JAC444 - Lecture 1

Introduction to

Java Programming Language

Segment 4

Overview of the Java Language

In this segment you will be learning about:

- Numeric Operators in Java
- Type Conversion
- If, For, While, Do-While Statements
- Labeled Break and Labeled Continue
- Arrays and Strings

Numeric operators in Java

Prec	Operator	Description	*Example*
1	++	Increment by 1(or 1.0)	*k++*
1	--	Decrement by 1(or 1.0)	*k--*
1	+	Unary plus	*+value*
1	-	Unary minus	*-value*
2	*	Multiplication	*x * y*
2	/	Division	*x / y*
2	%	Modulo	*x % y*
3	+	Addition	*x + y*
3	-	Subtraction	*x - y*
5	<	Less than	*x < y*
5	>	Greater than	*x > y*
5	<=	Less than/equal	*x <= y*
5	>=	Greater than/equal	*x >= y*
6	==	Equals (identical values)	*x == y*
6	!=	Is not equal to	*x != y*
13	op=	op assignment(+ =, - =, *=, etc)	*x += y*

Bitwise and logical operators

Pre	Operator	Type	Description	Example
1	~	Integral	Unary bitwise complement	~x
1	!	Logical	Unary logical complement	!b
4	<<	Integral	Left shift	x << 2
4	>>	Integral	Right shift (keep sign)	x >> 5
4	>>>	Integral	Right shift (zero fill)	x >>> 3
5	instanceof	Obj type	Tests class membership	o instanceof X
6	==	Object	Equals (same object)	x == y
6	!=	Object	Unequal (different object)	x != y
7	&	Integral	Bitwise AND	x & y
7	&	Logical	Logical AND	b1 & b2
8	^	Integral	Bitwise XOR	x ^ y
8	^	Logical	Logical XOR	b1 ^ b2
9	|	Integral	Bitwise OR	x | y
9	|	Logical	Logical OR	b1 | b2

Bitwise and logical operators

Prec	Operator	Type	Description	*Examp.*
10	&&	Logical	Logical AND (short-circuit)	***b1 && b2***
11	||	Logical	Logical OR (short-circuit)	***b1 || b2***
12	?:	Logical	Conditional (ternary)	***b ? x : y***
13	=	Variable,any	Assignment	***x = 1***
13	<<=	Binary	Left shift with assignment	***x <<= 2***
13	>>=	Binary	Right shift with assignment	***x =>> 3***
13	>>>=	Binary	Right shift, zero fill, assign	***x =>> 4***
13	&=	Binary	Bitwise AND with assign	***x &= y***
13	&=	Logical	Logical AND with assign	***b1 &=b2***
13	|=	Binary	Bitwise OR with assignment	***x |= y***
13	|=	Logical	Logical OR with assignment	***b1 |=b2***
13	^=	Binary	Bitwise XOR with assignment	***x ^= y***
13	^=	Logical	Logical XOR with assignment	***b1 ^=b2***

Bitwise logic rules

AND	0	1
0	0	0
1	0	1

$0 \mathbin{\&} 0 = 0$ $\quad$ $0 \mathbin{\&} 1 = 0$

$1 \mathbin{\&} 0 = 0$ $\quad$ $1 \mathbin{\&} 1 = 1$

OR	0	1
0	0	1
1	1	1

$0 \mid 0 = 0$ $\quad$ $0 \mid 1 = 1$

$1 \mid 0 = 1$ $\quad$ $1 \mid 1 = 1$

XOR	0	1
0	0	1
1	1	0

$0 \mathbin{\char`\^} 0 = 0$ $\quad$ $0 \mathbin{\char`\^} 1 = 1$

$1 \mathbin{\char`\^} 0 = 1$ $\quad$ $1 \mathbin{\char`\^} 1 = 0$

Bitwise operations on short primitives (16 bit integers)

Binary	Operation	Decimal	Hex
0000 0000 0101 0100	op1	84	0x0054
0000 0001 0100 0111	op2	327	0x0147
0000 0000 0100 0100	op1 & op2	68	0x0044
0000 0001 0101 0111	op1 | op2	343	0x0157
0000 0001 0001 0011	op1 ^ op2	275	0x0113
1111 1110 1011 1000	~op2	-328	0xFEB8

Results of some bit-shifting

0000	0000	0000	0000	0000	0000	0110	0011	starting a = 99
0	0	0	0	0	0	6	3	a = 0x00000063
0000	0000	0000	0000	0000	0011	0001	1000	after a << 3 (792)
0	0	0	0	0	3	1	8	a = 0x00000318
0000	0000	0000	0000	0000	0000	0001	1000	after a >> 2 (24)
0	0	0	0	0	0	1	8	a = 0x00000018
1111	1111	1111	1111	1111	1111	1001	1101	stating b = -99
F	F	F	F	F	F	9	D	b = 0xFFFFFF9D
1111	1111	1111	1111	1111	1111	1111	1001	after b >> 4 = -7
F	F	F	F	F	F	F	9	b = 0xFFFFFFF9
0000	0000	0000	1111	1111	1111	1111	1111	after b >>> 12 = 1048575
0	0	0	F	F	F	F	F	b = 0x000FFFFF

Type Conversions

- Java is a strong typed language

- Implicit conversion for primitive value: *any numeric value can be assigned to any numeric value whose type supports a larger range of values.*

  ```
  byte → short → int → long → float → double
  ```

- Explicit conversion – casting.
  - *boolean* type doesn't allow any casting at all.
  - A *char* can be cast to any integer type and vice versa excepting to a short type. When *chart* is cast to *int* type upper bits are filled with zeros.
  - Attention: interger types are converted by chopping off the upper bits.

    If the larger integer has a value outside the range off the smaller type, dropping the upper bits changes the value, including possibly changing sign.

Ex:

```
short x = -129;
byte y = (byte)x;    What is the value of y ???
```

if, if-else, if else –if else

```
1.  if  (boolean-expression) {
        statements;
    }
```

```
2.  if  (boolean-expression) {
        statements;
    } else {
        statements;
    }

3.  if  (boolean-expression) {
        statements;
    } else if  (boolean-expresion ) {
        statements;
    } else {
        statements;
        }
```

for Statement

- A for statement should have the following form:

```
for (initialization; condition; update) {
   statements;
}
for (k = 0, flag; k < 10 && flag; k++ ) {
   . . .
}
```

Enhanced for loop

```
for (variable : Collection ) {
   . . .
}
```

while, do - while Statements

- ***while, do-while*** and for control looping are classified as *iteration statements.*

```
while (condition) {
    statements;
}
```

```
do {
    statements;
} while (condition);
```

break - Labeled break

- A break “drops out of the bottom” of the loop. The break statement with no label attempts to transfer control to the innermost enclosing ***switch**, **for**, **while*** or ***do-while*** of immediately enclosing statement.

- A labeled break drops out of the bottom of the end of the loop denoted by the label.

  Ex:

```
out:  for (int i = 0; i < 10; i++ ) {
      for (int k = 0; k < 10; k++) {
         if (i   ==   k)
             break out;
      }
      System.out.println(i);
   }
```

continue – Labeled continue

- A plain continue goes to the top of the innermost loop and continues
- A labeled continue goes to the label and re-enters the loop right after that label

Ex: Calculates the factorials of odd number

```
outerLoop:    for (int i = 0; i < limit; i++ ) {
              for (int k = 2; k < i; k++) {
                 if (i % 2)
                     continue outerLoop;
                 factory *= i;
              }
          }
```

switch

- The switch is classified as a *selection statement*

```
switch (integral-selector) {
    case integral-value1:
        statements;
    break;
    default:
        statements;
}
```

Integral-selector is an expression that produces an integral value.

The switch compares the result of integral-selector to each integral-value.

If it finds a match, the corresponding statement (simple or compound) executes. If no match occurs, the default statement executes.

Array

- An array is simply a sequence of either objects or primitives, all the same type and accessed together under one identifier name.

- Arrays are implicit extensions of *Object.*
- Arrays are defined and used with square-brackets *indexing operator* [ ]

```
    int[] integerArray  =  new int[3];
```

- A Java array is guaranteed to be initialized and cannot be accessed outside of its range.

```
    for (int  k  =  0;  k <  integerArray.length;   k++)
       integerArray[k]  =  k;
```

- Creating an array of objects, one is really creating an array of references, and each of those references is automatically initialized to null.

```
Student[] room = new Student[3];
```

Strings

- Java strings are standard objects with built-in language support.
- The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of String class
- Strings are constant; their values cannot be changed after they are created. Stringbuffer class supports mutable strings. Because String objects are immutable they can be shared

```
String str = "abc";
```

is equivalent to:

```
char data[] = {'a', 'b', 'c'};
String str = new String(data);
```

String Examples

Here are some more examples of how strings can be used:

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

int length(); returns the length of this string.

char charAt(int index); returns the character at the specified index.

Conclusions

After completion of this lesson you should know:

- How to write a simple Java Program
- How to work with primitives: integers and floats
- To use Java tools like: compiler and interpreter