Road Map

• Constants
• Basic Mathematical Operators
• Integer Division
• Operator Precedence
• Floating point types
• Other integer types

• Reading:
  – Liang 5: Chapter 2: 2.6, 2.7 (excluding 2.7.4), 2.8, 2.11
  – Liang 6: Chapter 2: 2.6, 2.7 (excluding 2.7.4), 2.8 plus part of 3.7
  – Liang 7: Chapter 2: 2.6, 2.7 (excluding 2.7.4), 2.8 plus part of 3.7
Review - True / False

• When System.out.println is called, it always begins printing at the beginning of a new line.
• All variables must be declared before they are used.
• All variables must be given a type when they are declared.
• Java considers the variables number and NuMBer identical.
• Declarations can appear anywhere in the body of the method main().
• A Java program that prints three lines of output must contain three System.out.println() statements.
1. Find the error in each statement:
   - System.println ("The value is " + value); /*assume there is an int variable called value that has been initialized*/
   - int num1, int num2, int num3;
   - int num#1, num#2, num#3;

2. What is the output for the following Java statements:
   int x;
   x = x+1;
   System.out.println (x);

3. What is the difference between a compile-time and run-time error?
Basic Mathematical Operators
### Basic Mathematical Operators

<table>
<thead>
<tr>
<th>Java Operation</th>
<th>Arithmetic Operator</th>
<th>Algebraic Expression</th>
<th>Java Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>a + b</td>
<td>a + b</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>a – b</td>
<td>a – b</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
<td>ab</td>
<td>a * b</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
<td>a / b</td>
<td>a / b</td>
</tr>
<tr>
<td>Modulus</td>
<td>%</td>
<td>a mod b</td>
<td>a % b</td>
</tr>
</tbody>
</table>

Each of the operators in the table are binary operators. A binary operator acts on two operands.
Integer Division - The Problem

• Suppose you have the following code:

      int x;
      x = 7 / 4;

• Using a calculator, the answer is 1.75.
• But x can only hold integer values. 1.75 is clearly not an integer value.
To understand the solution, you need to remember your 3rd Grade Math (really.)

\[
\begin{array}{c}
1 \\
4 \sqrt{7} \\
- 4 \\
- 4 \\
3
\end{array}
\]

The answer: 1 remainder 3

- \(7/4 = 1\) (Integer Division)
- \(7 \% 4 = 3\) (Modulus Division)
Example: Integer Division

// Integer and Modulus Division
public class DivMod
{
    public static void main( String args[] )
    {
        int x = 5, y = 10;
        System.out.println("5 / 10: " + x/y);
        System.out.println("5 % 10: " + x%y);
    }
}

5 / 10: 0
5 % 10: 5
Modulus Division (cont.)

• Second Example:

\[
5/10 = 0 \\
10 \div 5 = 0
\]

\[
5\%10 = 5
\]

• No matter what, your answers *must* be integers.
Odd / Even Numbers

• Modulus division can also be used to determine whether a number is odd or even.
• Just divide by 2. If the remainder (modulus) is 0, the number is even.
• Examples:
  – $10 \% 2 = 0$. Hence 10 is even.
  – $11 \% 2 = 1$. Hence 11 is odd.

Common Programming Error: Dividing by zero is normally undefined on computer systems and generally results in a fatal error.
Operator Precedence
Here’s another problem. What’s the answer to this?

\[ x = 7 + 3 \times 6; \]

Two Options (depending on the order of operations):
- Perform addition first: \( 7 + 3 = 10 \rightarrow 10 \times 6 = 60 \)
- Perform multiplication first: \( 3 \times 6 = 18 \rightarrow 7 + 18 = 25 \)

Which option is correct? Clearly, we cannot have this kind of ambiguity.
**Operator Precedence**

- Operator precedence represent rules for evaluating mathematical expressions.
- Every programming language has similar rules.

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Operation(s)</th>
<th>Order of evaluation (precedence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Parentheses</td>
<td>Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses “on the same level” (i.e., not nested), they are evaluated left to right.</td>
</tr>
<tr>
<td>* , / , or %</td>
<td>Multiplication</td>
<td>Evaluated second. If there are several, they are evaluated left to right.</td>
</tr>
<tr>
<td></td>
<td>Division</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Modulus</td>
<td></td>
</tr>
<tr>
<td>+ or −</td>
<td>Addition</td>
<td>Evaluated last. If there are several, they are evaluated left to right.</td>
</tr>
<tr>
<td></td>
<td>Subtraction</td>
<td></td>
</tr>
</tbody>
</table>
Operator Precedence

• Hence, option #2 is always correct (multiplication is performed first):

\[ x = 7 + 3 \times 6; \]

Evaluates to \( x = 7 + 18 = 25 \)

• Example: Find the average of three variables \( a, b \) and \( c \)
  
  • Do not use: \( a + b + c / 3 \)
  
  • Use: \( (a + b + c) / 3 \)
Parentheses

• Are your friends
• Are your really good friends
• Because with them you can ensure expressions are evaluated as you expect
• Can avoid mistakes with operator precedence (one less thing to think about)

  - e.g. \( y = m \times x + b \);
  \[ y = (m \times x) + b; \]
  - e.g. \( y = a \times b \times b + c \times b - d; \)
  \[ y = (((a \times b) \times b) + (c \times b)) - d; \]
Floating Point Data Types
**double Data Type**

- Data type that can hold numbers with fractional values
  - e.g. 3.14, 98.6

- Doubles can be used to represent many values:
  - Money (but see warning below)
  - distance
  - weight, etc.
// Double Example Program
public class Double
{
    public static void main( String args[] )
    {
        double var1, var2, var3, sum;

        var1 = 87.25;
        var2 = 92.50;
        var3 = 96.75;

        sum = var1 + var2 + var3;

        System.out.println("Sum: "+ sum);
    }
}

Sum: 276.5
# Numeric type ranges in Java

## Integers

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32768</td>
<td>32767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>-9,223,372,036,854,770,000</td>
<td>9,223,372,036,854,770,000</td>
</tr>
</tbody>
</table>

## Floating point values

<table>
<thead>
<tr>
<th>Type</th>
<th>Size</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>32</td>
<td>-3.40282347E+38</td>
<td>3.40282347E+38</td>
</tr>
<tr>
<td>double</td>
<td>64</td>
<td>-1.79769313486231570E+308</td>
<td>1.79769313486231570E+308</td>
</tr>
</tbody>
</table>
Example: Find an Average

• Suppose you want to determine a student’s average.

```java
int totalTests = 4;
double average = 90+92+95+100/ totalTests;
```
**Example: Find an Average**

- **Problem #1: Operator Precedence**
  - By rules of operator precedence, 100/4 is evaluated first. Hence, average is set to: 302.
  - To solve this problem, use ():

```java
int totalTests = 4;
double average = (90+92+95+100)/ totalTests;
```
Example: Find an Average

- Problem #2:
  - 90, 92, 95, 100 and 4 are all integers. Hence, this is integer division.
  - Integer division can result in data truncation (or loss of data.)
  - Hence, average is set to: 94.0, but the students real average is 94.25.
  - There are actually three ways to solve this problem.
Rules of Promotion

- **Promotion**: when mixing integers and doubles, all the values are promoted to doubles.
- In our average example, there are three ways to force promotion, and get the right answer of 94.25:

  1. change totalTests to a double:

     ```java
dauble totalTests = 4.0;
dauble average = (90+92+95+100)/ totalTests;
```
Rules of Promotion

2. Use a double literal for one of the values on the right hand side

   double average = (90.0+92+95+100) / 4;

3. Use a Cast Operator

   int totalTests = 4;
   double average = (90+92+95+100)/(double)totalTests;

   In this case, totalTests is explicitly cast to a double. And, because we have one double, everything else is promoted. Note that you can also use the (int) cast to cast a double to an integer.
constants

```java
final int TOTAL_GRADES = 4;
```

- Used to avoid “magic numbers” in code.
  - Only need to change in one place
    - If not, best case must search entire code, worst case you miss a few occurrences.
  - Choose meaningful names for your constants as you would for any other identifier

- Unlike variables, you cannot change the value of a symbolic constant.

- Should be in ALL CAPS (style).
more on casting

• In an assignment statement, you can assign a value of a “less expressive” type to a variable which is “more expressive”. For example:
  – int gets byte
  – double gets int
  – double gets float

• You must explicitly cast a value of a “more expressive” type to a variable which is “less expressive”. For example:
  – byte gets (byte) int
  – int gets (int) double
  – float gets (float) double
  – If you do not explicitly cast these values, you will have a syntax error in Java.
Warning about floating point values

• Floats may be represented differently from what you think by the computer
  – E.g. 1.9 to you may be 1.89999999999999999999999
  – 1.9 will not necessarily equal 1.9!

• In critical calculations for the same reason
  – E.g. .1 added 10 times often will not add up to 1
  – Use long integers instead and keep track of where your decimal point is (e.g. $1.75 should be stored as 175)