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()
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 interger 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 date 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\(^3\)
  • cannot include any blank between letters
  • cannot contain +, −, *, / and %

• Note that Java is case sensitive\(^4\).

---

\(^3\)See the next page.
\(^4\)The letter A and a are different.
### Reserved Words

<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

- 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.

Data Types

- Java is a **strongly typed** 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.
Primitive Data Types

- boolean
- Numeric Type
  - Integral Type
    - byte
    - short
    - int
    - long
  - Floating-Point Type
    - char
    - 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.
Floating Points

<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>

- 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?)

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 decimal numbers, but also big integers.$^{10}$
- 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

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

- 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.

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

---

12See 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.\(^\text{13}\)
  • For example, 1 = x is wrong.
  • 1 cannot be resolved to a memory space.

\(^\text{13}\)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.\(^{14}\)
- 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.\(^{15}\)

\(^{14}\)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.

---

16See 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.
...
```

\[^{17}\text{Thanks to a lively discussion on on June 7, 2016.}\]
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)
Characters

- 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.\(^\text{18}\)

---

\(^\text{18}\)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>0x01</td>
<td>SOH</td>
<td>0x02</td>
<td>STX</td>
<td>0x03</td>
<td>ETX</td>
</tr>
<tr>
<td>0x04</td>
<td>EOT</td>
<td>0x05</td>
<td>ENQ</td>
<td>0x06</td>
<td>ACK</td>
<td>0x07</td>
<td>BEL</td>
</tr>
<tr>
<td>0x08</td>
<td>BS</td>
<td>0x09</td>
<td>HT</td>
<td>0x0A</td>
<td>LF</td>
<td>0x0B</td>
<td>VT</td>
</tr>
<tr>
<td>0x0C</td>
<td>FF</td>
<td>0x0D</td>
<td>CR</td>
<td>0x0E</td>
<td>SO</td>
<td>0x0F</td>
<td>SI</td>
</tr>
<tr>
<td>0x10</td>
<td>DLE</td>
<td>0x11</td>
<td>DC1</td>
<td>0x12</td>
<td>DC2</td>
<td>0x13</td>
<td>DC3</td>
</tr>
<tr>
<td>0x14</td>
<td>DC4</td>
<td>0x15</td>
<td>NAK</td>
<td>0x16</td>
<td>SYN</td>
<td>0x17</td>
<td>ETB</td>
</tr>
<tr>
<td>0x18</td>
<td>CAN</td>
<td>0x19</td>
<td>EM</td>
<td>0x1A</td>
<td>SUB</td>
<td>0x1B</td>
<td>ESC</td>
</tr>
<tr>
<td>0x1C</td>
<td>FS</td>
<td>0x1D</td>
<td>GS</td>
<td>0x1E</td>
<td>RS</td>
<td>0x1F</td>
<td>US</td>
</tr>
</tbody>
</table>

- **NULL** (0x00) - Start of a line, replaced by space
- **SOH** (0x01) - Start of heading
- **STX** (0x02) - Start of text
- **ETX** (0x03) - End of text
- **EOT** (0x04) - End of transmission
- **ENQ** (0x05) - Enquiry
- **ACK** (0x06) - Acknowledge
- **BEL** (0x07) - Bell
- **BS** (0x08) - Backspace
- **HT** (0x09) - Horizontal tab
- **LF** (0x0A) - New line
- **VT** (0x0B) - Vertical tab
- **FF** (0x0C) - Form feed
- **CR** (0x0D) - Carriage return
- **SO** (0x0E) - Shift out
- **SI** (0x0F) - Shift in
- **DLE** (0x10) - Data link escape
- **DC1** (0x11) - Device control 1
- **DC2** (0x12) - Device control 2
- **DC3** (0x13) - Device control 3
- **DC4** (0x14) - Device control 4
- **NAK** (0x15) - Negative ack
- **SYN** (0x16) - Synchronous idle
- **ETB** (0x17) - End transmission block
- **CAN** (0x18) - Cancel
- **EM** (0x19) - End of medium
- **SUB** (0x1A) - Substitute
- **ESC** (0x1B) - Escape
- **FS** (0x1C) - File separator
- **GS** (0x1D) - Group separator
- **RS** (0x1E) - Record separator
- **US** (0x1F) - Unit separator

Zheng-Liang Lu  
Java Programming  
69 / 88
Characters can also be used as an integer type on which you can perform arithmetic operations.\(^\text{19}\)

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.

---

\(^{19}\) Widely used in information security! See https://en.wikipedia.org/wiki/Cryptography.
Boolean Values

- The program is supposed to do decision making by itself, for example, Google Driverless Car.\(^\text{20}\)
- 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><code>&lt;</code></td>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td><code>≤</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>≥</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 comparison operator is double equality sign (==), not single equality sign (=).

---

21 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>

See Table 3-2 in YDL, p. 102.
Truth Table

- Let $X$ and $Y$ be two Boolean variables.
- Then the truth table for logical operators is as follows:

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$\neg X$</th>
<th>$X &amp; &amp; Y$</th>
<th>$X \parallel Y$</th>
<th>$X \wedge Y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

- Note that the instructions of computers, such as arithmetic operations, are implemented by logic gates.\(^{23}\)

\(^{23}\)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.)
Arithmetic Compound Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>Increment</td>
</tr>
<tr>
<td>+=</td>
<td>Addition assignment</td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assignment</td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assignment</td>
</tr>
<tr>
<td>/=</td>
<td>Division assignment</td>
</tr>
<tr>
<td>%=</td>
<td>Modulus assignment</td>
</tr>
<tr>
<td>--</td>
<td>Decrement</td>
</tr>
</tbody>
</table>

- Note that these shorthand operators are not available in languages such as Matlab, R, and Python.
Example

```java
...  
  int x = 1;
  System.out.println(x); // output 1
  x = x + 1;
  System.out.println(x); // output 2
  x += 2;
  System.out.println(x); // output 4
  x++; // equivalent to x += 1 and x = x + 1
  System.out.println(x); // output 5
...```

Zheng-Liang Lu  
Java Programming
• The compound assignment operators are also useful for char values.\textsuperscript{24}

• For example,

\begin{verbatim}
... char s = 'a'; System.out.println(s); // output a s += 1; System.out.println(s); // output b s++; System.out.println(s); // output c ...
\end{verbatim}
The expression `++x` first increments the value of `x` and then returns `x`.

Instead, the expression `x++` first returns the value of `x` and then increments itself.

For example,

```java
... int x = 1;
int y = ++x;
System.out.println(y); // output 2; aka preincrement
System.out.println(x); // output 2

int w = 1;
int z = w++;
System.out.println(z); // output 1; aka postincrement
System.out.println(w); // output 2
...
```

We will use these notations very often.
## Operator Precedence

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>var++</code> and <code>var--</code> (Postfix)</td>
</tr>
<tr>
<td></td>
<td><code>+</code>, <code>-</code> (Unary plus and minus), <code>++var</code> and <code>--var</code> (Prefix)</td>
</tr>
<tr>
<td></td>
<td>(type) (Casting)</td>
</tr>
<tr>
<td></td>
<td><code>!</code> (Not)</td>
</tr>
<tr>
<td></td>
<td><code>*</code>, <code>/</code>, <code>%</code> (Multiplication, division, and remainder)</td>
</tr>
<tr>
<td></td>
<td><code>+</code>, <code>-</code> (Binary addition and subtraction)</td>
</tr>
<tr>
<td></td>
<td><code>&lt;</code>, <code>&lt;=</code>, <code>&gt;</code>, <code>&gt;=</code> (Comparison)</td>
</tr>
<tr>
<td></td>
<td><code>==</code>, <code>!=</code> (Equality)</td>
</tr>
<tr>
<td></td>
<td><code>^</code> (Exclusive OR)</td>
</tr>
<tr>
<td></td>
<td><code>&amp;&amp;</code> (AND)</td>
</tr>
<tr>
<td></td>
<td>`</td>
</tr>
<tr>
<td></td>
<td><code>=</code>, <code>+=</code>, <code>-=</code>, <code>*=</code> <code>=/</code>, <code>%=</code> (Assignment operator)</td>
</tr>
</tbody>
</table>

25 See Table 3-10 in YDL, p. 116.
Using Parentheses

- Parentheses are used in expressions to change the natural order of precedence among the operators.
- One always evaluates the expression inside of parentheses first.
Scanner Objects

- It is not convenient to modify the source code and recompile it for a different radius.
- Reading from the console enables the program to receive an input from the user.
- A **Scanner** object provides some input methods, say the input received from the keyboard or the files.
- Java uses **System.in** to refer to the standard input device, by default, the keyboard.
Write a program which receives a number as input, and outputs the area of the circle.

```java
import java.util.Scanner;

Scanner input = new Scanner(System.in);
System.out.println("Enter r?");
// input
int r = input.nextInt();
// algorithm
double area = r * r * 3.14;
// output
System.out.println(area);
input.close();
```
• In the listing, Line 3 is to create a **Scanner** object by the `new` operator, as an agent between the keyboard and your program.
• Note that all objects are resided in the **heap** of the memory.
• To control this object, its **memory address** is then assigned to the variable `input` which is a variable in the **stack** of memory.
• So the variable `input` is a **reference**.
• We will discuss the objects and reference variables later.
Methods Provided by Scanner Objects

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextByte()</td>
<td>reads an integer of the <code>byte</code> type.</td>
</tr>
<tr>
<td>nextShort()</td>
<td>reads an integer of the <code>short</code> type.</td>
</tr>
<tr>
<td>nextInt()</td>
<td>reads an integer of the <code>int</code> type.</td>
</tr>
<tr>
<td>nextLong()</td>
<td>reads an integer of the <code>long</code> type.</td>
</tr>
<tr>
<td>nextFloat()</td>
<td>reads a number of the <code>float</code> type.</td>
</tr>
<tr>
<td>nextDouble()</td>
<td>reads a number of the <code>double</code> type.</td>
</tr>
<tr>
<td>next()</td>
<td>reads a string that ends before a whitespace character.</td>
</tr>
<tr>
<td>nextLine()</td>
<td>reads a line of text (i.e., a string ending with the <code>Enter</code> key pressed).</td>
</tr>
</tbody>
</table>

See Table 2-1 in YDL, p. 38.
Example: Mean and Standard Deviation

Write a program which calculates the mean and the standard deviation of 3 numbers.

- The mean of 3 numbers is given by $\bar{x} = \left( \sum_{i=1}^{3} x_i \right) / 3$.
- Also, the resulting standard deviation is given by

$$S = \sqrt{\frac{\sum_{i=1}^{3} (x_i - \bar{x})^2}{3}}.$$

- You may use these two methods:
  - Math.pow(double x, double y) for $x^y$
  - Math.sqrt(double x) for $\sqrt{x}$
- See more methods within Math class.
Scanner input = new Scanner(System.in);
System.out.println("a = ?");
double a = input.nextDouble();
System.out.println("b = ?");
double b = input.nextDouble();
System.out.println("c = ?");
double c = input.nextDouble();

double mean = (a + b + c) / 3;
double std = Math.sqrt((Math.pow(a - mean, 2) +
    Math.pow(b - mean, 2) +
    Math.pow(c - mean, 2)) / 3);

System.out.println("mean = " + mean);
System.out.println("std = " + std);