CS115 Week 10
Methods/Constructors, Scope, Object Reference, Conditions

Access modifiers - public vs. private
- Classes are usually declared to be public
- Instance variables are usually declared to be private
- Methods that will be called by the client of the class are usually declared to be public
- Methods that will be called only by other methods of the class are usually declared to be private
- APIs of methods are published (made known) so that clients will know how to instantiate objects and call the methods of the class

Methods
// method header
accessModifier returnType methodName(
    parameter list ) {
    // method body
}
- parameter list is a comma-separated list of data types and variable names (these variables are local variables, available in the method only)
- Note that the method header is in the class API

Constructors
- Special methods that are called when an object is instantiated using the new keyword.
- A class can have several constructors.
- The job of the class constructors is to initialize the instance variables of the new object by calling the set????? methods

Class Scope
- Instance variables have class scope
  - Any constructor or method of a class can directly refer to instance variables.
- Methods also have class scope
  - Any method or constructor of a class can call any other method of a class (without using an object reference).

Local Scope
- A method's parameters have local scope, meaning that:
  - a method can access its parameters.
  - a method's parameters cannot be accessed by other methods.
- A method can define local variables which also have local scope, meaning that:
  - a method can access its local variables.
  - a method's local variables cannot be accessed by other methods.
The Object Reference *this*
- How does a method know which object's data to use?
- *this* is an implicit parameter sent to methods and is an object reference to the object for which the method was called.
- When a method refers to an instance variable name, *this* is implied
- Thus:
  
  \[
  \text{variableName} \quad \text{model} \\
  \text{is understood to be} \quad \text{is understood to be} \\
  \text{this.variableName} \quad \text{this.model}
  \]

Object Reference vs. Object Data
- Object references point to the location of object data.
- An object can have multiple object references pointing to it.
- Or an object can have no object references pointing to it.
  - If so, the garbage collector will free the object's memory.

null Object References
- An object reference can point to no object. In that case, the object reference has the value *null*.
- Object references have the value *null* when they have been declared, but have not been used to instantiate an object.
- Attempting to use a *null* object reference causes a NullPointerException at run time.

Flow of Control
Four Types of Flow of Control
- Sequential Processing - Execute instructions in order
- Selection - Choose code to execute based on data value
- Looping or Iteration - Repeat operations for multiple data values
- Function Call - Jump to code in method, then return

boolean Data Type
- Type boolean is a primitive type consisting of just 2 values, the constants true and false
- We can declare variables of type boolean

Relational Operators
Are used in expressions of form:

<table>
<thead>
<tr>
<th>ExpressionA</th>
<th>Operator</th>
<th>ExpressionB</th>
</tr>
</thead>
<tbody>
<tr>
<td>temperature</td>
<td>&gt;</td>
<td>humidity</td>
</tr>
<tr>
<td>B<em>B - 4.0</em>A*C</td>
<td>&gt;=</td>
<td>0.0</td>
</tr>
<tr>
<td>one + two</td>
<td>&lt;</td>
<td>0</td>
</tr>
<tr>
<td>two * three</td>
<td>&lt;=</td>
<td>four</td>
</tr>
<tr>
<td>number</td>
<td>==</td>
<td>35</td>
</tr>
<tr>
<td>initial</td>
<td>!=</td>
<td>'Q'</td>
</tr>
</tbody>
</table>
Logical Operators

<table>
<thead>
<tr>
<th>Java exp.</th>
<th>Logical exp.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>! p</td>
<td>NOT p</td>
<td>! p is false if p is true. ! p is true if p is false.</td>
</tr>
<tr>
<td>p &amp;&amp; q</td>
<td>p AND q</td>
<td>p &amp;&amp; q is true if both p and q are true. It is false otherwise.</td>
</tr>
<tr>
<td>p</td>
<td></td>
<td>q</td>
</tr>
</tbody>
</table>

if-else (else can be omitted)

```java
if (Expression)
{   "if branch"
}
else
{   "else branch"
}
```

Examples
Assign value .25 to discountRate and assign value 10.00 to shipCost if purchase is over 100.00 Otherwise, assign value .15 to discountRate and assign value 5.00 to shipCost. Either way, calculate totalBill.

If taxCode has value 'T', increase price by adding taxRate times price to it.

Nested if statements

```java
if (Expression1)
   Statement1
else if (Expression2)
   Statement2
   ...
else if (ExpressionN)
   StatementN
else
   Statement N+1
```

EXACTLY 1 of these statements will be executed.

Output is “FAIL”!!??

```java
double average;
average = 100.0;
if (average >= 60.0)
   if (average < 70.0)
      System.out.println("Marginal PASS");
   else
      System.out.println("FAIL");
else
   System.out.println("FAIL");

WHY? The compiler ignores indentation and pairs the else with the second if
```

Floating-point values

- Do not compare floating-point values for equality
- Instead, compare them for near-equality

```java
double myNumber;
double yourNumber;

if (Math.abs(myNumber - yourNumber) < 0.00001)
   System.out.println("Close enough!");
```
