6. Operators

Java

Summer 2008

Instructor: Dr. Masoud Yaghini

Operators

- Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.

Outline

- Simple Assignment Operator
- Arithmetic Operators
- Unary Operators
- Equality and Relational Operators
- Logical Operators
- Conditional Operator
- Type Comparison Operator
- Bitwise and Bit Shift Operators
- Operator Precedence
- References

Simple Assignment Operator

The Simple Assignment Operator

- One of the most common operators that you'll encounter is the simple assignment operator "=".
- You saw this operator in the Bicycle class; it assigns the value on its right to the operand on its left:
  - int cadence = 0;
  - int speed = 0;
  - int gear = 1;

Arithmetic Operators

The Arithmetic Operators

- The Arithmetic Operators:
  - `+` additive operator (also used for String joining)
  - `-` subtraction operator
  - `*` multiplication operator
  - `/` division operator
  - `%` remainder operator

- The only symbol that might look new to you is "%", which divides one operand by another and returns the remainder as its result.

ArithmeticDemo Program

```
class ArithmeticDemo {
    public static void main (String[] args){
        int result = 1; // result is now 1
        System.out.println(result);
        result = 1 + 2; // result is now 3
        System.out.println(result);
        result = result - 1; // result is now 2
        System.out.println(result);
        result = result * 2; // result is now 4
        System.out.println(result);
        result = result / 2; // result is now 2
        System.out.println(result);
        result = result + 8; // result is now 10
        result = result % 7; // result is now 3
        System.out.println(result);
    }
}
```

Compound Assignments

- You can also combine the arithmetic operators with the simple assignment operator to create compound assignments.
- For example, `x+=1;` and `x=x+1;` both increment the value of `x` by 1.

+ operator for String concatenation

- The + operator can also be used for concatenating (joining) two strings together, as shown in the following ConcatDemo program:

```
class ConcatDemo {
    public static void main(String[] args){
        String firstString = "This is";
        String secondString = " a concatenated string.";
        String thirdString = firstString+secondString;
        System.out.println(thirdString);
    }
}
```

Unary Operators

The Unary Operators

- The unary operators:
  - `+` Unary plus operator; indicates positive value
  - `-` Unary minus operator; negates an expression
  - `++` Increment operator; increments a value by 1
  - `--` Decrement operator; decrements a value by 1
  - `!` Logical complement operator; inverts the value of a boolean

- The unary operators require only one operand

UnaryDemo Program

```
class UnaryDemo {
    public static void main(String[] args){
        int result = +1; // result is now 1
        System.out.println(result);
        result--; // result is now 0
        System.out.println(result);
        result++; // result is now 1
        System.out.println(result);
        result = -result; // result is now -1
        System.out.println(result);
        boolean success = false;
        System.out.println(success); // false
        System.out.println(!success); // true
    }
}
```

The Unary Operators

- The increment/decrement operators can be applied before (prefix) or after (postfix) the operand.
- The code result++; and ++result; will both end in result being incremented by one.
- The only difference is that the prefix version (++result) evaluates to the incremented value, whereas the postfix version (result++) evaluates to the original value.
- If you are just performing a simple increment/decrement, it doesn't really matter which version you choose. But if you use this operator in part of a larger expression, the one that you choose may make a significant difference.

PrePostDemo Program

```
class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        System.out.println(i); // "4"
        ++i;
        System.out.println(i); // "5"
        System.out.println(++i); // "6"
        System.out.println(i++); // "6"
        System.out.println(i); // "7"
    }
}
```

Relational Operators

The Relational Operators

- **The Relational Operators:**

  == equal to

  != not equal to

  > greater than

  >= greater than or equal to

  < less than

  <= less than or equal to

- Keep in mind that you must use "==", not "=", when testing if two primitive values are equal.

ComparisonDemo Program

```
class ComparisonDemo {
    public static void main(String[] args) {
        int value1 = 1;
        int value2 = 2;
        if (value1 == value2)
            System.out.println("value1 = value2");
        if (value1 != value2)
            System.out.println("value1 != value2");
        if (value1 > value2)
            System.out.println("value1 > value2");
        if (value1 < value2)
            System.out.println("value1 < value2");
        if (value1 <= value2)
            System.out.println("value1 <= value2");
    }
}
```

ComparisonDemo Program

- ComparisonDemo program output:

  value1 != value2

  value1 < value2

  value1 <= value2

Logical Operators

The Logical Operators

- **Logical Operators:**
  - && Conditional-AND
  - || Conditional-OR

ConditionalDemo1 Program

```
class ConditionalDemo1 {
    public static void main(String[] args) {
        int value1 = 1;
        int value2 = 2;
        if ((value1 == 1) && (value2 == 2))
            System.out.println("value1 is 1 AND value2 is 2");
        if ((value1 == 1) || (value2 == 1))
            System.out.println("value1 is 1 OR value2 is 1");
    }
}
```

Conditional Operator

Conditional Operator

- ?: which can be thought of as shorthand for an if-then-else statement.
- This operator is known as the ternary operator because it uses three operands.
- Use the ?: operator instead of an if-then-else statement if it makes your code more readable;

Conditional Operator

- In the following example, this operator should be read as:
  - "If someCondition is True, assign the value of value1 to result.
  - Otherwise, assign the value of value2 to result."

```
class ConditionalDemo2 {
    public static void main(String[] args) {
        int value1 = 1;
        int value2 = 2;
        int result;
        boolean someCondition = true;
        result = someCondition ? value1 : value2;
        System.out.println(result);
    }
}
```

Type Comparison Operator

The Type Comparison Operator instanceof

- The instanceof operator compares an object to a specified type.
- You can use it to test if an object is an instance of a class, an instance of a subclass, or an instance of a class that implements a particular interface.
- The InstanceofDemo program defines:
  - a parent class (named Parent),
  - a simple interface (named MyInterface), and
  - a child class (named Child) that inherits from the parent and implements the interface.

InstanceofDemo Program

```
class InstanceofDemo {
    public static void main(String args[]) {
        Parent obj1 = new Parent();
        Parent obj2 = new Child();
        System.out.println("obj1 instanceof Parent: " +
                (obj1 instanceof Parent));
        System.out.println("obj1 instanceof Child: " +
                (obj1 instanceof Child));
        System.out.println("obj1 instanceof MyInterface: " +
                (obj1 instanceof MyInterface));
        System.out.println("obj2 instanceof Parent: " +
                (obj2 instanceof Parent));
        System.out.println("obj2 instanceof Child: " +
                (obj2 instanceof Child));
        System.out.println("obj2 instanceof MyInterface: " +
                (obj2 instanceof MyInterface));
    }
}

class Parent { }
class Child extends Parent implements MyInterface { }
interface MyInterface { }
```

InstanceofDemo Program

- Output:

```
obj1 instanceof Parent: true
obj1 instanceof Child: false
obj1 instanceof MyInterface: false
obj2 instanceof Parent: true
obj2 instanceof Child: true
obj2 instanceof MyInterface: true
```

Bitwise and Bit Shift Operators

Bitwise and Bit Shift Operators

- Bitwise and Bit Shift Operators:
  - ~ unary bitwise complement
  - << signed left shift
  - >> signed right shift
  - >>> unsigned right shift
  - & bitwise AND
  - ^ bitwise exclusive OR (XOR)
  - | bitwise inclusive OR (OR)

Operator Precedence

Operator Precedence

- Java has an established precedence hierarchy to determine the order in which operators are evaluated.
- Operators with higher precedence are evaluated before operators with relatively lower precedence.

Operator Precedence

Operators	Precedence
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Operator Precedence

- The closer to the top of the table an operator appears, the higher its precedence.
- Operators on the same line have equal precedence.
- If two operations have the same precedence, the one on the left in the actual expression is handled before the one on the right.

Operator Precedence

- Given differing orders of precedence.
  - result = 14 + 8 / 2; // Divide first
  - / higher precedence than + and result is 18.
- Precedence can be forced using parentheses.
  - result = (14 + 8) / 2; // Add first
  - + is forced first by parentheses and result is 11.
- Given the same order of precedence.
  - result = 12 / 2 * 3; // Divide first
  - / is first (L to R), then * and result is 18.
- Adding a unary operator -.
  - result = 12 / -(-3 + 1) * 3; // Negation first
  - - is first (R to L), then / (L to R), then * and result is 18.

Operator Precedence

- Given increment/decrement operators. Assume int a = 5;.
  - result = a + (--a) + a;
    - Proceed L to R a = 5, then 5 + (--a) + a.
    - Now do a = a - 1= 4 → a, then 5 + 4 + a → 9 + a.
    - Finally, a = 4 and 9 + 4 → result is 13.
  - result = a + (a--) + a;
    - Proceed L to R a = 5, then 5 + (a--) + a.
    - Now do 5 + 5 + a = 10 + a, then do a = a - 1 = 4 → a.
    - Finally 10 + a = 10 + 4 → result is 14.

References

References

- S. Zakhour, S. Hommel, J. Royal, I. Rabinovitch, T. Risser, M. Hoeber, **The Java Tutorial: A Short Course on the Basics**, 4th Edition, Prentice Hall, 2006. (Chapter 3)

The End