:
case 9:
  System.out.println(“Less than 10”);
}
```

Conditional Operator

- Java’s only ternary operator
- `?:`
- Syntax:

`boolean-expression ? expression : else-expression`

- Semantics:
  - Evaluate the boolean-expression
  - If true, use *expression* as the value of the operation
  - If false, use *else-expression* as the value of the operation

Conditional Operator [2]

- Acts as an abbreviation for `if-else`
- Except that it
  - has a value
  - has operands that are expressions, not full statements
  - is not as easily readable
- You should avoid this operator unless you have a good reason to use it

`?:` Examples

max = (num1 > num2) ? num1 : num2;

abs = (num1 > 0) ? num1 : -num1;