Programming Basics
- Java programming -

02. Basic Features 1

Wei Weng (翁 瑋)

Graduate School of Information, Production and Systems

Waseda University

1. Variables

- integer and floating-point data types
- variable declarations, initialization, and assignment
- comments

Java Identifier

- A Java **identifier** is a name consisting of letters and digits, the first of which must be a letter.
- Most Java programmers run words together to produce long identifiers. They initiate the first word with a lowercase character and each interior word with an uppercase character, as in calculateArea.
- You should adhere to this convention; otherwise, other programmers will find your programs difficult to understand.
- Eg. insertLine, delFile, changeData, etc.

Variable and Value

- A **variable** is a chunk of computer memory that contains a **value**. The **name** of a variable is an identifier that refers to the variable.
- Because every variable is typed, the Java compiler can **allocate** a memory chunk of the right size for each variable.
- When you tell the Java compiler the type of a variable, you are said to **declare** the variable.

Variable and Value - cont.

- Storing a value in the memory chunk allocated for a variable is called **variable assignment**.
- Accordingly, whenever Java places a value in such a memory chunk, the variable is said to be **assigned a value**, and the value is said to be **assigned to the variable**.
- To change the value of a variable, you use the **assignment operator**, =.

[0063, 0067]

Variable and Value - cont.

- All the integer and floating-point types are said to be **primitive types**, as is the boolean type and the character type.
- All other types are called **reference types**. The reference types include strings, arrays, and types you define by yourself.

Type	Bytes	Stores
int	4	integer
long	8	integer

Type	Bytes	Stores
float	4	floating-point number
double	8	floating-point number

[0069, 0072, 0076]

2. Arithmetic Expressions

- arithmetic expressions
- operator precedence and association
- type casting

Arithmetic Expressions

```
6 + 3     // Add, evaluating to 9
6 – 3     // Subtract, evaluating to 3
6 * 3     // Multiply, evaluating to 18
6 / 3     // Divide, evaluating to 2
6 + y     // Add, evaluating to 6 plus y's value
x – 3     // Subtract, evaluating to x's value minus 3
x * y     // Multiply, evaluating to x's value times y's value
x / y     // Divide, evaluating to x's value divided by y's value
5 / 3     // Divide, evaluating to 1 (quotient) same as 3/3 4/3
-5 / 3    // Divide, evaluating to -1
5 % 3     // Divide, evaluating to the remainder, 2
-5 % 3    // Divide, evaluating to the remainder, -2
5.0 / 3.0           // Divide, evaluating to 1.66667
```

[0084,0085]

Operator precedence and association

Each box contains operators that have equal precedence. The top box contains the highest-precedence operators:

Operator level	Associativity
( ) [ ] .	left to right
! ++ --	right to left
* / %	left to right
+ -	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= &= ^=	right to left

[Appendix A]

Arithmetic Expressions

- The precedence of the negation operator, -, is higher than that of +, -, *, or /:

  ```
  - 6 * 3 / 2   // Equivalent to ((- 6) * 3) / 2 = -9
  ```

- When an arithmetic expression contains values that have a mixture of data types, it is called a **mixed expression**. When Java evaluates a mixed expression, it first uses the given values to produce a set of values that have identical types. Then, Java performs the prescribed arithmetic.
- Thus, when given a mixed expression that multiplies a floating-point number by an integer, Java first produces a floating-point number from the integer, and then multiplies.

[0090,0092]

type casting

- If you want to tell Java explicitly to convert a value from one type to another, rather than relying on automatic conversion, you **cast** the expression. To cast, you prefix the expression with the name of the desired type in parentheses.
- If, for example, i is an int and d is a double, you can cast i to a double and d to an int as follows:

```
(double) i          // A double expression
(int) d             // An int expression
(double) i * d      // Equivalent to i * d
```

Note that the original types of the i and d variables remain undisturbed: i remains an int variable, and d remains a double variable.

[0090,0092]

3. Predicates

- predicates and Boolean values
- Combination of Boolean Expressions

Predicates

Operators and methods that return value representing true or false are called **predicates**.

Java offers several operator predicates that test the relationship between pairs of numbers:

Predicate	Purpose
==	Are two numbers equal?
!=	Are two numbers not equal?
>	Is the first number greater than the second?
<	Is the first number less than the second?
>=	Is the first number greater than or equal to the second?
<=	Is the first number less than or equal to the second?

[00370]

Combination of Boolean Expressions

- The **and operator**, &&, returns true if *both* of its operands evaluate to true. The **or operator**, ||, returns true if *either* of its operands evaluates to true.
- The following expression, for example, evaluates to true only if the value of the length variable is between 60 and 90:

  60 < length && length < 90

[0404]

Combination of Boolean Expressions

- In && expressions, the left-side operand is evaluated first: If the value of the left-side operand is false, then the right-side operand is ignored completely, and the value of the && expression is false. Of course, if both operands evaluate to true, the value of the && expression is true.
- In || expressions, the left-side operand also is evaluated first: If the left-side operand evaluates to true, nothing else is done, and the value of the || expression is true; if both operands evaluate to false, the value of the || expression is false.

Predicates

- the value returned by a predicate must be a **Boolean value**; that must be either true or false.
- You can declare variables to have **boolean type**. You can assign either of the **literal Boolean values**, true or false, to such variables.

Note: A common error is to write a single equal-to sign, =, the assignment operator, when you intend to check for equality. Be sure to remember that the equality predicate is written as a double equal-to sign, ==.

[00372,0374]

Predicates

- You now know that, whenever the character ! is followed immediately by the character =, the two characters together denote the inequality operator.
- The not operator is a unary operator that converts true into false, and false into true. Thus, the value of !false is true and !true is false. Similarly, the value of !(6 == 3) is true. Also, the value of !(6 != 3) is false.

```
public class Demonstrate {
        public static void main (String argv[]) {
                boolean b; b = (2 + 2 == 4);
                System.out.println(b);
        }
}
--- Result ---
 true
```

Predicates

If you want to determine whether two class instances are the same instance, you use **==**, which is a operator rather than the method **equals**. Note that the equals method determines whether two instances are the equivalent instance, rather than same instances:

```
public class Demonstrate {
        public static void main (String argv[]) {
                String s1 = new String ("aaa");
                String s2 = new String ("aaa");
                String s3 = new String ("bbb");
                System.out.println(s1.equals(s2));
                System.out.println(s1.equals(s3));
                System.out.println(s1 == s2);
                System.out.println(s1 == s3);
                s2 = s1;
                System.out.println(s1.equals(s2));
                System.out.println(s1 == s2);
        }
}
```

[0378]