Announcements

• Assignment 1 is due Apr 10, 11:59 PM
• Assignment 2 will be released Apr 10
  – Due Apr 17, 11:59 PM
Numbers and mathematics

• Numerical data types (e.g., an integer)
• Numeric operations (e.g., addition)
• Mathematical functions (e.g., cosine)
• Reading numbers from the console
Data types

• Java is a strongly typed language
  – Programmers must explicitly identify the type of every variable, method, and object
# Numerical data types

<table>
<thead>
<tr>
<th>Name</th>
<th>Range</th>
<th>Storage Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>(-2^7 \text{ to } 2^7 - 1 (-128 \text{ to } 127))</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>short</td>
<td>(-2^{15} \text{ to } 2^{15} - 1 (-32768 \text{ to } 32767))</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int</td>
<td>(-2^{31} \text{ to } 2^{31} - 1 (-2147483648 \text{ to } 2147483647))</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>long</td>
<td>(-2^{63} \text{ to } 2^{63} - 1) (i.e., (-9223372036854775808 \text{ to } 9223372036854775807))</td>
<td>64-bit signed</td>
</tr>
</tbody>
</table>
| float | Negative range: -3.4028235E+38 to -1.4E-45  
Positive range: 1.4E-45 to 3.4028235E+38 | 32-bit IEEE 754 |
| double | Negative range: -1.7976931348623157E+308 to -4.9E-324  
Positive range: 4.9E-324 to 1.7976931348623157E+308 | 64-bit IEEE 754 |
Number literals

• A literal is a constant value that appears directly in the program

```java
int i = 34;
long x = 1000000;
double d = 5.0 + 1.0;
```

34, 1000000, 5.0, and 1.0 are literals
Integer literals

• An integer literal can be assigned to an integer variable as long as it can fit into the variable

• A compilation error will occur if the literal is too large for the variable to hold
  – For example, the statement `byte b = 1000` would cause a compilation error, because `1000` cannot be stored in a variable of the `byte` type

• An integer literal is assumed to be of the `int` type, whose value is between \(-2^{31}\) (equals `-2147483648`) to \(2^{31}-1\) (equals 2147483647)

• To denote an integer literal of the `long` type, append it with the letter `L` or `l`
  – `L` is preferred because `1` (lowercase `L`) can easily be confused with `1` (the digit one)
Floating-point literals

• Floating-point literals are written with a decimal point
• By default, a floating-point literal is treated as a double type value
  – Example: 5.0 is considered a double value, not a float value
• You can make a number a float by appending the letter f or F, and make a number a double by appending the letter d or D
  – Example: you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number
Scientific notation

• Floating-point literals can also be specified in scientific notation
  – Example: $1.23456e+2$ (same as $1.23456e2$) is equivalent to $123.456$, and $1.23456e-2$ is equivalent to $0.0123456$

• E or e represents an exponent
## Numeric operations

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>34.0 - 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>
**double vs float**

- The double type values are more accurate than the float type values
  - For example,

  ```java
  System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
  displays 1.0 / 3.0 is 0.3333333333333333
  16 digits
  ```

  ```java
  System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
  displays 1.0F / 3.0F is 0.3333334
  7 digits
  ```
Floating-point accuracy

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy.
  - For example,
    ```java
    System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
    displays 0.5000000000000001, not 0.5, and
    System.out.println(1.0 - 0.9);
    displays 0.09999999999999998, not 0.1
    ```
- Integers are stored precisely
  - Calculations with integers yield a precise integer result.
Integer division

- Warning: resulting fractional part (i.e., values after the decimal point) are **truncated, not rounded**
  - Example: 5 / 2 yields an integer 2
Remainder operator

• Example: an even number % 2 is always 0 and an odd number % 2 is always 1
  – You can use this property to determine whether a number is even or odd

• Example: If today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression.

Saturday is the 6th day in a week

\[(6 + 10) \mod 7 = 2\]

After 10 days

A week has 7 days

The 2nd day in a week is Tuesday
Augmented assignment operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+=</code></td>
<td>Addition assignment</td>
<td><code>i += 8</code></td>
<td><code>i = i + 8</code></td>
</tr>
<tr>
<td><code>-=</code></td>
<td>Subtraction assignment</td>
<td><code>i -= 8</code></td>
<td><code>i = i - 8</code></td>
</tr>
<tr>
<td><code>*=</code></td>
<td>Multiplication assignment</td>
<td><code>i *= 8</code></td>
<td><code>i = i * 8</code></td>
</tr>
<tr>
<td><code>/=</code></td>
<td>Division assignment</td>
<td><code>i /= 8</code></td>
<td><code>i = i / 8</code></td>
</tr>
<tr>
<td><code>%=</code></td>
<td>Remainder assignment</td>
<td><code>i %= 8</code></td>
<td><code>i = i % 8</code></td>
</tr>
</tbody>
</table>
# Increment and decrement operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
<th>Example (assume $i = 1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>++var</code></td>
<td>preincrement</td>
<td>Increment <code>var</code> by 1, and use the new <code>var</code> value in the statement</td>
<td>int $j = ++i$;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// $j$ is 2, $i$ is 2</td>
</tr>
<tr>
<td><code>var++</code></td>
<td>postincrement</td>
<td>Increment <code>var</code> by 1, but use the original <code>var</code> value in the statement</td>
<td>int $j = i++$;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// $j$ is 1, $i$ is 2</td>
</tr>
<tr>
<td><code>--var</code></td>
<td>predecrement</td>
<td>Decrement <code>var</code> by 1, and use the new <code>var</code> value in the statement</td>
<td>int $j = --i$;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// $j$ is 0, $i$ is 0</td>
</tr>
<tr>
<td><code>var--</code></td>
<td>postdecrement</td>
<td>Decrement <code>var</code> by 1, and use the original <code>var</code> value in the statement</td>
<td>int $j = i--$;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>// $j$ is 1, $i$ is 0</td>
</tr>
</tbody>
</table>
Conversion rules

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.
Type casting

Implicit casting
  double d = 3; (type widening)

Explicit casting
  int i = (int)3.0; (type narrowing)
  int i = (int)3.9; (fraction part is truncated, not rounded!)

range increases

byte, short, int, long, float, double
Operator precedence

- ( ), var++, var--
- ++var, --var, +, - (unary plus and minus), ! (not)
- (type) casting
- *, /, % (multiplication, division, and remainder)
- +, - (binary addition and subtraction)
- <, <=, >, >= (relational operators)
- ==, != (equality)
- ^ (exclusive or)
- && (and)
- || (or)
- =, +=, -=, *=, /=, %= (assignment operators)

Relational and logical operators will be covered next lecture
Operator associativity

• When two operators with the same precedence are evaluated, the associativity of the operators determines the order of evaluation.

• All binary operators except assignment operators are left-associative
  \[ a - b + c - d \] is equivalent to \[ ((a - b) + c) - d \]

• Assignment operators are right-associative
  \[ a = b += c = 5 \] is equivalent to \[ a = (b += (c = 5)) \]
Operator precedence and associativity

• The expression in the parentheses is evaluated first
  – Parentheses can be nested, in which case the expression in the inner parentheses is executed first
• When evaluating an expression without parentheses, the operators are applied according to the precedence rule and the associativity rule
• If operators with the same precedence are next to each other, their associativity determines the order of evaluation
Reading numbers from the console

• Create a Scanner object
  Scanner input = new Scanner(System.in);
  – Java 8 API documentation
    • https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
  – Java 11 API documentation
    • https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html
Explicit import and implicit Import

• At top of source file

```java
import java.util.Scanner; // Explicit Import

import java.util.*; // Implicit import
```
# Reading numbers from the console

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nextByte()</code></td>
<td>reads an integer of the <strong>byte</strong> type.</td>
</tr>
<tr>
<td><code>nextShort()</code></td>
<td>reads an integer of the <strong>short</strong> type.</td>
</tr>
<tr>
<td><code>nextInt()</code></td>
<td>reads an integer of the <strong>int</strong> type.</td>
</tr>
<tr>
<td><code>nextLong()</code></td>
<td>reads an integer of the <strong>long</strong> type.</td>
</tr>
<tr>
<td><code>nextFloat()</code></td>
<td>reads a number of the <strong>float</strong> type.</td>
</tr>
<tr>
<td><code>nextDouble()</code></td>
<td>reads a number of the <strong>double</strong> type.</td>
</tr>
</tbody>
</table>
Reading numbers from the console

• Example: use the method nextDouble() to obtain to a double value

```java
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```
Mathematical functions

- Java provides many useful methods in the Math class for performing common mathematical functions
  - Java 8 API documentation
    - [https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html](https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html)
  - Java 11 API documentation
Mathematical functions

• Constants
  Math.PI
  Math.E

• Math class methods
  – Trigonometric methods
  – Exponent methods
  – Rounding methods
  – min, max, abs, and random methods
Trigonometric methods

Math.toDegrees(radians)
Math.toRadians(degrees)
Math.sin(radians)
Math.cos(radians)
Math.tan(radians)
Math.acos(a)
Math.asin(a)
Math.atan(a)

Result is in radians
Exponent methods

Math.exp(a) \quad e^a
Math.log(a) \quad \log_e(a)
Math.log10(a) \quad \log_{10}(a)
Math.pow(a, b) \quad a^b
Math.sqrt(a) \quad \sqrt{a}
Rounding methods

Math.ceil(x)

Math.floor(x)

Math.rint(x)

• If you want to return an integer type, then

    int Math.round(float x)
    • Returns (int)Math.floor(x + 0.5f)

    long Math.round(double x)
    • Returns (long)Math.floor(x + 0.5)
min, max, abs, and random methods

Math.min(a, b)
Math.max(a, b)
Math.abs(a)
Math.random()

– Returns a random double value in the range [0.0, 1.0)
Next Lecture

• Characters and strings