class Lecture2 {

    "Data types, Variables, and Operators"

}

// Keywords:
byte, short, int, long, char, float, double, boolean, true, false, import, new
Example

Given the radius of a circle, say 10, determine the area.

Recall that a program comprises data and algorithms.

- How to store the data?
  → variables, data types

- How to compute the area?
  → arithmetic operators

- How to show the result?
  → System.out.println()
public class ComputeArea {
    public static void main(String[] args) {
        // input
        int r = 10;
        // algorithm
        double area = r * r * 3.14;
        // output
        System.out.println(area);
    }
}

- The type int and double are two of primitive data types.
- We use two variables r and area.
Variable $\approx$ Box
Variable Declaration

• You give a name for the variable, say x.
• Additionally, you need to assign a type for the variable.
• For example,

```java
... int x; // x is a variable declared an integer type.
...
```

• Variable declaration tells the compiler to allocate appropriate memory space for the variable based on its data type.\(^1\)
• It is worth to mention that, the data type determines the size, which is measured in bytes\(^2\).

---

\(^1\)Actually, all declared variables are created at the compile time.
\(^2\)1 byte = 8 bits; bit = binary digit.
Naming Rules

• Identifiers are the names that identify the elements such as variables, methods, and classes in the program.

• The naming rule excludes the following situations:
  • cannot start with a digit
  • cannot be any reserved word
  • cannot include any blank between letters
  • cannot contain +, −, *, / and %

• Note that Java is case sensitive.

---

3 See the next page.

4 The letter A and a are different.
<table>
<thead>
<tr>
<th>abstract</th>
<th>double</th>
<th>int</th>
<th>super</th>
</tr>
</thead>
<tbody>
<tr>
<td>assert</td>
<td>else</td>
<td>interface</td>
<td>switch</td>
</tr>
<tr>
<td>boolean</td>
<td>enum</td>
<td>long</td>
<td>synchronized</td>
</tr>
<tr>
<td>break</td>
<td>extends</td>
<td>native</td>
<td>this</td>
</tr>
<tr>
<td>byte</td>
<td>final</td>
<td>new</td>
<td>throw</td>
</tr>
<tr>
<td>case</td>
<td>finally</td>
<td>package</td>
<td>throws</td>
</tr>
<tr>
<td>catch</td>
<td>float</td>
<td>private</td>
<td>transient</td>
</tr>
<tr>
<td>char</td>
<td>for</td>
<td>protected</td>
<td>try</td>
</tr>
<tr>
<td>class</td>
<td>goto</td>
<td>public</td>
<td>void</td>
</tr>
<tr>
<td>const</td>
<td>if</td>
<td>return</td>
<td>volatile</td>
</tr>
<tr>
<td>continue</td>
<td>implements</td>
<td>short</td>
<td>while</td>
</tr>
<tr>
<td>default</td>
<td>import</td>
<td>static</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td>instanceof</td>
<td>strictfp*</td>
<td></td>
</tr>
</tbody>
</table>

---

5See Appendix A in YDL, p. 1253.
### Variable as Alias of Memory Address

<table>
<thead>
<tr>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000abc26</td>
</tr>
<tr>
<td>0x000abc27</td>
</tr>
<tr>
<td>0x000abc28</td>
</tr>
<tr>
<td>0x000abc29</td>
</tr>
</tbody>
</table>

- The number 0x000abc26 stands for one memory address in hexadecimal (0-9, and a-f).
- The variable \( x \) itself refers to 0x000abc26 in the program after compilation.

---

6 See https://en.wikipedia.org/wiki/Hexadecimal
Data Types

- Java is a strongly typed\textsuperscript{7} programming language.
- Every variable has a type.
- Also, every (mathematical) expression has a type.
- There are two categories of data types: \texttt{primitive} data types, and \texttt{reference} data types.

\textsuperscript{7}You cannot change the type of the variable after declaration.
Primitive Data Types

- Primitive Data Type
  - boolean
  - Numeric Type
    - Integral Type
      - byte
      - short
      - int
      - long
      - char
    - Floating-Point Type
      - float
      - double

\(^8\)See Figure 3-4 in Sharan, p. 67.
Integers

- The most commonly used integer type is `int`.
- If the integer values are larger than its feasible range, then an overflow occurs.

<table>
<thead>
<tr>
<th>Name</th>
<th>Width</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>64</td>
<td>-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>int</td>
<td>32</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>short</td>
<td>16</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td>byte</td>
<td>8</td>
<td>-128 to 127</td>
</tr>
</tbody>
</table>
Floating Points

- Floating points are used when evaluating expressions that require fractional precision.
  - For example, \( \sin() \), \( \cos() \), and \( \sqrt{} \).
- The performance for the double values is actually faster than that for float values on modern processors that have been optimized for high-speed mathematical calculations.
- Be aware that floating-point arithmetic can only approximate real arithmetic.\(^9\)  (Why?)


<table>
<thead>
<tr>
<th>Name</th>
<th>Width in Bits</th>
<th>Approximate Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>64</td>
<td>4.9e−324 to 1.8e+308</td>
</tr>
<tr>
<td>float</td>
<td>32</td>
<td>1.4e−045 to 3.4e+038</td>
</tr>
</tbody>
</table>
Example: $0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 = 0$?

```java
public class FloatingPointsDemo {
    public static void main(String args[]) {
        System.out.println(0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);
    }
}
```

- The result is surprising. (Why?)
- You may try this [decimal-binary converter](#).
- This issue is not only associated with decimal numbers, but also big integers.\(^\text{10}\)
- So double values are not reliable if the program runs for high-precision calculation.

\(^{10}\)Thanks to a lively discussion on June 26, 2016.
Example: Loss of Significance

• For example,

```java
... System.out.println(3.14 + 1e20 - 1e20); // output ? System.out.println(3.14 + (1e20 - 1e20)); // output ? ...
```

• Can you explain why?
IEEE Floating-Point Representation\textsuperscript{11}

\[ x = (-1)^s \times M \times 2^E \]

- The sign $s$ determines whether the number is negative ($s = 1$) or positive ($s = 0$).
- The significand $M$ is a fractional binary number that ranges either between 1 and $2 - \epsilon$, or between 0 and $1 - \epsilon$.
- The exponent $E$ weights the value by a (possibly negative) power of 2.

\textsuperscript{11}William Kahan (1985); Aka IEEE754.
That is why we call a double value.

See Figure 2-31 in Byrant, p. 104.
Assignments

- An assignment statement designates a value to the variable.

```java
int x; // make a variable declaration
...
x = 1; // assign 1 to x
```

- The equal sign (=) is used as the assignment operator.
  - For example, is the expression `x = x + 1` correct?
  - Direction: from the right-hand side to the left-hand side

- To assign a value to a variable, you must place the variable name to the left of the assignment operator.\(^{13}\)
  - For example, `1 = x` is wrong.
  - `1` cannot be resolved to a memory space.

\(^{13}\)x can be a l-value and r-value, but 1 and other numbers can be only r-value but not l-value. See Value.
Two “Before” Rules

- Every variable has a scope.
  - The scope of a variable is the range of the program where the variable can be referenced.\(^\text{14}\)

- A variable must be declared before it can be assigned a value.
  - In practice, do not declare the variable until you need it.

- A declared variable must be assigned a value before it can be used.\(^\text{15}\)

\(^{14}\)The detail of variable scope is introduced later.

\(^{15}\)In symbolic programming, such as Mathematica and Maple, a variable can be manipulated without assigning a value. For example, \(x + x\) returns \(2x\).
### Arithmetic Operators

<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>

- Note that the operator depends on the operands involved.

---

\(^{16}\)See Table 2-3 in YDL, p. 46.
Tricky Pitfalls

• Can you explain this result?

```java
... double x = 1 / 2;
System.out.println(x); // output?
...
```

• Revisit \(0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 = 0\)\(^{17}\)

```java
... System.out.println(1 / 2 - 1 / 10 - 1 / 10 - 1 / 10 - 1 / 10); // output 0; however, this is not the real solution to the original problem.
...
```

\(^{17}\)Thanks to a lively discussion on on June 7, 2016.
Type Conversion and Compatibility

• If a type is **compatible** to another, then the compiler will perform the conversion **implicitly**.
  • For example, the integer 1 is compatible to a **double** value 1.0.
• However, there is no automatic conversion from **double** to **int**. (Why?)
• To do so, you must use a **cast**, which performs an **explicit** conversion for compilation.
• Similarly, a **long** value is not compatible to **int**.
Casting

```java
... 
int x = 1;
double y = x; // compatible; implicit conversion
x = y; // incompatible; need an explicit conversion by casting
x = (int) y; // succeed!!
... 
```

- Note that the Java compiler does only **type-checking** but no real execution before compilation.
- In other words, the values of `x` and `y` are unknown until they are really executed.
Type Conversion and Compatibility (concluded)

- small-size types $\rightarrow$ large-size types
- small-size types $\leftrightarrow$ large-size types (need a cast)
- simple types $\rightarrow$ complicated types
- simple types $\leftrightarrow$ complicated types (need a cast)
A character stored by the machine is represented by a sequence of 0’s and 1’s.
- For example, ASCII code. (See the next page.)
- The `char` type is a 16-bit unsigned primitive data type.\(^\text{18}\)

\(^{18}\)Java uses **Unicode** to represent characters. Unicode defines a fully international character set that can represent all of the characters found in all human languages.
# ASCII (7-bit version)

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec Char</th>
<th>Hex</th>
<th>Dec Char</th>
<th>Hex</th>
<th>Dec Char</th>
<th>Hex</th>
<th>Dec Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>NULL</td>
<td>0x20</td>
<td>Space</td>
<td>0x40</td>
<td>@</td>
<td>0x60</td>
<td>~</td>
</tr>
<tr>
<td>0x01</td>
<td>SOH</td>
<td>0x21</td>
<td>!</td>
<td>0x41</td>
<td>A</td>
<td>0x61</td>
<td>a</td>
</tr>
<tr>
<td>0x02</td>
<td>STX</td>
<td>0x22</td>
<td>&quot;</td>
<td>0x42</td>
<td>B</td>
<td>0x62</td>
<td>b</td>
</tr>
<tr>
<td>0x03</td>
<td>ETX</td>
<td>0x23</td>
<td>#</td>
<td>0x43</td>
<td>C</td>
<td>0x63</td>
<td>c</td>
</tr>
<tr>
<td>0x04</td>
<td>EOT</td>
<td>0x24</td>
<td>$</td>
<td>0x44</td>
<td>D</td>
<td>0x64</td>
<td>d</td>
</tr>
<tr>
<td>0x05</td>
<td>ENQ</td>
<td>0x25</td>
<td>%</td>
<td>0x45</td>
<td>E</td>
<td>0x65</td>
<td>e</td>
</tr>
<tr>
<td>0x06</td>
<td>ACK</td>
<td>0x26</td>
<td>&amp;</td>
<td>0x46</td>
<td>F</td>
<td>0x66</td>
<td>f</td>
</tr>
<tr>
<td>0x07</td>
<td>BEL</td>
<td>0x27</td>
<td>'</td>
<td>0x47</td>
<td>G</td>
<td>0x67</td>
<td>g</td>
</tr>
<tr>
<td>0x08</td>
<td>BS</td>
<td>0x28</td>
<td>(</td>
<td>0x48</td>
<td>H</td>
<td>0x68</td>
<td>h</td>
</tr>
<tr>
<td>0x09</td>
<td>HT</td>
<td>0x29</td>
<td>)</td>
<td>0x49</td>
<td>I</td>
<td>0x69</td>
<td>i</td>
</tr>
<tr>
<td>0x0A</td>
<td>LF</td>
<td>0x2A</td>
<td>*</td>
<td>0x4A</td>
<td>J</td>
<td>0x6A</td>
<td>j</td>
</tr>
<tr>
<td>0x0B</td>
<td>VT</td>
<td>0x2B</td>
<td>+</td>
<td>0x4B</td>
<td>K</td>
<td>0x6B</td>
<td>k</td>
</tr>
<tr>
<td>0x0C</td>
<td>FF</td>
<td>0x2C</td>
<td>,</td>
<td>0x4C</td>
<td>L</td>
<td>0x6C</td>
<td>l</td>
</tr>
<tr>
<td>0x0D</td>
<td>CR</td>
<td>0x2D</td>
<td>-</td>
<td>0x4D</td>
<td>M</td>
<td>0x6D</td>
<td>m</td>
</tr>
<tr>
<td>0x0E</td>
<td>SO</td>
<td>0x2E</td>
<td>.</td>
<td>0x4E</td>
<td>N</td>
<td>0x6E</td>
<td>n</td>
</tr>
<tr>
<td>0x0F</td>
<td>SI</td>
<td>0x2F</td>
<td>/</td>
<td>0x4F</td>
<td>O</td>
<td>0x6F</td>
<td>o</td>
</tr>
<tr>
<td>0x10</td>
<td>DLE</td>
<td>0x30</td>
<td>0</td>
<td>0x50</td>
<td>P</td>
<td>0x70</td>
<td>p</td>
</tr>
<tr>
<td>0x11</td>
<td>DC1</td>
<td>0x31</td>
<td>1</td>
<td>0x51</td>
<td>Q</td>
<td>0x71</td>
<td>q</td>
</tr>
<tr>
<td>0x12</td>
<td>DC2</td>
<td>0x32</td>
<td>2</td>
<td>0x52</td>
<td>R</td>
<td>0x72</td>
<td>r</td>
</tr>
<tr>
<td>0x13</td>
<td>DC3</td>
<td>0x33</td>
<td>3</td>
<td>0x53</td>
<td>S</td>
<td>0x73</td>
<td>s</td>
</tr>
<tr>
<td>0x14</td>
<td>DC4</td>
<td>0x34</td>
<td>4</td>
<td>0x54</td>
<td>T</td>
<td>0x74</td>
<td>t</td>
</tr>
<tr>
<td>0x15</td>
<td>NAK</td>
<td>0x35</td>
<td>5</td>
<td>0x55</td>
<td>U</td>
<td>0x75</td>
<td>u</td>
</tr>
<tr>
<td>0x16</td>
<td>SYN</td>
<td>0x36</td>
<td>6</td>
<td>0x56</td>
<td>V</td>
<td>0x76</td>
<td>v</td>
</tr>
<tr>
<td>0x17</td>
<td>ETB</td>
<td>0x37</td>
<td>7</td>
<td>0x57</td>
<td>W</td>
<td>0x77</td>
<td>w</td>
</tr>
<tr>
<td>0x18</td>
<td>CAN</td>
<td>0x38</td>
<td>8</td>
<td>0x58</td>
<td>X</td>
<td>0x78</td>
<td>x</td>
</tr>
<tr>
<td>0x19</td>
<td>EM</td>
<td>0x39</td>
<td>9</td>
<td>0x59</td>
<td>Y</td>
<td>0x79</td>
<td>y</td>
</tr>
<tr>
<td>0x1A</td>
<td>SUB</td>
<td>0x3A</td>
<td>:</td>
<td>0x5A</td>
<td>Z</td>
<td>0x7A</td>
<td>z</td>
</tr>
<tr>
<td>0x1B</td>
<td>ESC</td>
<td>0x3B</td>
<td>;</td>
<td>0x5B</td>
<td>[</td>
<td>0x7B</td>
<td>{</td>
</tr>
<tr>
<td>0x1C</td>
<td>FS</td>
<td>0x3C</td>
<td>&lt;</td>
<td>0x5C</td>
<td>\</td>
<td>0x7C</td>
<td></td>
</tr>
<tr>
<td>0x1D</td>
<td>GS</td>
<td>0x3D</td>
<td>=</td>
<td>0x5D</td>
<td>}</td>
<td>0x7D</td>
<td>}</td>
</tr>
<tr>
<td>0x1E</td>
<td>RS</td>
<td>0x3E</td>
<td>&gt;</td>
<td>0x5E</td>
<td>^</td>
<td>0x7E</td>
<td>~</td>
</tr>
<tr>
<td>0x1F</td>
<td>US</td>
<td>0x3F</td>
<td>?</td>
<td>0x5F</td>
<td>_</td>
<td>0x7F</td>
<td>DEL</td>
</tr>
</tbody>
</table>
Characters can also be used as an integer type on which you can perform arithmetic operations.\(^{19}\)

For example,

```java
...  // A single-quoted value is the char type.
char x = 'a';
System.out.println(x + 1); // output 98!!
System.out.println((char)(x + 1)); // output 'b'
...
```

Notice that a double-quoted string is a `String` object, which can have more `char` values.

---

Boolean Values

- The program is supposed to do decision making by itself, for example, Google Driverless Car.\(^{20}\)
- To do this, Java has the boolean-type flow controls (selections and iterations).
- Only two possible values, true and false.
- Note that a boolean value cannot be cast into a value of another type, nor can a value of another type be cast into a boolean value.

\(^{20}\)See https://www.google.com/selfdrivingcar/
Rational Operators

<table>
<thead>
<tr>
<th>Java Operator</th>
<th>Mathematics Symbol</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
<td>not equal to</td>
</tr>
</tbody>
</table>

- These operators take two operands.
- Rational expressions return a `boolean` value.
- Note that the equality comparison operator is double equality sign (==), not single equality sign (=).

---

21 See Table 3-1 in YDL, p. 82.
Example

```java
...
    int x = 2;
    boolean a = x > 1;
    boolean b = x < 1;
    boolean c = x == 1;
    boolean d = x != 1;
    boolean e = 1 < x < 3; // sorry?
...
```

- Be aware that e is logically correct but syntactically wrong.
- Usually, the boolean expression consists of a combination of rational expressions.
  - For example, $1 < x < 3$ should be $(1 < x) \&\& (x < 3)$, where \&\& refers to the AND operator.