Lecture 11: Switch statements (take-two!), Conditional Operators, Formatting Output, and Operator Precedence and Associativity

Feb 6 2015
Syntax of the switch statement

```c
switch ( SWITCH-EXPRESSION ){
  case VALUE1: STATEMENT11;
      STATEMENT12;
  ...
  break; //End of case VALUE1
  case VALUE2: STATEMENT21;
      STATEMENT22;
  ...
  break; //End of case VALUE2
  ... (more cases if desired) ...
  [default: STATEMENTd1; // Optional clause
   STATEMENTd2;
   ...
   break; ]
}
```
A couple examples

- LetterGrade2.java
- ZodiacSign.java
Why use switch statements?

• Cleaner code
  – Does not require the extra curly brackets
  – Cases are easy to see
  – Typically requires less space
  – Often helps organize logical flow into cases

• More efficient execution
  – For reasons beyond scope of this class

Warning: don’t forget the breaks!
Conditional Operators

Syntax (Ternary operator):

(boolean-expression) ? expression1 : expression2

For example the if-else:

```java
if (x > 0)
    y = 1;
else
    y = -1;
```

is equivalent to

```java
y = (x > 0) ? 1 : -1;
```
Conditional Operator Example

```java
if (num % 2 == 0)
    System.out.println(num + "is even");
else
    System.out.println(num + "is odd");
```

Is equivalent to

```java
System.out.println(((num % 2 == 0)?
    num + "is even" :num + "is odd");
```
Conditional Operator Mini-Quiz

What set of if-else statements is equivalent to:

```c
ticketPrice = (age >= 16) ? 20 : 10;
```
Formatting Console Output

The `printf` statement.

```java
System.out.printf(format, item1, item2, ...);
```

- `format` is a string that may consist of substrings and format specifiers.
  - A format specifier specifies how an item should be displayed.
  - Each specifier begins with a percent sign. `%`

- Each `item` may be a numeric value, character, boolean value, or a string.
# Frequently used specifiers

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Output</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%b</td>
<td>a boolean value</td>
<td>true or false</td>
</tr>
<tr>
<td>%c</td>
<td>a character</td>
<td>'a'</td>
</tr>
<tr>
<td>%d</td>
<td>a decimal integer</td>
<td>200</td>
</tr>
<tr>
<td>%f</td>
<td>a floating-point number</td>
<td>45.460000</td>
</tr>
<tr>
<td>%e</td>
<td>a number in standard scientific notation</td>
<td>4.556000e+01</td>
</tr>
<tr>
<td>%s</td>
<td>a string</td>
<td>&quot;Java is cool&quot;</td>
</tr>
</tbody>
</table>

Suggestion: reread Section 3.16 while at a computer, test out the code snippets, and do the exercises! (3.36, 3.37, 3.38)
Operator Precedence and Associativity

Operator precedence and associativity determine the order in which operators are evaluated.

**High**
- var++, var--
- +, - (Unary plus and minus), ++var,--var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and remainder)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Comparison)
- ==, !=; (Equality)
- ^ (Exclusive OR)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR

**Low**
- =, +=, -=, *=, /=, %= (Assignment operator)
Operator Precedence and Associativity

Operator precedence and associativity determine the order in which operators are evaluated.

**High**
- var++, var--
- +, - (Unary plus and minus), ++var, --var
- (type) Casting
- ! (Not)
- *, /, % (Multiplication, division, and remainder)
- +, - (Binary addition and subtraction)
- <, <=, >, >= (Comparison)
- ==, !=; (Equality)
- ^ (Exclusive OR)
- && (Conditional AND) Short-circuit AND
- || (Conditional OR) Short-circuit OR

**Low**
- =, +=, -=, *=, /=, %= (Assignment operator)

All binary operators are left-associative. (except assignment operators)