C4 Operators

Java 8 Associate

1Z0-808

Link

https://mylearn.oracle.com/ou/exam/java-se-8-programmer-i-1z0-808/105037/110679/170387
https://docs.oracle.com/javase/specs/jls/se8/html
https://docs.oracle.com/javase/tutorial
http://www.java2s.com

https://enthuware.com
https://github.com/carlosherrerah/1Z0-808

JUAN CARLOS HERRERA HERNANDEZ
carlos.herrera@upa.edu.mx

Contenido

0. CERTIFICATION SUMMARY 3
1. Assignment Operators 4
2. Relational Operators 5
2.1. Equality for Reference Variables 5
3. instanceof Comparison 7
4. Arithmetic Operators 8
4.1. The Remainder (%) Operator 8
4.2. String Concatenation Operator 8
4.3. Increment and Decrement Operators 8
5. Conditional Operator (Ternary) 10
6. Logical Operators 11
6.1. Bitwise Operators 11
6.2. Short-Circuit Logical Operators (&& ||) 11
6.3. Logical Operators (not Short-Circuit) 11
6.4. Logical Operators ^ and ! 11
7. Operator Precedence 12

>>

0. CERTIFICATION SUMMARY

CERTIFICATION OBJECTIVES

Using Java Operators
• Use Parentheses to Override Operator Precedence
• Test Equality Between Strings and Other Objects Using == and equals( )
• *Create if and if/else and ternary constructs*

The logical operators (`&&`, `||`, `&`, `|`, and `^`) can be used only to evaluate two `boolean` expressions.
The difference between `&&` and `&` is that the `&&` operator will not bother testing the right operand if the left evaluates to `false`, because the result of the `&&` expression can never be `true`.
The difference between `||` and `|` is that the `||` operator will not bother testing the right operand if the left evaluates to `true` because the result is already known to be `true` at that point.

The `&&` and `||` operators are known as short-circuit operators.
The `&` and `|` operators always evaluate both operands.

The `==` operator can be used to compare values of primitives, but it can also be used to determine whether two reference variables refer to the same object.

The `instanceof` operator is used to determine whether the object referred to by a reference variable passes the IS-A test for a specified type.

The `++` and `--` operators will be used throughout the exam, and you must pay attention to whether they are prefixed or postfixed to the variable being updated.

The `+` operator can be used to add two numeric primitives together or to perform a concatenation operation if either operand is a `String`.

>>

1. Assignment Operators

Remember that a reference variable isn't an object; it's a way to *get* to an object. (We know all you C++ programmers are just dying for us to say, "it's a pointer," but we're not going to.)

Don't spend time preparing for topics that are no longer on the exam! The following topics have NOT been on the exam since Java 1.4:
Bit-shifting operators: ***5 <<1 =10;***
Bitwise operators: ***5 & 3= 1; 5\|3 =7; 5^3 =6; ~5 = -6***
Two's complement ***-5 comp2 = 1111 1011***
Divide-by-zero stuff

without using a compound operator:	Now, with compound operators:
`y = y - 6;` `x = x + 2 * 5;`	`y -= 6;` `x += 2 * 5;`

>>

2. Relational Operators

- Relational operators always result in a `boolean` value (`true` or `false`).
- There are six relational operators: >, >=, <, <=, ==, and `!=`. The last two (`==` and `!=`) are sometimes referred to as *equality operators*.
- When comparing characters, Java uses the Unicode value of the character as the numerical value.
- Equality operators
  - There are two equality operators: `==` and `!=`.
  - Four types of things can be tested: numbers, characters, booleans, and reference variables.
- When comparing reference variables, `==` returns `true` only if both references refer to the same object.

boolean b = 10 > 9;

2.1. Equality for Reference Variables

Two reference variables can refer to the same object, as the following code snippet demonstrates:

```
import javax.swing.JButton;
class CompareReference {
   public static void main(String[] args) {
      JButton a = new JButton("Exit");
      JButton b = new JButton("Exit");
      JButton c = a;
      System.out.println("Is reference a == b? " + (a == b)); // false
      System.out.println("Is reference a == c? " + (a == c)); // true
   }
}
```

The equals() Method in Class String The `equals()` method in class `String` has been overridden. When the `equals()` method is used to compare two strings, it will return true if the strings have the same value, and it will return false if the strings have different values. For `String`'s `equals()` method, values ARE case sensitive.

The equals() Method in Class Object The `equals()` method in class `Object` works the same way that the `==` operator works. If two references point to the same object, the `equals()` method will return true. If two references point to different objects, even if they have the same values, the method will return false.

```
class Budgie {
  public static void main(String[] args) {
    Budgie b1 = new Budgie();
    Budgie b2 = new Budgie();
    Budgie b3 = b1;

    String s1 = "Bob";
    String s2 = "Bob";
    String s3 = "bob";                  // lower case "b"

    System.out.println(b1.equals(b2));  // false, different objects
    System.out.println(b1.equals(b3));  // true, same objects
    System.out.println(s1.equals(s2));  // true, same values
    System.out.println(s1.equals(s3));  // false, values are case sensitive
  }
}
```

Equality for enums

```
class EnumEqual {
   enum Color {RED, BLUE} // ; is optional
   public static void main(String[] args) {
      Color c1 = Color.RED; Color c2 = Color.RED;
      if(c1 == c2) { System.out.println("=="); }
      if(c1.equals(c2))System.out.println("dot equals");
   }
}
```

==

dot equals

>>

3. instanceof Comparison

- `instanceof` is for reference variables only; it checks whether the object is of a particular type.
- The `instanceof` operator can be used only to test objects (or `null`) against class types that are in the same class hierarchy.
- For interfaces, an object passes the `instanceof` test if any of its superclasses implement the interface on the right side of the `instanceof` operator.
- ***Remember that arrays are objects, even if the array is an array of primitives.***

First Operand (Reference Being Tested)	instanceof Operand (Type We're Comparing the Reference Against)	Result
null	Any class or interface type	false
Foo instance	Foo, Bar, Face, Object	true
Bar instance	Bar, Face, Object	true
Bar instance	Foo	false
Foo [ ]	Foo, Bar, Face	compiler error
Foo [ ]	Object	true
Foo [1]	Foo, Bar, Face, Object	true

```
interface Face { }
class Bar implements Face{ }
class Foo extends Bar { }

public class Prueba {
    public static void main(String[] args) {
        Foo f = new Foo();
        Foo[] arr = new Foo[10];
        arr[1] = new Foo();

        if (f instanceof Foo) {System.out.println("f Es una instancia de Foo");}
        if (arr instanceof Foo[]) {System.out.println("arr es una instancia de Foo[]");}
        if (arr instanceof Object) {System.out.println("arr es una instancia de Object");}
        if (arr[1] instanceof Bar) {System.out.println("arr[1] es una instancia de Bar");}
    }
}
```

>>

4. Arithmetic Operators

- The four primary math operators are add (+), subtract (–), multiply (*), and divide (/).
- The remainder (a.k.a. modulus) operator (%) returns the remainder of a division.
- Expressions are evaluated from left to right, unless you add parentheses, or unless some operators in the expression have higher precedence than others.
- The *, /, and % operators have higher precedence than + and –.

4.1. The Remainder (%) Operator

The remainder operator can also be used with noninteger operands in Java.

En Java, el resultado de % tiene el mismo signo que el dividendo.

4.2. String Concatenation Operator

- If either operand is a `String`, the + operator concatenates the operands.
- If both operands are numeric, the + operator adds the operands.

```
String a = "String";
int b = 3;
int c = 7;
System.out.println(b + c + a);   // 10String
System.out.println(a + b + c);   // String37
System.out.println(a + (b + c)); // String10
```

```
String s = "123";
s += "45";
s += 67;
System.out.println(s); // 1234567
```

4.3. Increment and Decrement Operators

- Prefix operators (e.g. `--x`) run before the value is used in the expression.
- Postfix operators (e.g., `x++`) run after the value is used in the expression.
- In any expression, both operands are fully evaluated *before* the operator is applied..
- Variables marked `final` cannot be incremented or decremented.

```
int players = 10;
System.out.println("players online: " + players++);                 // 10
System.out.println("The value of players is " + players);           // 11
System.out.println("The value of players is now " + ++players);  // 12
```

```
int x = 2; int y = 3;
if ((y == ++x) || (x < ++y)) {
   System.out.println("x = " + x + " y = " + y);
}
```

x = 3 y = 3

```
int x = 2; int y = 3;
if ((y == ++x) | (x < ++y)) {
   System.out.println("x = " + x + " y = " + y);
}
```

x = 3 y = 4

```
final int x = 5;
int y = x++; // Error de Compilacion
```

>>

5. Conditional Operator (Ternary)

- Returns one of two values based on the state of its `Boolean` expression.
- Returns the value after the `?` if the expression is `true`.
- Returns the value after the `:` if the expression is `false`.

x = (boolean expression) ? value to assign if true : value to assign if false

```
// operador ternario que regresa: positivo, negativo, neutro  de un valor de x
String s = (x > 0) ? "positivo" : (x < 0) ? "negativo" : "neutro";
System.out.println(s);
```

>>

6. Logical Operators

- The exam covers six "logical" operators: `&`, `|`, `^`, `!`, `&&`, and `||`.
- Work with two expressions (except for `!`) that must resolve to Boolean values.
- The `&&` and `&` operators return `true` only if both operands are `true`.
- The `||` and | operators return `true` if either or both operands are `true`.
- The `&&` and `||` operators are known as short-circuit operators.
- The `&&` operator does not evaluate the right operand if the left operand is `false`.
- The `||` does not evaluate the right operand if the left operand is `true`.
- The `&` and `|` operators always evaluate both operands.
- The `^` operator (called the "logical XOR") returns `true` if exactly one operand is `true`.
- The `!` operator (called the "inversion" operator) returns the opposite value of the boolean operand it precedes.

6.1. Bitwise Operators

5 & 3 = 1; 5|3 =7; 5^3 =6; ~5 = -6;

Bit-shifting operators: 5 <<1 =10;

BITWISE OPERATORS ARE NOT ON THE Java 6, Java 7, or Java 8 EXAM!

6.2. Short-Circuit Logical Operators (&& ||)

```
boolean b7 = false && true;
int x = 2; int y = 3;
if ((y == ++x) || (x < ++y)) {
  System.out.println("x=" + x + " y=" + y);
}
```

```
x=3 y=3
```

```
int z = 5;
if (++z > 5 || ++z > 6) z++;
System.out.println("z = " + z);
// z = 7 after this code
```

6.3. Logical Operators (not Short-Circuit)

```
boolean b7 = false & true;
int x = 2;  int y = 3;
if ((y == ++x)  |  (x < ++y)) {
  System.out.println("x = " + x + " y = " + y);
}
```

```
x = 3 y = 4
```

```
int z = 5;
if (++z > 5 | ++z > 6) z++;
System.out.println("z = " + z);
// z = 8 after this code
```

6.4. Logical Operators ^ and !

```
System.out.println("xor " + ((2<3) ^ (4>3)));
```

produces the output: `xor false`

```
if(!(7 == 5)) { System.out.println("not equal"); }
```

>>

7. Operator Precedence

- In real life, use parentheses to clarify your code, and force Java to evaluate expressions as intended.
- For the exam, memorize Table 4-2 to determine how parentheses-free code will be evaluated.

Types of Operators	Symbols	Example Uses
Unary operators	-, !, ++, -- ~ (type)	-7 * 4, !myBoolean
Multiplication, division, modulus	*, /, %	7 % 4
Addition, subtraction	+, -	7 + 4
shift Relational operators	<, >, <=, >= instanceof	y > x
Equality operators	==, !=	y != x
Logical operators (& beats |) Bitwise	&,^|	myBool & yourBool
Short-circuit (&& beats ||)	&&, ||	myBool || yourBool
Ternary Assignment operators	=, +=, -=	X += 5;

UMARELSA

There are three important general rules for determining how Java will evaluate expressions with operators:

- When two operators of the same precedence are in the same expression, Java evaluates the expression from left to right.
- When parts of an expression are placed in parentheses, those parts are evaluated first.
- When parentheses are nested, the innermost parentheses are evaluated first. In other words, solve the first closed parenthesis.

```
int a = 5;
int b = 10;
int c = 15;
boolean result;

// Ejemplo de precedencia de operadores   U M A R E L S A
result = a +  b *  c >  b &&  b - a <  c || a == 5;   // true
         5 + 10 * 15 > 10 && 10 - 5 < 15 || 5 == 5
         5 +   150   >
          150        > 10 &&
                     t    &&    5   < 15
                     t    &&        t
                          t              ||    t
                                         t
```

```
int x = 1;
int y = 2;
int z = 3;

int result2 = x > y ?  x + y : x - y * z;    // -5
```

>>