Object-Oriented Programming

Java Programming Essentials

Computer Science and Technology
United International College

Review

Can you write HelloWorld without looking at the code now?

Java Programming Essentials

- **Identifiers and Naming**
- **Keywords**
- **Variable Declarations and Assignments**
- **Data Types**
- **Variable Initialization**
- **Type Casting**
- **Arithmetic Expressions and Operators**

Punctuations

'	Single Quote
"	Double Quote
( )	Brackets / Parentheses
[ ]	Square Brackets
{ }	Curly Brackets / Braces
;	Semi-colon
.	Dot / Full-stop
,	Comma
/	Slash
\	Backslash

Identifiers

The name of a variable or other item (class, method, object, etc.) defined in a program.

- A Java identifier must NOT start with a digit, and all the characters must be letters, digits, "`$`" or "`_`"
  - `W_12`, `HelloWorld`, `_983`, `$bS5_c7` Correct
  - `4W2`, `class`, `Data#`, `98.3`, `Hell world` Not Ok

- Java is a **case-sensitive** language: `Rate`, `rate`, and `RATE` are the names of three different variables.

Choice of Identifiers

- Easy to understand and remember.
  - e.g. `numberOfEnquiries`, `Trees`, `timeToLive`, `name`, `address`, `isOK`
- May use multiple words (without space!).
  - e.g. `myBirthday`, `numberOfStudents`
- Don’t be lazy!
  - e.g. `i`, `j`, `k`, `a`, `b`, `c`, `d`, `e`, …

Keywords

Keywords and **Reserved** words: these are identifiers having a predefined meaning in Java.

- Do not use them to name anything else!

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

Java Library Identifiers

Java Library Identifiers: defined in libraries required by the Java language standard.

- Although they can be redefined, this could be confusing and dangerous if doing so since it would change their standard meaning.

```
System      String      println
```

Naming Conventions

Camel Case

- **Variable and Method** names should begin with a lowercase letter. Indicate "word" boundaries with an uppercase letter and restrict the remaining characters to digits and lowercase letters.

  `topSpeed` **`bankRate1`** **`timeOfArrival`**

- **Class** names should begin with an uppercase letter and, otherwise, adhere to the rules above.

  `FirstProgram` **`MyClass`** **`String`**

Variable Declarations

```
int numberOfBeans;
double myBalance, totalWeight;
```

- Every variable in a Java program must be ***declared*** before it is used (just like in C).
  - A variable declaration tells the compiler what kind of data (type) will be stored in the variable.
  - The type of the variable is followed by one or more variable names separated by commas, and terminated with a semicolon.
  - Variables are typically declared just before they are used or at the start of a block (indicated by an opening brace `{`).
  - Basic types in Java are called *primitive types.*

Declaration of a Local Variable

```
String myName = "Chunyan Ji";
double myHeight = 1.62, rainfall = 3.4;
```

- “Type” refers to both *primitive type* or *class*.
- In general, four basic forms of declarations:

```
1.<type name> identifier;
2.<type name> identifier1, identifier2, … ;
3.<type name> identifier = <initial value>;
4.<type name> identifier1 = <initial value1>,
      identifier2 = <initial value2>, …;
```

- Tip: always initialize the variable, set an initial value!

Write and Show Time

- Write a class called `Room`.
- Define four different types of variables in it.
- Show it to your neighbor.

Discuss with your neighbor!

What is a Type?

- A ***type*** restricts the **kind and range of values** a data item or an expression could take.
  - e.g. Sex → [**`'M'`**, **`'F'`**]
    Phone → **`13211113231`**

- Types must match!
  - e.g.

```
String name = "Lily Lin";
String myID = "123";
```

`123` is an **`integer`**

`"123"` is a **`String`**

Data Types

- 数据类型 (Data type)
  - 基本数据类型 (primitive type)
    - 数值型
      - 整数类型(byte, short, int, long)
      - 浮点类型(float, double)
    - 字符型(char)
    - 布尔型（boolean）
  - 引用数据类型
    - 类(class)
    - 接口(interface)
    - 数组 (array)

Primitive Types

Java defined Four categories and Eight different primitive types:

- Boolean: `boolean`
- Character: `char`
- Integer: `byte`, `short`, `int`, `long`
- Floating point: `float`, `double`

Primitive Types

Data Type	Default Value (for fields)	Size (in bits)	Minimum Range	Maximum Range
byte	0	Occupy 8 bits in memory	-128	+127
short	0	Occupy 16 bits in memory	-32768	+32767
int	0	Occupy 32 bits in memory	-2147483648	+2147483647
long	0L	Occupy 64 bits in memory	-9223372036854775808	+9223372036854775807
float	0.0f	Occupy 32-bit IEEE 754 floating point	1.40129846432481707e-45	3.40282346638528860e+38
double	0.0d	Occupy 64-bit IEEE 754 floating point	4.94065645841246544e-324d	1.79769313486231570e+308d
char	'\u0000'	Occupy 16-bit, unsigned Unicode character		0 to 65,535
boolean	false	Occupy 1- bit in memory	NA	NA

Boolean

- The `boolean` type is used for evaluating logical conditions. Usually used in flow of control.
- Has two values only: `false` and `true`.
- Not like the C language: a non-zero value is not equivalent to `true`. The value `0` is not equivalent to `false`.

```
boolean flag;
flag = true;
if(flag) {
        // do something
}
```

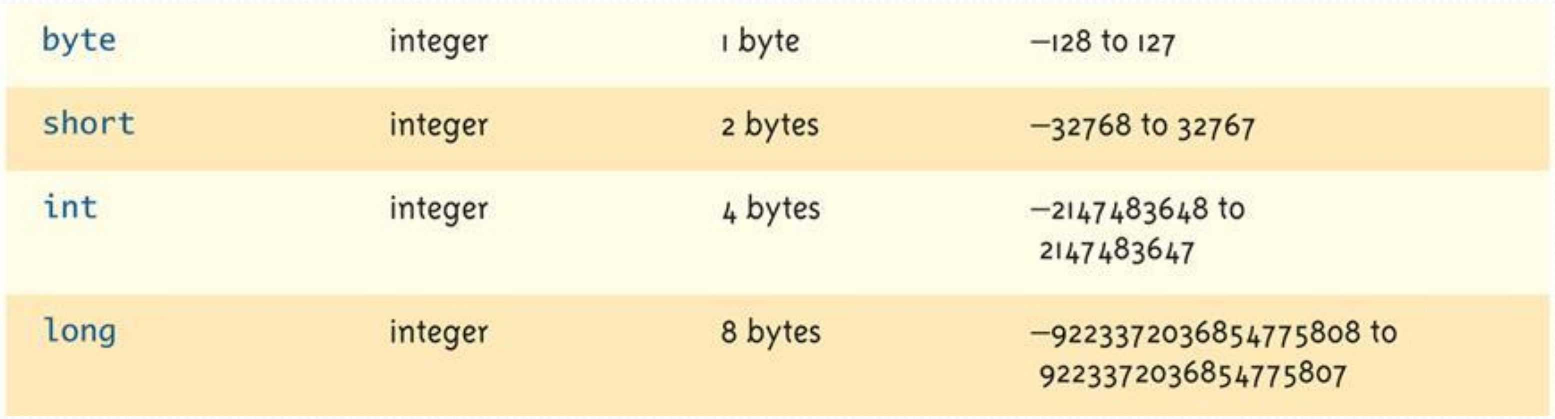
Character

- A `char` variable stores a single character.
- Character literals are delimited by single quotes:
  - `'a'    'X'    '7'    '$'    ','    '\n'`
- Example declarations:
  - `char topGrade = 'A';`
  - `char terminator = ';';`

Do you see the difference between **`char`** and **`String`**?

Integer

- The **`integer`** types are for numbers without factional parts. Negative values are allowed.
- Long integer numbers have a suffix ‘**`L`**’ or ‘**`l`**’.
  - **`int i = 600;`** // correct
  - **`long l = 88888888881;`** // wrong without suffix ‘`l`’



byte	integer	1 byte	−128 to 127
short	integer	2 bytes	−32768 to 32767
int	integer	4 bytes	−2147483648 to 2147483647
long	integer	8 bytes	−9223372036854775808 to 9223372036854775807

Integer

- Decimal number:
  - E.g.: `12`, `-23`, `0`
- Octal number:
  - Begin with "`O`". E.g.: `O12`
- Hexadecimal number:
  - Begin with "`OX`" or "`Ox`". E.g.: `Ox12`

Floating numbers

- Floating-point Number literals are considered to be of type **`double`** by *default*.

```
double d1 = 3.14159;                    // ok
double d2 = 3e8;                        // ok
double d3 = -0.27e-5;                   // ok
float  f1 = 3.14159;                    // not ok
```

- Adding a suffix F/ f to the number changes this default:

```
float  f2 = 3.14159F;             // ok
float  f3 = -0.27e-5f;            // ok
```

Default Type of numbers (int and double)

- ***Integer literals*** (integer numbers appearing in program code) are generally treated as `int` type.
  - e.g. `i = 7 * j / (-9 + k);`
  - e.g. `Math.cos(30 * Math.PI / 180);`
- ***Real number/ floating point number literals*** (numbers with decimal point or in exponential notation appearing in program code) are generally treated as `double` type.
  - e.g. `p = 7.0 * q + .958;`
  - e.g. `Math.toRadians(3e-1 * 100.);`

3.0×10^{-1}

Example: Initial variables

```
public class PrimaryType {
        public static void main (String args []) {
                boolean b = true;       //boolean type
                int x, y=8;             // int type
                float f = 4.5f;   '     // float type
                double d = 3.1415;      //double type
                char c;                 //char type
                c = '\u0031';           //initial char type
                x = 12;                 //initial int type
         }
}
```

Write and Show Time

- Write 8 variables with 8 primitive types and use random numbers as their initial values.

  For example: `int i = 5;`

- Show it to your neighbor.

Assignment Compatibility

- In general, the value of one type cannot be stored in a variable of another type:

```
int intVariable = 2.99; //Illegal
```

  - The above example results in a type mismatch because a `double` value cannot be stored in an `int` variable.
- However, there are exceptions to this:

```
double doubleVariable = 2;
```

  - For example, an `int` value can be stored in a `double` type.

Assignment Compatibility

- More generally, a value of any type in the following list can be assigned to a variable of any type that appears to the right of it:

$$\begin{tikzcd}
\texttt{byte} \arrow[r] & \texttt{short} \arrow[r] & \texttt{int} \arrow[r] & \texttt{long} \arrow[r] & \texttt{float} \arrow[r] & \texttt{double} \\
\texttt{char} \arrow[urr] & & & & &
\end{tikzcd}$$

  - Note that as your move down the list from left to right, the range of allowed values for the types becomes larger.

Assignment Compatibility cont.

- An explicit ***type cast*** is required to assign a value of one type to a variable whose type appears to the left of it on the above list (e.g., **`double`** to **`int`**):

```
double aDoubleNum = 2.23;
int aInteger = (int)aDoubleNum;
```

- Note that in Java an **`int`** cannot be assigned to a variable of type **`boolean`**, nor can a **`boolean`** be assigned to a variable of type **`int`**.

Discussion Time

Check the following codes carefully, and find the place where may cause compile error or overflow mistake:

```
public void method() {
        int i = 1, j;
        float f1 = 0.1; float f2 = 123;
        long l1 = 12345678, l2 = 8888888888;
        double d1 = 2e20, d2 = 124;
        byte b1 = 1, b2 = 2, b3 = 129;
        j = j + 10;
        i = i / 10;
        i = i * 0.1;
        char c1 = 'a', c2 = 125;
        byte b = b1 - b2;
        char c = c1 + c2 - 1;
        float f3 = f1 + f2;
        float f4 = f1 + f2 * 0.1;
        double d = d1 * i + j;
        float f = (float)(d1 * 5 + d2);
}
```

Operators

Category of operators:

- Arithmetic: `+ - * / % ++ --`
- Relational: `< > <= >= == !=`
- Logical: `! & | && || ^`
- Conditional: `bool_expression ? true_case : false_case`
- Short-hand: `++ -- += -= *=` …
- Assignment: `=` (to store a value)
- Bitwise Operators: `& | ^ ~ >> << >>>` (unsigned shift right)
- Concatenate : `+`

Arithmetic Operators

Addition/ Sum:	`a + b`
Subtraction/ Difference:	`a - b`
Multiplication/ Product:	`a * b`
Division/ Quotient:	`a / b`
Remainder (of integer division):	`a % b`
Negation/ Minus:	`-a`

Arithmetic Operators and Expressions

- If an arithmetic operator is combined with **`int`** operands, then the resulting type is **`int`**.
- If an arithmetic operator is combined with one or two **`double`** operands, then the resulting type is **`double`**.
- If different types are combined in an expression, then the resulting type is the right-most type on the following list that is found within the expression:

$$\begin{tikzcd}
\texttt{byte} \arrow[r] & \texttt{short} \arrow[r] & \texttt{int} \arrow[r] & \texttt{long} \arrow[r] & \texttt{float} \arrow[r] & \texttt{double} \\
\texttt{char} \arrow[urr] & & & & &
\end{tikzcd}$$

  - Exception: If the type produced should be **`byte`** or **`short`** (according to the rules above), then the type produced will actually be an **`int`**.

Type in Expression

- Type promotion / conversion:

  byte → short → int → long → float → double

```
byte   b = 104;
float  f = 3.14159f;
double d = 35 * f - 29.7 / b;
```

```
[double ←int * float - double / byte]
```

Integer and Floating-Point Division

- When one or both operands are a floating-point type, division results in a floating-point type:

  `15.0 / 2` evaluates to `7.5`

- When both operands are integer types, any fractional part is discarded and the division results in an integer type:

  `15 / 2` evaluates to `7`

- Be careful to make at least one of the operands a floating-point type if the fractional portion is needed!

The `%` Operator -- remainder

- The `%` operator is used with operands of type **`int`** to compute the remainder of an integer division:

  `15 / 2` evaluates to the integer quotient `7`

  `15 % 2` evaluates to the integer remainder `1`

String Concatenation

- "`+`" can be used to concatenate two string:
  - `int id = 80 + 90;` // addition
  - `String s = "hello" + "world";` // concatenation
- If one of the operands is a String type, the other one will be converted to String automatically.
- When using `System.out.println(??)`, the content `??` will be printed as a `String` type.

Shorthand Assignment Statements

Example:	Equivalent To:
`count += 2;`	`count = count + 2;`
`sum -= discount;`	`sum = sum - discount;`
`bonus *= 2;`	`bonus = bonus * 2;`
`time /= rushFactor;`	`time = time / rushFactor;`
`change %= 100;`	`change = change % 100;`

Shorthand Assignment Statements

Operation	Example	Equivalent Expression
`+=`	`a += b`	`a = a + b`
`-=`	`a -= b`	`a = a - b`
`*=`	`a *= b`	`a = a * b`
`/=`	`a /= b`	`a = a / b`
`%=`	`a %= b`	`a = a % b`
`&=`	`a &= b`	`a = a & b`
`|=`	`a |= b`	`a = a | b`
`^=`	`a ^= b`	`a = a ^ b`
`<<=`	`a <<= b`	`a = a << b`
`>>=`	`a >>= b`	`a = a >> b`
`>>>=`	`a >>>= b`	`a = a >>> b`

Conditional Operators

- `x ? y : z`
- `x` is a `boolean` expression. If `x` is `true`, then return the value of `y`, else return the value of `z`.
- E.g.:

```
int score = 80;
String type = score < 60 ? "failed" : "pass";
System.out.println("type = " + type);
```

Increment(++) and Decrement (--)Operators

```
public class Test {
   public static void main(String[] args) {
        int x = 10, y = 20;
        int i = y++;
        System.out.print("i = " + i);
        System.out.println(" y = " + y);
        i = ++y;
        System.out.print("i = " + i);
        System.out.println(" y = " + y);
        i = --x;
        System.out.print("i = " + i);
        System.out.println(" x = " + x);
        i = x--;
        System.out.print("i = " + i);
        System.out.println(" x = " + x);
   }
}
```

Output:

```
i = 20  y = 21
i = 22  y = 22
i = 9  x = 9
i = 9  x = 8
```

Note:

- ++(--)
- When before variable, calculation first;
- When after variable, assign the value first;

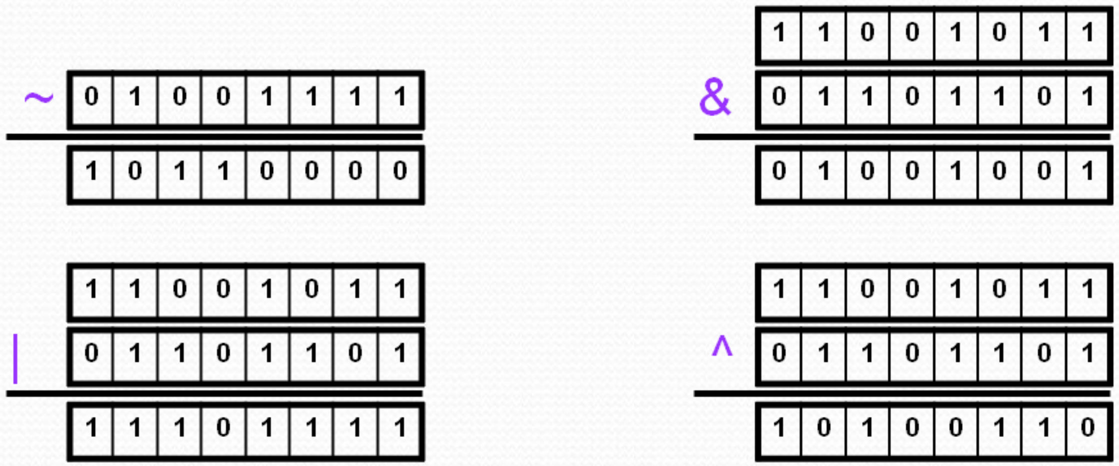
Logical Operators

- Boolean operators
  - `&` (and), `|` (or), `!` (negate), `^` (exclusive-or)
  - operate on `boolean` values
  - result is a `boolean` value
  - `&&` (logical and), `||` (logical or) are short-circuiting so better.

a	b	!a	a&b	a|b	a^b	a&&b	a||b
true	true	false	true	true	false	true	true
true	false	false	false	true	true	false	true
false	true	true	false	true	true	false	true
false	false	true	false	false	false	false	false

Bitwise Operators

- Bitwise operators:
  - Work with any of the integer types;
  - Work directly the bits that make up the integers.
  - & ("and"), | ("or"), ^ ("xor"), ~ ("not")



Parentheses and Precedence Rules

- An expression can be *fully parenthesized* in order to specify exactly what sub expressions are combined with each operator.
- If some or all of the parentheses in an expression are omitted, Java will follow *precedence* rules to determine, in effect, where to place them.

Always include parentheses in practice!

Operator Precedence Rules

high

Separator	. ( ) { } ; ,
Associative	Operators
R to L	++ -- ~ ! (data type)
L to R	* / %
L to R	+ -
L to R	<< >> >>>
L to R	< > <= >= instanceof
L to R	== !=
L to R	&
L to R	^
L to R	|
L to R	&&
L to R	||
R to L	?:
R to L	= *= /= %= += -= <<= >>= >>>= &= ^= |=

low

Summary

- Identifier
- Primitive type
- Type Casting
- Variable declaration and initialize
- Operators and expression