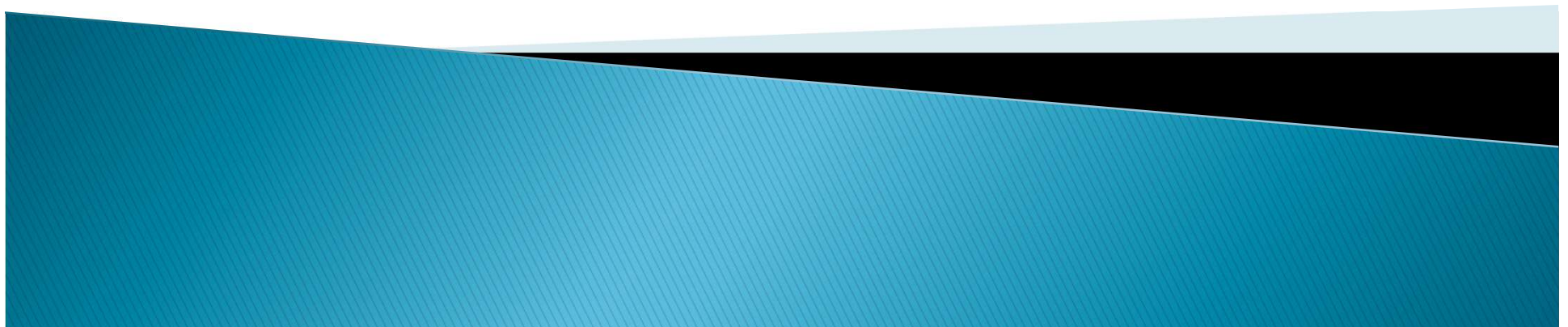




Introduction to Data Types

Xuetao Wei

weixt@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 (算术表达式求值顺序)
- ▶ Decision-making statements (决策/条件语句)



Outline

- ▶ Java's primitive types
- ▶ Arithmetic computation
- ▶ Evaluation order of arithmetic expressions
- ▶ Decision-making statements



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 (a sequence of bits)?
 - What operations can be done on the data?
 - How to store the data in memory?
- ▶ Computers only know about a few types of data: **numbers**, **booleans**, **characters (strings)**, **arrays**, **structures (objects)**



Primitive data types

- ▶ Complex data types are built from primitive data types, which are built-in and basic to a language implementation
- ▶ Java has eight primitive types
 - Integral types: `byte`, `short`, `int`, `long`
 - Floating-point types: `float`, `double`
 - The `boolean` data type
 - The `char` data type



Integral data types (Integers)

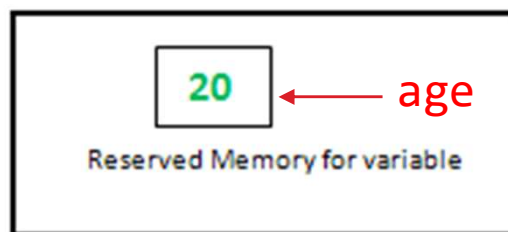
Type	Size	Range
byte	8 bits	-128 to +127
short	16 bits	-32,768 to +32,767
int	32 bits	(about) -2 billion to +2 billion
long	64 bits	(about) -10E18 to +10E18

Example: `int age = 20;`



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 as 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

<https://www.geeksforgeeks.org/variables-in-java/>



Floating-Point Numbers

- ▶ Computers represent real numbers (numbers that can contain a fractional part) using complex standard, such as the most popular **IEEE Floating-Point Standard**
- ▶ The term “*floating point*” is derived from the fact that there is no fixed number of digits before and after the decimal point; that is, the decimal point can float.
- ▶ There are also fixed-point representations: the number of digits before and after the decimal point is set (they can only handle a smaller range of numbers)

https://www.webopedia.com/TERM/F/floating_point_number.html



Floating-Point Numbers

Type	Size	Range
float	32 bits	-3.4E+38 to +3.4E+38
double	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 `double` type: **double-precision** floating-point number
 - A double has approximately 16 decimal digits
- ▶ The `float` type: **single-precision** floating-point number
 - A float has approximately 7 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



Think about this

- ▶ **Why computers cannot store real numbers of infinite precisions (such as the irrational number π)?**
- ▶ 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 implementations, could be 8 bits)
- ▶ Has only two possible values: **true** and **false**
- ▶ Often used as simple flags for tracking program conditions

Example: **boolean testResult = 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';
```

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

Prints: 南科大



Outline

- ▶ Java's primitive types
- ▶ **Arithmetic computation**
- ▶ Evaluation order of arithmetic expressions
- ▶ Decision-making statements



Adding two integers

```
import java.util.Scanner;
public class Addition {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number1 = 0;
        int number2 = 0;
        int sum = 0;
        System.out.print("Enter the first integer: ");
        number1 = input.nextInt();
        System.out.print("Enter the second integer: ");
        number2 = input.nextInt();
        sum = number1 + number2;
        System.out.printf("Sum is %d\n", sum);
        input.close();
    }
}
```



Adding two integers

```
import java.util.Scanner; ←
public class Addition {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number1 = 0;
        int number2 = 0;
        int sum = 0;
        System.out.print("Enter the first integer: ");
        number1 = input.nextInt();
        System.out.print("Enter the second integer: ");
        number2 = input.nextInt();
        sum = number1 + number2;
        System.out.printf("Sum is %d\n", sum);
        input.close();
    }
}
```




Import declaration

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

- ▶ Helps the compiler locate a class that is used in this program
- ▶ In Java, related classes are grouped into **packages**
- ▶ **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 = 0;
        int number2 = 0;
        int sum = 0;
        System.out.print("Enter the first integer: ");
        number1 = input.nextInt();
        System.out.print("Enter the second integer: ");
        number2 = input.nextInt();
        sum = number1 + number2;
        System.out.printf("Sum is %d\n", sum);
        input.close();
    }
}
```



Variable declaration statement

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

- ▶ Variable is a storage location, where a value can be stored for use in a program, paired with a symbolic name (an identifier)
- ▶ Variables must be declared with a **name** and a **type** before use
- ▶ A variable's **name** enables the program to access the value of the variable in memory
- ▶ A variable's **type** specifies what kind of information is stored at that location in memory



Variable declaration statement

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

- ▶ 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



Variable declaration statement

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

- ▶ 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 = 0;
        int number2 = 0;
        int sum = 0;
        System.out.print("Enter the first integer: ");
        number1 = input.nextInt();
        System.out.print("Enter the second integer: ");
        number2 = input.nextInt();
        sum = number1 + number2;
        System.out.printf("Sum is %d\n", sum);
        input.close();
    }
}
```

Declare variables of int type and initialize them
Same as `int number1 = 0, number2 = 0, sum = 0;`



Adding two integers

```
import java.util.Scanner;
public class Addition {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number1 = 0;
        int number2 = 0;
        int sum = 0;
        System.out.print("Enter the first integer: ");
        number1 = input.nextInt();
        System.out.print("Enter the second integer: ");
        number2 = input.nextInt();
        sum = number1 + number2;
        System.out.printf("Sum is %d\n", sum);
        input.close();
    }
}
```

Read the first number from user

System.out.print("Enter the first integer: ");
number1 = input.nextInt();

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

sum = number1 + number2;
System.out.printf("Sum is %d\n", sum);
input.close();



Receiving input with Scanner

```
System.out.print("Enter the first integer:"); // prompt  
number1 = input.nextInt(); // read number from user
```

- ▶ **Prompt** is a message that directs the user to take a specific action
- ▶ **System** is a class, why we don't import it like Scanner? Because it belongs to the `java.lang` package, which is imported by default
- ▶ 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**).



Receiving input with Scanner

```
System.out.print("Enter the first integer:"); // prompt  
number1 = input.nextInt(); // read number from user
```

- ▶ The result of the call to method `nextInt` will be assigned to the variable `number1` by the **assignment operator** `=`
- ▶ Note that `number1`'s initial value 0 will be replaced by the new value from `input.nextInt()`



Adding two integers

```
import java.util.Scanner;
public class Addition {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number1 = 0;
        int number2 = 0;
        int sum = 0;
        System.out.print("Enter the first integer: ");
        number1 = input.nextInt();
        System.out.print("Enter the second integer: ");
        number2 = input.nextInt();
        sum = number1 + number2;
        System.out.printf("Sum is %d\n", sum);
        input.close();
    }
}
```

Read the second
number from 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 = 0;
        int number2 = 0;
        int sum = 0;
        System.out.print("Enter the first integer: ");
        number1 = input.nextInt();
        System.out.print("Enter the second integer: ");
        number2 = input.nextInt();
        sum = number1 + number2; ←
        System.out.printf("Sum is %d\n", sum);
        input.close();
    }
}
```



Addition operation

sum = number1 + number2; An expression

- ▶ The computer **reads / loads** the values of number1 and number2 from memory, adds the two values and **stores** the result to the memory location represented by sum
- ▶ **Expressions:** Portions of statements that contain calculations



Adding two integers

```
import java.util.Scanner;
public class Addition {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number1 = 0;
        int number2 = 0;
        int sum = 0;
        System.out.print("Enter the first integer: ");
        number1 = input.nextInt();
        System.out.print("Enter the second integer: ");
        number2 = input.nextInt();
        sum = number1 + number2;
        System.out.printf("Sum is %d\n", sum); ←
        input.close();
    }
}
```



Formatted output

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

- ▶ Format specifier `%d` is a placeholder for an `int` value
- ▶ The letter 'd' stands for “decimal integer”



Adding two integers

```
import java.util.Scanner;
public class Addition {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number1 = 0;
        int number2 = 0;
        int sum = 0;
        System.out.print("Enter the first integer: ");
        number1 = input.nextInt();
        System.out.print("Enter the second integer: ");
        number2 = input.nextInt();
        sum = number1 + number2;
        System.out.printf("Sum is %d\n", sum);
        input.close(); ← Close the scanner after use (good practice)
    }
}
```



A sample execution

```
> java Addition
```

```
Enter the first integer: 72
```

```
Enter the second integer: 34
```

```
Sum is 106
```




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)



Outline

- ▶ Java's primitive types
- ▶ Arithmetic computation
- ▶ **Evaluation order of arithmetic expressions**
- ▶ Decision-making statements



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.



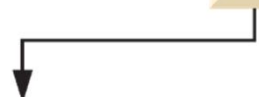
Associativity of operators

- ▶ 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



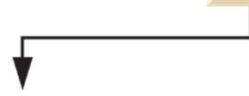
Step 2. $y = 10 * 5 + 3 * 5 + 7;$ (Leftmost multiplication)

$10 * 5$ is 50



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

$3 * 5$ is 15



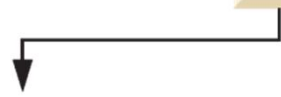
Step 4. $y = 50 + 15 + 7;$ (Leftmost addition)

$50 + 15$ 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)



There is a complete table of Java operator precedence on Blackboard



Outline

- ▶ Java's primitive types
- ▶ Arithmetic computation
- ▶ Evaluation order of arithmetic expressions
- ▶ **Decision-making statements**



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): **>**, **<**, **>=**, **<=**



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

- ▶ Relational operators $<$, $<=$, $>$, $>=$ have the same level of precedence. They are associated from left to right.
- ▶ Equality operators $==$, $!=$ have the same level of precedence. They are associated from left to right.
- ▶ Relational and equality operators have **lower precedence** than the five binary arithmetic operators
- ▶ In expression $1 + 3 != 5 * 3$, multiplication will be done first, then addition, the inequality check will be done at last



Decision-making statements

- ▶ **if selection statement** allows a program to make a **decision** based on a condition's value

```
if (condition) actions;
```

- ▶ In the above statement, the actions will be performed only if condition evaluates to true



Example

```
import java.util.Scanner;
public class Comparison {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int number1, number2;

        System.out.print("Enter first integer: ");
        number1 = input.nextInt();

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

        if(number1 == number2)
            System.out.printf("%d == %d\n", number1, number2);
        if(number1 != number2)
            System.out.printf("%d != %d\n", number1, number2);

        input.close();
    }
}
```



Sample executions

```
> java Comparison
```

```
Enter first integer: 72
```

```
Enter second integer: 34
```

```
72 != 34
```

```
> java Comparison
```

```
Enter first integer: 25
```

```
Enter second integer: 25
```

```
25 == 25
```



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