Unit 1: Primitive Types
Arithmetic Operations

Adapted from:
1) Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp
2) Runestone CSAwesome Curriculum

https://longbaonguyen.github.io
Expressions

• **expression**: A value or operation that computes a value.

• Examples:
  
  1 + 4 * 5
  
  (7 + 2) * 6 / 3
  
  42

  – The simplest expression is a *literal value*.
  
  – A complex expression can use operators and parentheses.
Arithmetic operators

- **operator**: Combines multiple values or expressions.
  - +  addition
  - -  subtraction (or negation)
  - *  multiplication
  - /  division
  - %  modulus (a.k.a. remainder)

- As a program runs, its expressions are *evaluated*.
  - 1 + 1 evaluates to 2
  - `System.out.println(3 * 4);` prints 12
  - How would we print the text `3 * 4`?
• When we divide integers, the quotient is also an integer.
  
  \[-14 \div 4 \text{ is } 3, \text{ not } 3.5\]

  
  \[
  \begin{array}{c}
  4 ) 14 \\
  12 \\
  2 \\
  \end{array}
  \quad \begin{array}{c}
  4 ) 45 \\
  40 \\
  5 \\
  \end{array}
  \quad \begin{array}{c}
  27 ) 1425 \\
  135 \\
  75 \\
  54 \\
  21 \\
  \end{array}
  \]

• More examples:
  
  \[-32 \div 5 \text{ is } 6\]
  \[-84 \div 10 \text{ is } 8\]
  \[-156 \div 100 \text{ is } 1\]

  – Dividing by 0 causes an error when your program runs. This error is also called an **ArithmeticException**.
Integer remainder with \%

- The \% operator computes the remainder from integer division.
  - $14 \% 4$ is 2
  - $218 \% 5$ is 3

\[
\begin{array}{c}
4 \quad \boxed{3} \\
4 \quad 14 \quad 12 \quad 15 \\
12 \quad 20 \\
2 \\
\end{array}
\begin{array}{c}
5 \quad \boxed{43} \\
5 \quad 218 \quad 18 \\
20 \\
18 \\
3 \\
\end{array}
\]

- Applications of \% operator:
  - Obtain last digit of a number: $230857 \% 10$ is 7
  - Obtain last 4 digits: $658236489 \% 10000$ is 6489
  - See whether a number is odd: $7 \% 2$ is 1, $42 \% 2$ is 0
public static void main(String[] args){
    System.out.println(45 % 6);
    System.out.println(2 % 2);
    System.out.println(8 % 10);
    System.out.println(11 % 0);
    System.out.println(-21 % 4); // probably not on AP
    System.out.println(21 % -4); // probably not on AP
}

Output:
3
0
8
ArithmeticException
-1
1
Expressions

Find the exact change for 137 cents using quarters, dimes, nickels and cents. Use the least number of coins.

How many quarters? \( \frac{137}{25} = 5 \) quarters (Integer Division!)

What’s leftover? \( 137 \% 25 = 12 \) cents

How many dimes? \( \frac{12}{10} = 1 \) dime

What’s leftover? \( 12 \% 10 = 2 \) cents

How many nickels? \( \frac{2}{5} = 0 \) nickels.

What’s leftover? \( 2 \% 5 = 2 \) cents.

How many pennies? \( \frac{2}{1} = 2 \) pennies
An important use of the % operator is to test for divisibility. For example, is a number even or odd? Is a number a multiple of 3?

// a number is even if it has no remainder
// when divided by 2.
if(number % 2 == 0){
    ...
}

// multiple of 3
if(number % 3 == 0){
    ...
}
• **precedence**: Order in which operators are evaluated.
  – Generally operators evaluate left-to-right.
    
    \[ 1 - 2 - 3 = (1 - 2) - 3 \text{ which is } -4 \]

  – But * / % have a higher level of precedence than + –
    
    \[ 1 + 3 * 4 \text{ is } 13 \]
    \[ 6 + 8 / 2 * 3 \]
    \[ 6 + 4 * 3 \]
    \[ 6 + 12 \text{ is } 18 \]

  – Parentheses can force a certain order of evaluation:
    
    \[ (1 + 3) * 4 \text{ is } 16 \]

  – Spacing does not affect order of evaluation
    
    \[ 1 + 3 * 4 - 2 \text{ is } 11 \]
Precedence examples

\[1 \times 2 + 3 \times 5 \mod 4\]

\[1 + 8 \mod 3 \times 2 - 9\]

\[1 + 2 \times 2 - 9\]

\[1 + 4 - 9\]

\[5 - 9\]

\[-4\]
Real numbers (type double)

- Examples: $6.022, -42.0, 2.143$
  - Placing .0 or . after an integer makes it a double.

- The operators $+ - * / \% ()$ all still work with double.
  - / produces an exact answer: $15.0 / 2.0$ is $7.5$
  - Precedence is the same: () before * / % before + -
Real number example

\[ 2.0 \times 2.4 + 2.25 \times 4.0 / 2.0 \]

\[ \\
 4.8 + 2.25 \times 4.0 / 2.0 \\
 4.8 + 9.0 / 2.0 \\
 4.8 + 4.5 \\
 9.3 \]
• When `int` and `double` are mixed, the result is a `double`.
  - `4.2 * 3` is `12.6`

• The conversion is per-operator, affecting only its operands.

\[
\begin{align*}
7 / 3 * 1.2 + 3 / 2 & = 3.4 \\
2 / 3 * 1.2 + 3 / 2 & = 2.4 \\
2.4 + 3 / 2 & = 3.4 \\
2.4 + 1 & = 3.4 \\
- 3 / 2 & is 1 above, not 1.5.
\end{align*}
\]

\[
\begin{align*}
2.0 + 10 / 3 * 2.5 - 6 / 4 & = 8.5 \\
2.0 + 3 * 2.5 - 6 / 4 & = 9.5 \\
2.0 + 7.5 - 6 / 4 & = 11.5 \\
2.0 + 7.5 - 1 & = 9.5 \\
- 3 / 2 & is 1 above, not 1.5.
\end{align*}
\]
Type casting

• **type cast:** A conversion from one type to another.
  – To promote an `int` into a `double` to get exact division from `/`
  – To truncate a `double` from a real number to an integer

• Syntax:

  \[(\text{type}) \ \text{expression}\]

Examples:

double result = (double) 19 / 5; // 3.8
int result2 = (int) result; // 3
int x = (int) Math.pow(10, 3); // 1000
More about type casting

• Type casting has high precedence and only casts the item immediately next to it.
  
  - double x = (double) 1 + 1 / 2; // 1.0
  - double y = 1 + (double) 1 / 2; // 1.5

• You can use parentheses to force evaluation order.
  - double average = (double) (a + b + c) / 3;
  - The code above cast the sum (a+b+c) into a double.

• A conversion to double can be achieved in other ways.
  - double average = 1.0 * (a + b + c) / 3;
public class Test{
    public static void main(String[] args){
        System.out.println(1 / 3);
        System.out.println(1.0 / 3);
        System.out.println(1 / 3.0);
        System.out.println((double) 1 / 3);
    }
}

0
0.3333333333333333
0.3333333333333333
0.3333333333333333
0.3333333333333333
public static void main(String[] args) {
    double x = 4 / 3;
    double y = (double) (125/10);
    double z = (double) 28 / 5;
    System.out.println(x + " " + y + " " + z);
}

Output:
1.0 12.0 5.6
• casting can be used to round a number to its nearest integer.

```java
double number = 7.0 / 3;
// round a positive number to its nearest integer
int nearestInt = (int)(number + 0.5);
double negNumber = -20.0 / 3;
// round a negative number to its nearest integer
int nearestNegInt = (int)(negNumber – 0.5);
```

What is the value of nearestInt and nearestNegInt?
Answer: 2 and -7
**Increment and decrement**

*shortcuts to increase or decrease a variable's value by 1*

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>variable++;</code></td>
<td><code>variable = variable + 1;</code></td>
</tr>
<tr>
<td><code>variable--;</code></td>
<td><code>variable = variable - 1;</code></td>
</tr>
</tbody>
</table>

```java
int x = 2;
x++;  // x = x + 1;
// x now stores 3

double gpa = 2.5;
gpa--;  // gpa = gpa - 1;
// gpa now stores 1.5
```
## Modify-and-assign

### Shortcuts to modify a variable's value

<table>
<thead>
<tr>
<th>Shorthand</th>
<th>Equivalent longer version</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable += value;</td>
<td>variable = variable + value;</td>
</tr>
<tr>
<td>variable -= value;</td>
<td>variable = variable - value;</td>
</tr>
<tr>
<td>variable *= value;</td>
<td>variable = variable * value;</td>
</tr>
<tr>
<td>variable /= value;</td>
<td>variable = variable / value;</td>
</tr>
<tr>
<td>variable %= value;</td>
<td>variable = variable % value;</td>
</tr>
</tbody>
</table>

```plaintext
x += 3;       // x = x + 3;
gpa -= 0.5;   // gpa = gpa - 0.5;
number *= 2;  // number = number * 2;
```
What are the values of x, y and z after tracing through the following code?

```java
int x = 0;
int y = 5;
int z = 1;
x++;
x++;
y -= 3;
z = x + z;
x = y * z;
y %= 2;
z--;
```

**Answer:** x = 4, y = 0, z = 1
• Let \{a_1, a_2, a_3, \ldots, a_n\} be a list of \(n\) real numbers.

• The average of the list is \(\text{ave} = \frac{a_1 + a_2 + \ldots + a_n}{n}\).

• The variance of the list =

\[
\left[ (a_1 - \text{ave})^2 + (a_2 - \text{ave})^2 + \ldots + (a_n - \text{ave})^2 \right] / n.
\]

• The standard deviation of the list = the square root of the variance of the list.

HINT: Use Math.sqrt() for square root: Math.sqrt(9) is 3.0
For example, if the list is \{78,80,77\}.

Average = 78.33333333333333

Variance = 1.5555555555555556
Standard deviation = 1.247219128924647
Create a new repl on repl.it and follow the comments below to write a program that compute some statistics.

```java
public class Statistics {
    public static void main(String[] args) {
        // 1. Declare 3 int variables for grades and initialize them to 3 values
        // 2. Declare an int variable for the sum of the grades
        // 3. Declare a double variable for the average of the grades
        // 4. Write a formula to calculate the sum of the 3 grades
        // 5. Write a formula to calculate the average of the 3 grades from the
            //    sum using division and type casting.
        // 6. Print out the average
        // 7. Declare a double variable and calculate the variance
        // 8. Declare a double variable to compute the standard deviation.
        // 9. Print out the variance and standard deviation.
    }
}
```
Lab 2

Use the following template (or something similar) to write a program that gives exact change with the least number of coins for a given number of cents. **Use intermediate variables to help your calculation.**

```java
public static void main(String[] args){
    int totalCents = 137;  //137 can be any number
    .....  
    // your code here.
}
```

Output: 5 quarters, 1 dimes, 0 nickels, 2 pennies.
1) Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum:
https://runestone.academy/runestone/books/published/csawesome/index.html

For more tutorials/lecture notes in Java, Python, game programming, artificial intelligence with neural networks:

https://longbaonguyen.github.io