

Java Programming

Zheng-Liang Lu

Department of Computer Science & Information Engineering
National Taiwan University

Online Course

```
1 class Lecture2 {  
2  
3     "Data types, Variables, and Operators"  
4  
5 }  
6  
7 // Keywords:  
8 byte, short, int, long, char, float, double, boolean, true, false,  
9 import, new
```

Example

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

- Input: how to store the data?
- Algorithm: how to compute the area?
- Output: how to show the result?

```
1 public class ComputeAreaDemo {  
2  
3     public static void main(String[] args) {  
4  
5         // INPUT  
6         int r = 10;  
7  
8         // ALGORITHM  
9         double A = r * r * 3.14;  
10  
11        // OUTPUT  
12        System.out.println(A);  
13  
14    }  
15 }
```

- Here we use two of **primitive data types**: **int**, **double**.



Variable \approx Box

Variable Declaration

- First, we name the variable, say x.
- We then need to determine a proper type for x.
- For example,

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

Naming Rules

- The naming rule excludes the following cases:
 - cannot start with a digit;
 - cannot be any reserved word (see the next page);
 - cannot include any blank between letters;
 - cannot contain operators, like $+$, $-$, $*$, $/$.
- Note that Java is case-sensitive.¹
- These rules are also applicable to methods, classes (, and more).

¹For example, the letter A is different from the letter a.

Reserved Words²

<code>abstract</code>	<code>double</code>	<code>int</code>	<code>super</code>
<code>assert</code>	<code>else</code>	<code>interface</code>	<code>switch</code>
<code>boolean</code>	<code>enum</code>	<code>long</code>	<code>synchronized</code>
<code>break</code>	<code>extends</code>	<code>native</code>	<code>this</code>
<code>byte</code>	<code>final</code>	<code>new</code>	<code>throw</code>
<code>case</code>	<code>finally</code>	<code>package</code>	<code>throws</code>
<code>catch</code>	<code>float</code>	<code>private</code>	<code>transient</code>
<code>char</code>	<code>for</code>	<code>protected</code>	<code>try</code>
<code>class</code>	<code>goto</code>	<code>public</code>	<code>void</code>
<code>const</code>	<code>if</code>	<code>return</code>	<code>volatile</code>
<code>continue</code>	<code>implements</code>	<code>short</code>	<code>while</code>
<code>default</code>	<code>import</code>	<code>static</code>	
<code>do</code>	<code>instanceof</code>	<code>strictfp*</code>	

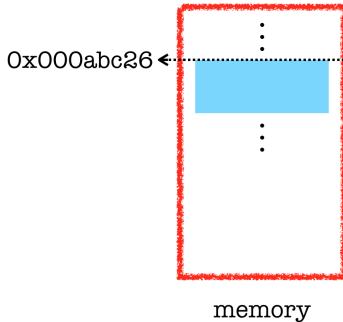
- Coverage: 44 / 50 = 88%.

²Based on JDK8.

Things behind Variable Declaration

- Variable declaration is actually asking the compiler to **allocate** a memory space for the variable.
- In particular, its data type determines the size of memory allocation.
- The size is quantified in **bits** or **bytes**.
 - A bit presents a binary digit.
 - 1 byte is equal to 8 bits.
- For example, 32 bits (or 4 bytes) are allocated for an **int** value.

Variable as Alias of Memory Address



- Literals that start with 0x are hexadecimal (hex) integers.³
- Hexadecimal numbers are widely used to present, say a memory address and a color.⁴

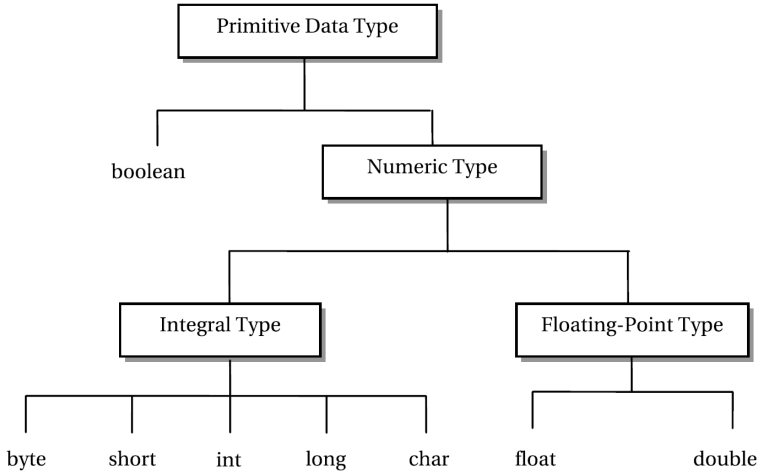
³See <https://en.wikipedia.org/wiki/Hexadecimal>.

⁴Try <https://htmlcolorcodes.com/>.

Data Types

- Every variable needs a type.
- Also, every statement (or expression) has a type.
- The notion of data types is vital to programming languages.
 - I would say that, the idea of data types acts like the law of nature.
- Java is a static-typed language.
 - A variable is available after declaration.
- We now proceed to introduce the two categories of data types: primitive types, and reference types.

Primitive Types⁵



⁵See Figure 3-4 in Sharan, p. 67.

Digression: Binary System

- We are familiar with the decimal system. (Why?)
- For computers, the binary system is adopted because of its nature: the only two states, on and off.
- However, both are equivalent except that they differ in representations.
- For example,

$$999_{10} = 9 \times 10^2 + 9 \times 10^1 + 9 \times 10^0.$$

- Similarly,

$$111_2 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 7_{10}.$$

- No need to write code in binary because we are using high-level languages.

Integers

Name	Bits	Range	Approx. Range
byte	8	0 to 255	≤ 255
short	16	-32768 to 32767	$\pm 3 \times 10^4$
int	32	-2147483648 to 2147483647	$\pm 2 \times 10^9$
long	64	-9223372036854775808 to 9223372036854775807	$\pm 9 \times 10^{18}$

- The most commonly used integer type is **int**.
- Note that the range is limited to its finite size of storage.
- If a number exceeds the feasible range of the type, then an **overflow** occurs.

Float-Point Numbers

Name	Bits	Approx. Range
float	32	$1.4e-045$ to $3.4e+038$
double	64	$4.9e-324$ to $1.8e+308$

- The notation e (or E in some cases) represents the scientific notation, based 10.
 - For example, $1e2 = 100$ and $-1.8e-3 = -0.0018$.
- Float-point numbers are used when evaluating expressions that require fractional precision.
 - For example, $\sin()$, $\cos()$, and $\text{sqrt}()$.
- If so, then integers seem redundant!
- However, the floating-point system can only **approximate** the real-number arithmetic! (Why?)

Machine Epsilon⁷

```
1 public class MachineEpsilonDemo {  
2  
3     public static void main(String[] args) {  
4  
5         System.out.println(0.5 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);  
6         // Output? Why?  
7  
8     }  
9  
10 }
```

- Could you explain why?
- We then design algorithms to abbreviate the machine epsilon.
- Even more, we avoid to use the float-point numbers but integers in critical applications.⁶


⁶Also read <https://news.cnyes.com/news/id/3680649>.

⁷See https://en.wikipedia.org/wiki/Machine_epsilon and <https://0.30000000000000004.com/>.

Another Example

```
1 ...  
2     System.out.println(3.14 + 1e20 - 1e20);    // Output?  
3     System.out.println(3.14 + (1e20 - 1e20));  // Output?  
4 ...
```

- Floating-point arithmetic (FP)⁸ is arithmetic using formulaic representation of real numbers as an approximation to support a **trade-off between range and precision**.
- You may also read this article [What Every Computer Scientist Should Know About Floating-Point Arithmetic](https://en.wikipedia.org/wiki/What_Every_Computer_Scientist_Should_Know_About_Floating-Point_Arithmetic).

⁸See https://en.wikipedia.org/wiki/Floating-point_arithmetic. 

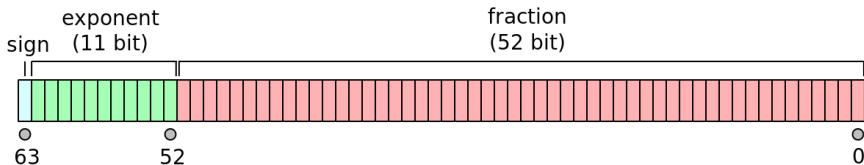
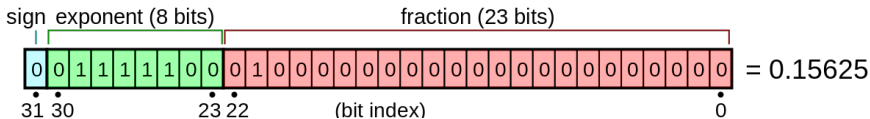
IEEE Floating-Point Representation⁹

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

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

⁹IEEE754; William Kahan (1985). Also see https://en.wikipedia.org/wiki/Double-precision_floating-point_format.

Illustration



- That is why we call a **double** value.
- Double values have at least **16** significant digits!

Assignments

- An assignment statement designates a value to the variable.

```
1      int x;      // Variable declaration.  
2      ...  
3      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.¹⁰
 - For example, $1 = x$ is wrong.
 - **1 cannot be resolved to a memory space.**

¹⁰ 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

- 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.

Arithmetic Operators¹¹

Operator	Operation	Example	Result
+	Addition	12 + 34	46
-	Subtraction	56 - 78	-22
*	Multiplication	90 * 12	1080
/	Division	3.0 / 2.0	1.5
%	Remainder	20 % 3	2

- What if 3 / 2?
- Note that the operator depends on the operands involved.

¹¹See Table 2-3 in YDL, p. 46.

Concept Check

```
1 ...  
2     double x = 1 / 2;  
3     System.out.println(x); // Output?  
4 ...
```

- What is the output?
- Can you explain this result?

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.
 - Therefore, Java is a weakly-typed language.¹²
- However, there is no automatic conversion from **double** to **int**. (Why?)
- To do so, you must use a **cast**, which performs an explicit conversion.
- Similarly, a **long** value is not compatible to **int**.

¹²See <https://android.jlelse.eu/>

Casting

```
1 ...  
2     int x = 1;  
3     double y = x; // Compatible; implicitly converted.  
4     x = y;        // Not allowed unless casting.  
5     x = (int) y;   // Succeeded!!  
6 ...
```

- 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 \nleftrightarrow large-size types (need a cast).
- Simple types \rightarrow complicated types.
- Simple types \nleftrightarrow complicated types (need a cast).

Text: Characters & String

- Each character is encoded in a sequence of 0's and 1's.
 - For example, ASCII. (See the next page.)
- Java uses **Unicode** to represent characters denoted by **char**, which is a 16-bit unsigned value.¹³
- However, we often use **String** to present text, as shown before.
- As an analogy, a molecule (string) consists of atoms (characters).¹⁴

¹³Unicode defines a fully international character set that can represent all of the characters found in all human languages.

¹⁴A **String** object comprises characters equipped with plentiful tools. 

ASCII (7-bit version)

Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT End of transmission	0x24	36	\$	0x44	68	D	0x64	100	d
0x05	5	ENQ Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS Backspace	0x28	40	(0x48	72	H	0x68	104	h
0x09	9	TAB Horizontal tab	0x29	41)	0x49	73	I	0x69	105	i
0x0A	10	LF New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1 Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2 Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3 Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4 Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC Escape	0x3B	59	;	0x5B	91	[0x7B	123	{
0x1C	28	FS File separator	0x3C	60	<	0x5C	92	\	0x7C	124	
0x1D	29	GS Group separator	0x3D	61	=	0x5D	93]	0x7D	125	}
0x1E	30	RS Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Example

```
1 ...
2     char c = 'a'; // A char value should be single-quoted.
3     System.out.println(c + 1); // Output 98!! (why?)
4     System.out.println((char)(c + 1)); // Output b.
5
6     String s = "Java"; // A string should be double-quoted.
7     System.out.println(s + 326); // Output Java326.
8 ...
```

- Why applying arithmetic operators on characters, for example, Line 4?¹⁵
- In Line 7, the result of applying the + operator to **String** is totally different from Line 3 & 4. (Why?)

¹⁵See <https://en.wikipedia.org/wiki/Cryptography>.

Boolean Values¹⁷

- The program could do **decision making** by itself, for example, self-driving cars.¹⁶
- To do this, Java has the **boolean**-type flow controls (branching and iteration).
- 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?)

¹⁶See <https://www.google.com/selfdrivingcar/>.

¹⁷George Boole (1815–1864) is the namesake of the branch of algebra known as Boolean algebra. See https://en.wikipedia.org/wiki/George_Boole.

Relational Operators¹⁸

<i>Java Operator</i>	<i>Mathematics Symbol</i>	<i>Name</i>
<	<	less than
<=	≤	less than or equal to
>	>	greater than
>=	≥	greater than or equal to
==	=	equal to
!=	≠	not equal to

- Relational operators take two operands and return a **boolean** value.
- Note that the mathematical equality operator is ==, not =.

¹⁸See Table 3-1 in YDL, p. 82.

Example

```
1 ...  
2     int x = 2;  
3     System.out.println(x > 1);      // Output true.  
4     System.out.println(x < 1);      // Output false.  
5     System.out.println(x == 1);     // Output false.  
6     System.out.println(x != 1);     // Output true.  
7     System.out.println(1 < x < 3);  // Sorry?  
8 ...
```

- In Line 7, $1 < x < 3$ is **syntactically wrong**.
- You need to split a complex statement into several basic statements and joint them by **logical operators**.
- For example, $1 < x < 3$ should be $1 < x \&\& x < 3$, where $\&\&$ presents the AND operator.

Logical Operators^{19,20}

Operator	Name
!	NOT
&&	AND
	OR
^	EXCLUSIVE-OR

- We use XOR to denote the exclusive-or operator.

¹⁹See Table 3-2 in YDL, p. 102.

²⁰I skip over the bit-wise operators because most of Java programmers do not use these directly. See <https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html> if necessary.

Truth Table

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

X	Y	$\neg X$	$X \& Y$	$X \parallel Y$	$X \wedge Y$
T	T	F	T	T	F
T	F	F	F	T	T
F	T	T	F	T	T
F	F	T	F	F	F

- It is worth to know that basic instructions of computers, such as arithmetic operators, are implemented by Boolean logics.²¹
 - For example, 1-bit adder can be implemented by using the XOR operator. (Try!)

²¹Can you image that the combination of these very basic elements (0, 1, AND, OR, NOT) with jumps produces [AlphaGo](#) which beat human beings in 2016.

"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 Operators²²

Operator	Operator name
++	Increment (by one)
+=	Addition assignment
- =	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
--	Decrement (by one)

²²Note that these shorthand operators are not available in languages such as Matlab and R.

Example

```
1 ...  
2     int x = 1;  
3     System.out.println(x); // Output 1.  
4  
5     x = x + 1;  
6     System.out.println(x); // Output 2.  
7  
8     x += 2;  
9     System.out.println(x); // Output 4.  
10  
11     x++; // Equivalent to x += 1 and x = x + 1.  
12     System.out.println(x); // Output 5.  
13 ...
```

- The compound assignment operators are also useful for **char** values.²³
- For example,

```
1 ...  
2     char s = 'a';  
3     System.out.println(s); // Output a.  
4  
5     s += 1;  
6     System.out.println(s); // Output b.  
7  
8     s++;  
9     System.out.println(s); // Output c.  
10 ...
```

²³Contribution by Mr. Edward Wang (Java265) on May 1, 2016.

++x VS. x++

```
1 ...  
2     int x = 1;  
3     int y = ++x;  
4     System.out.println(y); // Output 2.  
5     System.out.println(x); // Output 2.  
6  
7     int w = 1;  
8     int z = w++;  
9     System.out.println(z); // Output 1.  
10    System.out.println(w); // Output 2.  
11 ...
```

- The expression ++x first increments the value of x and then returns x.
- Instead, x++ first returns the value of x and then increments itself.
- We will use these notations very often.

Operator Precedence²⁴

<i>Precedence</i>	<i>Operator</i>
	<code>var++</code> and <code>var--</code> (Postfix)
	<code>+</code> , <code>-</code> (Unary plus and minus), <code>++var</code> and <code>--var</code> (Prefix)
	(type) (Casting)
	<code>!</code> (Not)
	<code>*</code> , <code>/</code> , <code>%</code> (Multiplication, division, and remainder)
	<code>+</code> , <code>-</code> (Binary addition and subtraction)
	<code><</code> , <code><=</code> , <code>></code> , <code>>=</code> (Comparison)
	<code>==</code> , <code>!=</code> (Equality)
	<code>^</code> (Exclusive OR)
	<code>&&</code> (AND)
	<code> </code> (OR)
	<code>=</code> , <code>+=</code> , <code>-=</code> , <code>*=</code> , <code>/=</code> , <code>%=</code> (Assignment operator)

²⁴See Table3-10 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.

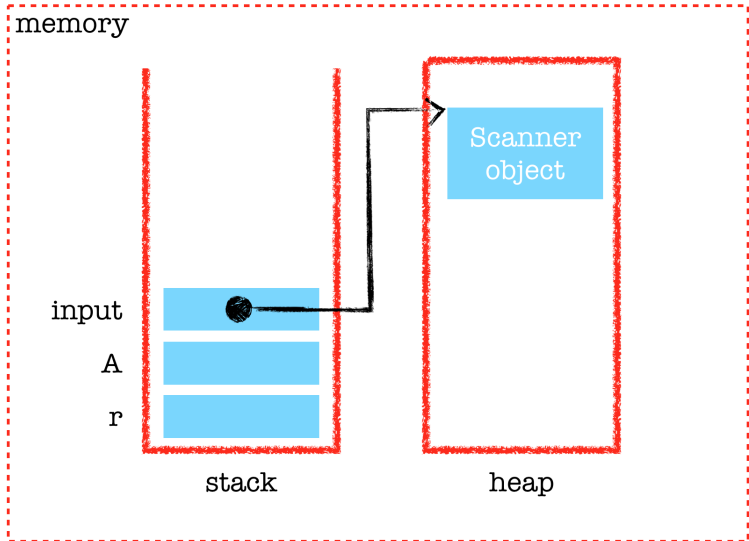
Example: Reading Input From Console

```
1 import java.util.Scanner;
2 ...
3     // Create Scanner object to receive data from keyboard.
4     Scanner input = new Scanner(System.in);
5
6     // INPUT
7     System.out.println("Enter r?");
8     // Assign an integer from the keyboard to r.
9     int r = input.nextInt();
10
11    // ALGORITHM
12    double A = r * r * 3.14;
13
14    // OUTPUT
15    System.out.println(A);
16    input.close(); // Cleanup: reclaim the resource.
17 ...
```

- Line 4 is to create a **Scanner** object as an agent between the keyboard and your program, by using the `new` operator.
- It is worthy to note that all objects are **dynamically created** and resided in the **heap** of memory. (See the figure in the next page.)
- To manipulate this object, its **memory address** is then assigned to the variable *input*, which is allocated in the **stack** of memory.
- Then *input* is called a **reference** to the **Scanner** object.²⁵
- From this, the memory contains human data and also references (i.e., memory addresses).

²⁵If you have programming experience in C/C++, then this reference is similar to the concept of **pointers**.

Simplified Memory Model



Methods Provided by Scanner²⁶

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

²⁶See Table 2-1 in YDL, p. 38.

Example: Body Mass Index (BMI)

Write a program which takes user name, height (in cm), weight (in kgw) as input, and then outputs the user name attached with his/her BMI, which is

$$\text{BMI} = \frac{\text{weight (kgw)}}{\text{height}^2(\text{m}^2)}.$$

- Be careful about unit conversion!

```
1 ...
2     Scanner input = new Scanner(System.in);
3
4     // INPUT
5     System.out.println("Enter your name?");
6     String name = input.nextLine();
7
8     System.out.println("Enter your height (cm)?");
9     double height = input.nextDouble();
10
11    System.out.println("Enter your weight (kgw)?");
12    double weight = input.nextDouble();
13
14    // ALGORITHM
15    double bmi = 10000 * weight / height / height;
16
17    // OUTPUT: name (bmi)
18    System.out.println(name + " (" + bmi + ")");
19 ...
```


Exercise: Two Simple Descriptive Statistics

Write a program which takes 3 numbers as user input, and calculates the average with the standard deviation.

- Note that the standard deviation of these three is given by

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

where $\bar{x} = \left(\sum_{i=1}^3 x_i \right) / 3$.

- Two of **Math** methods are listed below:²⁷
 - Math.pow**(double x , double y) for x^y .
 - Math.sqrt**(double x) for \sqrt{x} .

²⁷See

```

1  ...
2      // INPUT
3      Scanner input = new Scanner(System.in);
4      System.out.println("a = ?");
5      double a = input.nextDouble();
6      System.out.println("b = ?");
7      double b = input.nextDouble();
8      System.out.println("c = ?");
9      double c = input.nextDouble();
10     input.close();
11
12     // ALGORITHM
13     double mean = (a + b + c) / 3;
14     double std = Math.sqrt((Math.pow(a - mean, 2) +
15                             Math.pow(b - mean, 2) +
16                             Math.pow(c - mean, 2)) / 3);
17
18     // OUTPUT
19     System.out.println("Mean = " + mean);
20     System.out.println("Std = " + std);
21     ...

```