Introduction to Computer Science I

Math Class, Wrapper Classes, Conditionals

Janyl Jumadinova

October 3-5, 2016
Math Class

- Math plays a large role in computer programs.
- Because of this, there is an entire class (Math) that provides easy-to-use interfaces to many common mathematical methods.
- Unlike most other classes, the Math class is part of the java.lang package, which is imported automatically by the compiler when you compile a program. Therefore, you don’t need to do anything special in your program to have access to these methods.
Math class consists of:

- static methods, which are methods that don’t depend on the contents of an object.
- static fields, which are values that are usually defined to be public, final and static, meaning that anyone can access them outside the package. Since their values are final, that means that they are constant and can’t be changed.
Math.abs( x )  Absolute value of x
Math.ceil( x )  Rounds x to smallest integer not less than x
Math.exp( x )  Exponential method $e^x$
Math.floor( x ) Rounds x to the largest integer not greater than x
Math.max(x, y) Larger value of x and y
Math.min(x, y) Smaller value of x and y
Math.pow(x, y) x raised to the power y, $x^y$
Math.sqrt( x ) Square root of x
Math.PI The mathematical value Pi
Math.E Base value for natural logs

Math.abs( 23.7 ) is 23.7
Math.abs( -23.7 ) is 23.7
Math.ceil( 9.2 ) is 10.0
Math.ceil( -9.8 ) is -9.0
Math.exp( 1.0 ) is 2.71828
Math.exp( 2.0 ) is 7.38906
Math.floor( 9.2 ) is 9.0
Math.floor( -9.8 ) is -10.0
Math.max( 2.3, 12.7 ) is 12.7
Math.max( -2.3, -12.7 ) is -2.3
Math.min( 2.3, 12.7 ) is 2.3
Math.min( -2.3, -12.7 ) is -12.7
Math.pow( 2.0, 7.0 ) is 128.0
Math.pow( 9.0, 0.5 ) is 3.0
Math.sqrt( 900.0 ) is 30.0

3.14159265358979323846
2.7182818284590452354
May want to have an object hold a simple primitive value.
Wrapper Classes

- May want to have an object hold a simple primitive value.
- A wrapper class represents a particular primitive type.
Wrapper Classes

- May want to have an object hold a simple primitive value.
- A **wrapper class** represents a particular primitive type.
- For example, `Integer` represents a simple integer value.
- An object created from the `Integer` class stores a single `int` value.
Algorithms

Any problem can be solved by an algorithm

- **Algorithm** is a procedure for solving a problem in terms of the *actions* to be executed and the *order* in which those actions are to be executed.
Any problem can be solved by an algorithm

- **Algorithm** is a procedure for solving a problem in terms of the *actions* to be executed and the *order* in which those actions are to be executed.
Control Structures

Three Groups of Control Structures

1. **Sequential Structure**
   - It is just built into the language itself.
Control Structures

Three Groups of Control Structures

1. Sequential Structure

2. Selection Structures
   - if : single selection
   - if/else : double or multiple selection
   - switch : multiple selection
Three Groups of Control Structures

1. Sequential Structure
2. Selection Structures
3. Repetition Structure

while
do/while
for
Control Structures

- Java programs are built from only these seven control structures:
  - *three selection* (if, if/else, switch)
  - *three repetition* (while, do/while, for)
- You implement computer algorithms by stringing sequences of these seven control structures together.
if/else

- if only has a “do it or don’t do it” mentality – if the assertion is true, you do the associated action, if it’s false, you skip it.
- **if** only has a “do it or don’t do it” mentality – if the assertion is true, you do the associated action, if it's false, you skip it.

- The **if/else** structure gives more flexibility by providing something to do if the assertion is false – the “else” portion of the structure.

- **Nested if/else** structure strings together multiple if/else statements to handle a range of values.
Which of these code segments will determine a letter grade correctly based on variable grade?

- if ( grade < 60 )
  - System.out.println("F");
- else if ( grade >= 60 )
  - System.out.println("D");
- else if ( grade >= 70 )
  - System.out.println("C");
- else if ( grade >= 80 )
  - System.out.println("B");
- else
  - System.out.println("A");

- 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");
What if you wanted to do more than one thing in an `if` or an `if/else` action?

Need to use braces (`{` and `}`) to form a compound statement.
Compound Statements

```java
if ( grade >= 60 )
    System.out.println( "Passed." );
else
    System.out.println( "Failed." );
    System.out.println( "You must take this course again." );

if ( grade >= 60 )
{
    System.out.println( "Passed." );
}
else
{
    System.out.println( "Failed." );
    System.out.println( "You must take this course again." );
}
```
**if** and **if/else** tips to remember:

- They can be used to test ranges of values.
- In a nested **if/else** structure, an else always attempts to match up with the closest and most immediately unmatched preceding if statement.
- Always use compound statements with **if/else** structures to prevent problems down the road.
Logical Operators

- Using logical operators, we have a way to string multiple simple conditions together to help avoid/simplify nesting statements.
- These logical operators are based on the concept of Boolean logic or Boolean algebra.
Using logical operators, we have a way to string multiple simple conditions together to help avoid/simplify nesting statements.

These logical operators are based on the concept of Boolean logic or Boolean algebra.

These are the three logical operators in Java:

1. `&&` (logical AND)
2. `||` (logical OR)
3. `!` (logical NOT, or negation)
**Logical **and **Truth Table**

<table>
<thead>
<tr>
<th>expr1</th>
<th>expr2</th>
<th>expr1 &amp;&amp; expr2</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
## Logical or Truth Table

<p>| expr1 | expr2   | expr1 ||| expr2 |
|-------|---------|--------|--------|
| false | false   | false  |
| false | true    | true   |
| true  | false   | true   |
| true  | true    | true   |</p>
<table>
<thead>
<tr>
<th>expr1</th>
<th>!expr1</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>