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,

```
...  
1  int x; // x is a variable declared an interger type.
2  ...
3  ...
```

• 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.
### Reserved Words

<table>
<thead>
<tr>
<th>reserved_word</th>
<th>reserved_word</th>
<th>reserved_word</th>
<th>reserved_word</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
<td>double</td>
<td>int</td>
<td>super</td>
</tr>
<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

- The number 0x000abc26 stands for one memory address in hexadecimal (0-9, and a-f).\(^6\)
- 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 static typed\(^7\) programming language.
• Every variable has a type.
• Also, every (mathematical) expression has a type.
• There are two categories of data types: primitive data types, and reference data types.

\(^7\)You cannot change the type of the variable after declaration.
Primitive Data Types

- boolean

Numeric Type

- Integral Type
  - byte
  - short
  - int
  - long
  - char

- Floating-Point Type
  - float
  - double

---

See Figure 3-4 in Sharan, p. 67.
Integers

<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>

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

- Floats 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?)

---

Example: 0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 = 0?

```java
public class FloatsDemo {
    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](https://finance.technews.tw/2017/01/10/largan-stock-trouble/).
- This issue occurs not only in decimal numbers, but also big integers represented in floats.\(^\text{10}\)
- So the floats are not reliable unless the algorithm is designed elaborately for numerical errors.\(^\text{11}\)

\(^\text{10}\)Thanks to a lively discussion on June 26, 2016.

\(^\text{11}\)See [https://finance.technews.tw/2017/01/10/largan-stock-trouble/](https://finance.technews.tw/2017/01/10/largan-stock-trouble/).
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{12}

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

- 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{12}William Kahan (1985); Aka IEEE754.
• That is why we call a **double** value.

---

13 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.\(^{14}\)
  - For example, \( 1 = x \) is wrong.
  - \( 1 \) cannot be resolved to a memory space.

\(^{14}\) \( x \) 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.\(^{15}\)
- 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.\(^{16}\)

\(^{15}\)The detail of variable scope is introduced later.

\(^{16}\)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.*

---

[17] 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 $0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1 = 0.18$

```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.
...
```

\[^{18}\text{Thanks to a lively discussion on on June 7, 2016.}\]
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.\(^\text{19}\)

\(^{19}\)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.
<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>0</td>
<td>NULL</td>
<td>null</td>
<td>0x20</td>
<td>32</td>
<td>Space</td>
<td>0x40</td>
</tr>
<tr>
<td>0x01</td>
<td>1</td>
<td>SOH</td>
<td>Start of heading</td>
<td>0x21</td>
<td>33</td>
<td>!</td>
<td>0x41</td>
</tr>
<tr>
<td>0x02</td>
<td>2</td>
<td>STX</td>
<td>Start of text</td>
<td>0x22</td>
<td>34</td>
<td>”</td>
<td>0x42</td>
</tr>
<tr>
<td>0x03</td>
<td>3</td>
<td>ETX</td>
<td>End of text</td>
<td>0x23</td>
<td>35</td>
<td>#</td>
<td>0x43</td>
</tr>
<tr>
<td>0x04</td>
<td>4</td>
<td>EOT</td>
<td>End of transmission</td>
<td>0x24</td>
<td>36</td>
<td>$</td>
<td>0x44</td>
</tr>
<tr>
<td>0x05</td>
<td>5</td>
<td>ENQ</td>
<td>Enquiry</td>
<td>0x25</td>
<td>37</td>
<td>%</td>
<td>0x45</td>
</tr>
<tr>
<td>0x06</td>
<td>6</td>
<td>ACK</td>
<td>Acknowledge</td>
<td>0x26</td>
<td>38</td>
<td>&amp;</td>
<td>0x46</td>
</tr>
<tr>
<td>0x07</td>
<td>7</td>
<td>BEL</td>
<td>Bell</td>
<td>0x27</td>
<td>39</td>
<td>’</td>
<td>0x47</td>
</tr>
<tr>
<td>0x08</td>
<td>8</td>
<td>BS</td>
<td>Backspace</td>
<td>0x28</td>
<td>40</td>
<td>(</td>
<td>0x48</td>
</tr>
<tr>
<td>0x09</td>
<td>9</td>
<td>HT</td>
<td>Horizontal tab</td>
<td>0x29</td>
<td>41</td>
<td>)</td>
<td>0x49</td>
</tr>
<tr>
<td>0x0A</td>
<td>10</td>
<td>LF</td>
<td>New line</td>
<td>0x2A</td>
<td>42</td>
<td>*</td>
<td>0x4A</td>
</tr>
<tr>
<td>0x0B</td>
<td>11</td>
<td>VT</td>
<td>Vertical tab</td>
<td>0x2B</td>
<td>43</td>
<td>+</td>
<td>0x4B</td>
</tr>
<tr>
<td>0x0C</td>
<td>12</td>
<td>FF</td>
<td>Form Feed</td>
<td>0x2C</td>
<td>44</td>
<td>,</td>
<td>0x4C</td>
</tr>
<tr>
<td>0x0D</td>
<td>13</td>
<td>CR</td>
<td>Carriage return</td>
<td>0x2D</td>
<td>45</td>
<td>-</td>
<td>0x4D</td>
</tr>
<tr>
<td>0x0E</td>
<td>14</td>
<td>SO</td>
<td>Shift out</td>
<td>0x2E</td>
<td>46</td>
<td>.</td>
<td>0x4E</td>
</tr>
<tr>
<td>0x0F</td>
<td>15</td>
<td>SI</td>
<td>Shift in</td>
<td>0x2F</td>
<td>47</td>
<td>/</td>
<td>0x4F</td>
</tr>
<tr>
<td>0x10</td>
<td>16</td>
<td>DLE</td>
<td>Data link escape</td>
<td>0x30</td>
<td>48</td>
<td>0</td>
<td>0x50</td>
</tr>
<tr>
<td>0x11</td>
<td>17</td>
<td>DC1</td>
<td>Device control 1</td>
<td>0x31</td>
<td>49</td>
<td>1</td>
<td>0x51</td>
</tr>
<tr>
<td>0x12</td>
<td>18</td>
<td>DC2</td>
<td>Device control 2</td>
<td>0x32</td>
<td>50</td>
<td>2</td>
<td>0x52</td>
</tr>
<tr>
<td>0x13</td>
<td>19</td>
<td>DC3</td>
<td>Device control 3</td>
<td>0x33</td>
<td>51</td>
<td>3</td>
<td>0x53</td>
</tr>
<tr>
<td>0x14</td>
<td>20</td>
<td>DC4</td>
<td>Device control 4</td>
<td>0x34</td>
<td>52</td>
<td>4</td>
<td>0x54</td>
</tr>
<tr>
<td>0x15</td>
<td>21</td>
<td>NAK</td>
<td>Negative ack</td>
<td>0x35</td>
<td>53</td>
<td>5</td>
<td>0x55</td>
</tr>
<tr>
<td>0x16</td>
<td>22</td>
<td>SYN</td>
<td>Synchronous idle</td>
<td>0x36</td>
<td>54</td>
<td>6</td>
<td>0x56</td>
</tr>
<tr>
<td>0x17</td>
<td>23</td>
<td>ETB</td>
<td>End transmission block</td>
<td>0x37</td>
<td>55</td>
<td>7</td>
<td>0x57</td>
</tr>
<tr>
<td>0x18</td>
<td>24</td>
<td>CAN</td>
<td>Cancel</td>
<td>0x38</td>
<td>56</td>
<td>8</td>
<td>0x58</td>
</tr>
<tr>
<td>0x19</td>
<td>25</td>
<td>EM</td>
<td>End of medium</td>
<td>0x39</td>
<td>57</td>
<td>9</td>
<td>0x59</td>
</tr>
<tr>
<td>0x1A</td>
<td>26</td>
<td>SUB</td>
<td>Substitute</td>
<td>0x3A</td>
<td>58</td>
<td>:</td>
<td>0x5A</td>
</tr>
<tr>
<td>0x1B</td>
<td>27</td>
<td>FSC</td>
<td>Escape</td>
<td>0x3B</td>
<td>59</td>
<td>;</td>
<td>0x5B</td>
</tr>
<tr>
<td>0x1C</td>
<td>28</td>
<td>FS</td>
<td>File separator</td>
<td>0x3C</td>
<td>60</td>
<td>&lt;</td>
<td>0x5C</td>
</tr>
<tr>
<td>0x1D</td>
<td>29</td>
<td>GS</td>
<td>Group separator</td>
<td>0x3D</td>
<td>61</td>
<td>=</td>
<td>0x5D</td>
</tr>
<tr>
<td>0x1E</td>
<td>30</td>
<td>RS</td>
<td>Record separator</td>
<td>0x3E</td>
<td>62</td>
<td>&gt;</td>
<td>0x5E</td>
</tr>
<tr>
<td>0x1F</td>
<td>31</td>
<td>US</td>
<td>Unit separator</td>
<td>0x3F</td>
<td>63</td>
<td>?</td>
<td>0x5F</td>
</tr>
</tbody>
</table>

Zheng-Liang Lu
Java Programming
Characters can also be used as (positive) integers on which you can perform arithmetic operations.\footnote{20}

For example,

```
... char x = 'a'; // single-quoted: a char value
System.out.println(x + 1); // output 98!!
System.out.println((\texttt{char})(x + 1)); // output b
String s = "Java"; // double-quoted: a String object
...
```

You can imagine that a String object comprises characters equipped with plentiful tools.\footnote{21}

\footnote{20}See https://en.wikipedia.org/wiki/Cryptography.\footnote{21}As an analogy, a molecule (string) consists of atoms (characters).
Boolean Values

- The program is supposed to do decision making by itself, for example, Google Driverless Car.\(^{22}\)
- To do this, Java has the boolean-type flow controls (selections and iterations).
- This type has 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. (Why?)

\(^{22}\)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><code>&lt;</code></td>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
</tr>
<tr>
<td><code>==</code></td>
<td><code>=</code></td>
<td>equal to</td>
</tr>
<tr>
<td><code>!=</code></td>
<td><code>≠</code></td>
<td>not equal to</td>
</tr>
</tbody>
</table>

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

\[ ^{23} \text{See Table 3-1 in YDL, p. 82.} \]
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.
## Logical Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>logical negation</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>logical conjunction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
<td>logical exclusion</td>
</tr>
</tbody>
</table>

---

\(^{24}\) See Table 3-2 in YDL, p. 102.
• Let $X$ and $Y$ be two Boolean variables.

• Then the truth table for logical operators is as follows:

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
X & Y & \neg X & X \& \& Y & X \lor Y & X \land Y \\
\hline
T & T & F & T & T & F \\
T & F & F & F & T & T \\
F & T & T & F & T & T \\
F & F & T & F & F & F \\
\hline
\end{array}
\]

• Note that the instructions of computers, such as arithmetic operations, are implemented by logic gates.\textsuperscript{25}

\textsuperscript{25}See any textbook for digital circuit design.
“Logic is the anatomy of thought.”

– John Locke (1632–1704)

“This sentence is false.”

– anonymous

“I know that I know nothing.”

– Plato

(In Apology, Plato relates that Socrates accounts for his seeming wiser than any other person because he does not imagine that he knows what he does not know.)