



# Chapter 2

## Data Types & Computation

Yepang Liu (刘烨庞)

[liuyp1@sustech.edu.cn](mailto:liuyp1@sustech.edu.cn)



# Outline

- ▶ Java's primitive types (基本数据类型)
- ▶ Arithmetic computation (算术运算)
- ▶ Evaluation order of arithmetic expressions (求值顺序)



# Data types

- ▶ All programs are composed of **data** and **operations** on the data.
- ▶ A **data type** tells the computer how the programmer intends to use the data
  - What is the meaning of the data?
  - What operations can be done on the data?
  - How to store the data in memory? (i.e., the arrangement of 0/1 bits)
- ▶ Java supports different types of data: **primitive data types** (e.g., numbers) and **complex data types** (e.g., Strings, Objects), a.k.a **reference types** or **non-primitive data types**



# Primitive data types

- ▶ Java supports eight primitive types (基本数据类型)
  - Integral types (整数类型): `byte`, `short`, `int`, `long` => whole numbers
  - Floating-point types (浮点数类型): `float`, `double` => real numbers (having both integer and decimal parts)
  - The `boolean` data type (布尔类型) => truth values: `true/false`
  - The `char` data type (字符类型) => characters



# Integral numbers (Integers)

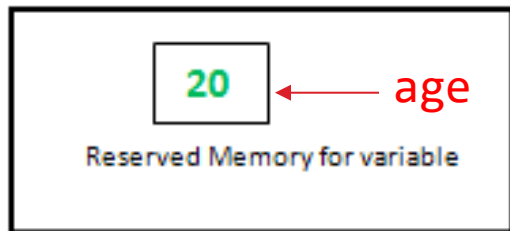
Type	Size	Range
byte	8 bits	-128 (100000000*) to +127 (01111111*)
short	16 bits	-32,768 to +32,767
int	32 bits	(about) -2.147 billion to +2.147 billion
long	64 bits	(about) -10E18 to +10E18 (-9.22 quadrillions to +9.22 quadrillions)

Example: `int age = 20;`

\*The left-most is the sign bit: 0 means positive, 1 means negative

# Meaning of int age = 20;

- ▶ The statement tells the computer to
  - Allocate space in memory to hold data of `int` type
  - Give the memory location a name “age”, such that we can refer to the data stored in the location using the name in the program (we say we created a **variable** named `age`)
  - Store the value 20 to the allocated space



**RAM**

If age receives another assignment later, the value “20” will be replaced by the new value.

# Floating-Point Numbers

Type	Size	Range
float (单精度)	32 bits	-3.4E+38 to +3.4E+38
double (双精度)	64 bits	-1.7E+308 to 1.7E+308

Example:

- `double pi = 3.1415926;`
- `float f = 234.5f;`

The value 234.5 by default is of type double, so f is needed to tell the compiler this is a value of float type



# The precision of double and float

- ▶ The **float** type: **single-precision** floating-point number
  - A float has approximately 7 decimal digits
- ▶ The **double** type: **double-precision** floating-point number
  - A double has approximately 16 decimal digits

```
float f = 1.2345678990922222f; // 16 decimal digits
double d = 1.22222222222222222222; // 20 decimal digits
System.out.println("f = " + f + "\t" + "d = " + d);
```

f = 1.2345679

d = 1.22222222222222223

**Note:** You may encounter **precision problems** when using float and double. For example, the double type cannot precisely represent the value 0.1 due to the binary representation of floating-point numbers (IEEE 754 standard). For computations requiring high precisions, you may consider using BigDecimal: <https://www.tutorialspoint.com/rounding-off-errors-in-java>.





# Think about this

- ▶ **Why computers cannot store real numbers of infinite precisions (such as the irrational number  $\pi$ )?**
- ▶ It would otherwise require infinite memory (resources are finite in computers). This is why the built-in primitive types can only represent a range of values.



# The boolean data type

- ▶ Represents **one bit of information** (the real size in memory depends on language implementation, typically 8 bits)
- ▶ Has only two possible values: **true** and **false**
- ▶ Often used to represent conditions in programs

Example: **boolean isChecked = true;**

# The char data type

- ▶ Represents a single 16-bit Unicode character
- ▶ Ranges from ‘\u0000’ to ‘\uffff’: 65536 characters, covering characters of most modern languages and a large number of symbols

```
char c1 = 'a';
```

```
char c2 = '\u5357';
```

```
char c3 = '\u79d1';
```

```
char c4 = '\u5927';
```

Prints: 南 科 大

```
System.out.printf("%c %c %c", c2, c3, c4);
```



# Outline

- ▶ Java's primitive types
- ▶ Arithmetic computation (算术运算)
- ▶ Evaluation order of arithmetic expressions



# Adding two integers

```
import java.util.Scanner;

public class Addition {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        int number1 = 1;

        System.out.print("Enter the second integer: ");
        int number2 = input.nextInt();

        int sum = 0;
        sum = number1 + number2;
        System.out.printf("Sum is %d\n", sum);
    }
}
```

```
1 package lecture1;
2
3 import java.util.Scanner;
4
5 public class Addition {
6     public static void main(String[] args) {
7         Scanner input = new Scanner(System.in);
8         int number1 = 1;
9
10        System.out.print("Enter the second integer: ");
11        int number2 = input.nextInt();
12
13        int sum = 0;
14        sum = number1 + number2;
15        System.out.printf("Sum is %d\n", sum);
16    }
17 }
```

Run: Addition x



# Adding two integers

```
import java.util.Scanner;
```

```
public class Addition {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        int number1 = 1;
```

```
        System.out.print("Enter the second integer: ");
```

```
        int number2 = input.nextInt();
```

```
        int sum = 0;
```

```
        sum = number1 + number2;
```

```
        System.out.printf("Sum is %d\n", sum);
```

```
    }
```

```
}
```

- `import` helps the compiler locate a class (e.g., `Scanner`) that is used in this program
- In Java, related classes are grouped into **packages**, e.g., `java.util` package provides commonly-used library classes.
- These classes are collectively called **Java class library**, or **Java Application Programming Interface (Java API)**



# Adding two integers

```
import java.util.Scanner;
```

```
public class Addition {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        int number1 = 1;
```

```
        System.out.print("Enter the second integer: ");
```

```
        int number2 = input.nextInt();
```

```
        int sum = 0;
```

```
        sum = number1 + number2;
```

```
        System.out.printf("Sum is %d\n", sum);
```

```
    }
```

```
}
```

- ▶ The **Scanner** class enables a program to read input data
- ▶ The data can come from different sources, such as the keyboard or a file on disk
- ▶ **Standard input object, System.in**, enables a program to read input data typed by the user



# Adding two integers

```
import java.util.Scanner;
```

```
public class Addition {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        int number1 = 1;
```

```
        System.out.print("Enter the second integer: ");
```

```
        int number2 = input.nextInt();
```

```
        int sum = 0;
```

```
        sum = number1 + number2;
```

```
        System.out.printf("Sum is %d\n", sum);
```

```
    }
```

```
}
```

## Variable Declaration

- Variables must be declared with a **type** (e.g., **Scanner**) and a **name** (e.g., **input**) before use
- A variable's **type** specifies what kind of information is stored at that location in memory
- A variable's **name** enables the program to access the value of the variable in memory



# Adding two integers

```
import java.util.Scanner;
```

```
public class Addition {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        int number1 = 1;
```

```
        System.out.print("Enter the second integer: ");
```

```
        int number2 = input.nextInt();
```

```
        int sum = 0;
```

```
        sum = number1 + number2;
```

```
        System.out.printf("Sum is %d\n", sum);
```

```
    }
```

```
}
```

- ▶ The **new** keyword creates an object (we will talk more later)
- ▶ The **assignment operator** = assigns the value on its right to the operand on its left. Here, the input variable will point to the scanner object.



# Adding two integers

```
import java.util.Scanner;
```

```
public class Addition {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        int number1 = 1;
```

```
        System.out.print("Enter the second integer: ");
```

```
        int number2 = input.nextInt();
```

```
        int sum = 0;
```

```
        sum = number1 + number2;
```

```
        System.out.printf("Sum is %d\n", sum);
```

```
    }
```

```
}
```

- ▶ Declare a variable named “number1” with type `int`
- ▶ Assign value 1 to the variable “number1”

# Adding two integers

```
import java.util.Scanner;
```

```
public class Addition {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        int number1 = 1;
```

```
        System.out.print("Enter the second integer: ");  
        int number2 = input.nextInt();
```

```
        int sum = 0;
```

```
        sum = number1 + number2;
```

```
        System.out.printf("Sum is %d\n", sum);
```

```
    }
```

```
}
```

- ▶ **Print** a message (a.k.a. prompt) that directs the user to take a specific action
- ▶ Scanner method **nextInt** obtains an integer from the user. The program waits until the user types the number on the keyboard and press the Enter key to submit the number (the method is **blocking**).
- ▶ Assign the input value directly to the variable "number2"



# Adding two integers

```
import java.util.Scanner;
```

```
public class Addition {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        int number1 = 1;
```

```
        System.out.print("Enter the second integer: ");
```

```
        int number2 = input.nextInt();
```

```
        int sum = 0;
```

```
        sum = number1 + number2;
```

```
        System.out.printf("Sum is %d\n", sum);
```

```
    }
```

```
}
```

- ▶ **Expressions:** Portions of statements that contain calculations (e.g., `number1+number2`)
- ▶ The computer **reads** the values of `number1` and `number2` from memory, adds the two values and **stores** the result to the memory location represented by `sum`
- ▶ Note that `sum`'s initial value 0 will be replaced by the new value



# Adding two integers

```
import java.util.Scanner;

public class Addition {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);

        int number1 = 1;

        System.out.print("Enter the second integer: ");
        int number2 = input.nextInt();

        int sum = 0;
        sum = number1 + number2;

        System.out.printf("Sum is %d\n", sum);
    }
}
```

printf(): helps print formatted output

Format specifier %d is a placeholder for an int value (the letter 'd' stands for "decimal integer")



# Arithmetic operators

Java has five **binary arithmetic operators** (they operate on two operands)

Operator	Use	Description
+	op1 + op2	Adds op1 and op2; also used to concatenate strings
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes the remainder of dividing op1 by op2

# Examples

- ▶ `int x = 3; int y = 2; int z = x / y;`
- ▶ **Integer division** yields an integer quotient. The fractional part is simply discarded (**z gets the value 1**)
  
- ▶ `int a = 10; int b = 3; int c = a % b;`
- ▶ **c gets the value 1** (the **remainder** of dividing 10 by 3 is 1)



# Evaluation order

- ▶ An arithmetic expression may contain multiple operators and operands (e.g.,  $1 + 2 * 5$ )
- ▶ The order in which the operators get evaluated depends on their **precedence** (优先级) and **associativity** (结合性)





# Precedence of operators

- ▶ Precedence specifies **the priority of an operator**
- ▶  $*$ ,  $/$  and  $\%$  operators have the same level of precedence
- ▶  $+$  and  $-$  have the same level of precedence
- ▶  $*$ ,  $/$  and  $\%$  have higher precedence than  $+$  and  $-$
- ▶ So, in expression  $1 + 2 * 5$ , the multiplication operator will be applied first.



# Associativity of operators

- ▶ In case there are multiple operators of the same precedence in an expression, their evaluation order is determined by their **associativity**
- ▶ If an expression contains multiple  $*$ ,  $/$  and  $\%$  operators, they are applied **from the left to right**
- ▶ If an expression contains multiple  $+$  and  $-$  operators, they are also applied **from the left to right**



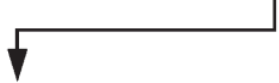
Step 1.  $y = 2 * 5 * 5 + 3 * 5 + 7;$  (Leftmost multiplication)

$2 * 5$  is 10



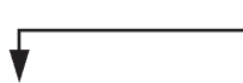
Step 2.  $y = 10 * 5 + 3 * 5 + 7;$  (Leftmost multiplication)

$10 * 5$  is 50



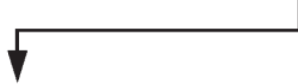
Step 3.  $y = 50 + 3 * 5 + 7;$  (Multiplication before addition)

$3 * 5$  is 15



Step 4.  $y = 50 + 15 + 7;$  (Leftmost addition)

$50 + 15$  is 65



Step 5.  $y = 65 + 7;$  (Last addition)

$65 + 7$  is 72



Step 6.  $y = 72$  (Last operation—place 72 in y)



# Parentheses in expressions

- ▶ In Java, parentheses operator  $()$  has the highest level of precedence
- ▶ In expression  $(1 + 2) * 3$ , the addition will be done first because of the parentheses
- ▶ Parentheses have left associativity.
- ▶ In expression  $(1 + 2) * (3 + 4)$ ,  $1 + 2$  will be done first
- ▶ In case of **nested parentheses**, the expression in the innermost set of parentheses is evaluated first:  $((a + b) * c)$



# Conditional expressions

- ▶ An expression that can be **true** or **false**
- ▶ Conditional expressions involve two types of operators:
  - **Equality operators** (相同运算符): **==**, **!=**
  - **Relational operators** (关系运算符): **>**, **<**, **>=**, **<=**
- ▶ Some refer to these two types as “comparison operators”

Standard algebraic equality or relational operator	Java equality or relational operator	Sample Java condition	Meaning of Java condition
<i>Equality operators</i>			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
<i>Relational operators</i>			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

# Precedence and associativity

Precedence ↓ High Low	Operators				Associativity	Type
	*	/	%		left to right	multiplicative
	+	-			left to right	additive
	<	<=	>	>=	left to right	relational
	==	!=			left to right	equality
	=				right to left	assignment

boolean isSame = 1+3 != 5\*3;

What is the value of isSame?





# Displaying formatted data with printf

```
public class Printer {  
    public static void main(String[] args) {  
        System.out.printf("%d\n", -123);  
        System.out.printf("Total: $%.2f\n", 1234.5678);  
        System.out.printf("%s World", "Hello");  
    }  
}
```

Printer prints the following text on the console:

-123

Total: \$1234.57

Hello World

- `System.out.printf()` method takes a **format string** (格式字符串) as an argument.
- The **format specifiers** (格式说明符) begin with a percent sign (%) and are followed by a character that represents the data type (e.g., **%s** is a placeholder for a string)



# Tabulating output with printf

radius	perimeter	area
1	6.2832	3.1416
2	12.5664	12.5664
3	18.8496	28.2743
4	25.1327	50.2655



How to generate beautiful tables using printf?



# Here is the magic code

```
double pi = Math.PI;  
System.out.printf("%-20s%-20s%-20s\n", "radius", "perimeter", "area");  
System.out.printf("%-20d%-20.4f%-20.4f\n", 1, 2*pi*1, pi*1*1);  
System.out.printf("%-20d%-20.4f%-20.4f\n", 2, 2*pi*2, pi*2*2);  
System.out.printf("%-20d%-20.4f%-20.4f\n", 3, 2*pi*3, pi*3*3);  
System.out.printf("%-20d%-20.4f%-20.4f\n", 4, 2*pi*4, pi*4*4);
```

Please decode the format strings by yourself.

Or visit the link below.

[https://www.cs.colostate.edu/~cs160/.Summer16/resources/Java\\_printf\\_method\\_quick\\_reference.pdf](https://www.cs.colostate.edu/~cs160/.Summer16/resources/Java_printf_method_quick_reference.pdf)



# Appendix – Terms

- ▶ Comment 注释 End-of-line comments 行末注释 Syntax error 语法错误
- ▶ String 字符串 Command window 命令窗口 Argument 参数 Cursor 光标
- ▶ Console 控制台 White-space characters 空白字符 Escape character 转义字符
- ▶ Carriage return 回车 Format string 格式字符串 Format specifier 格式说明符
- ▶ Primitive types 基本数据类型 Floating-point number 浮点数
- ▶ Decimal digits 小数位数 Unicode 万国码
- ▶ Standard input/output 标准输入输出 Assignment operator 赋值运算/操作符
- ▶ Prompt 提示符 Binary arithmetic operator 二元算术操作符
- ▶ Precedence 优先级 Associativity 结合性 Nested parentheses 嵌套的圆括号
- ▶ Equality operator 相同运算符 Relational operator 关系运算符