Programming in Java

Introduction to JAVA

Lecture : Operators and Expressions

Contents

- Naming convention
- Primitive data types
- Operators in Java
- Arithmetic Operators
- Bitwise Operators
- Relational Operators
- Logical Operators
- Data Types

Identifiers and Naming Conventions

Names of things that appear in the program are called identifiers.

Rules:

- ✓ An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).
- ✓ An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- ✓ An identifier cannot be a reserved word.
- ✓ An identifier cannot be true, false, or null.
- ✓ An identifier can be of any length.

*Java is case sensitive

*Do not name identifiers with the **$** character. By convention, the **$** character should be used only in mechanically generated source code

*Descriptive identifiers make programs easy to read

Identifiers and Naming Conventions

Names of things that appear in the program are called identifiers.

Rules:

- ✓ An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).
- ✓ An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- ✓ An identifier cannot be a reserved word.
- ✓ An identifier cannot be true, false, or null.
- ✓ An identifier can be of any length.

Which of the following identifiers are valid?

applet, Applet, a++, —a, 4#R, $4, #44, apps

Type Conversion, Casting and Promotion

1. Numeric Type Conversions

Consider the following statements:

```
byte i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

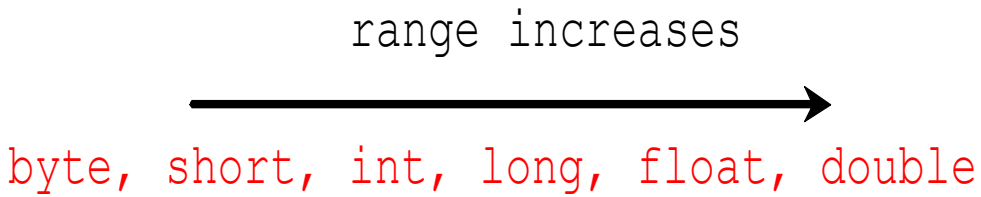
When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.

Type Conversion, Casting and Promotion

1. Numeric Type Conversions

The result of 1/2 is 0 but result of 1.0/2 is 0.5



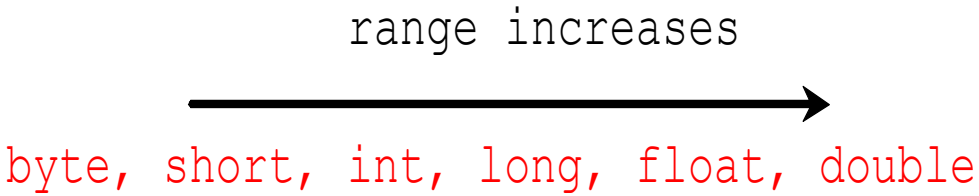
Casting converts a value of one data type into a value of another data type

* widening a type – 1. Casting a variable of a type with a small range to a variable of a type with a larger range can be performed automatically (implicit casting)
* narrowing a type – 1. Casting a variable of a type with a large range to a variable of a type with a smaller range must be performed explicitly (explicit casting)

Type Conversion, Casting and Promotion

1. Numeric Type Conversions

The result of 1/2 is 0 but result of 1.0/2 is 0.5



Implicit casting

double d = 3; (type widening)

Explicit casting

int i = (int)3.0; (type narrowing)

int i = (int)3.9; (Fraction part is truncated)

What is wrong? int x = 5 / 2.0; It would work if int x = (int)(5 / 2.0);

Operators in Java

- Java's operators can be grouped into following four categories:

1. Arithmetic
2. Bitwise
3. Relational
4. Logical.

Arithmetic Operators

- Used in mathematical expressions.
- Operands of the arithmetic operators must be of a numeric type.
- Most operators in Java work just like they do in C/C++.
- We can not use them on boolean types, but we can use them on char types, since the char type in Java is a subset of int.

Arithmetic Operators

Operator	Result
+	Addition
-	Subtraction (also unary minus)
*	Multiplication
/	Division
%	Modulus
++	Increment
+=	Addition assignment
-=	Subtraction assignment
*=	Multiplication assignment
/=	Division assignment
%=	Modulus assignment
- -	Decrement

```
class ArithmeticOp
{   public static void main(String args[])
    {
    int a = 2 + 1; int b = a * 2; int c = b / 4; int d = c - a; int e = -d;
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
    System.out.println("d = " + d);
    System.out.println("e = " + e);
    double p = 2 + 1; double q = p * 2; double r = q / 4; double s = r -a;  double t = -
    q;
    System.out.println(“p = " + p);
    System.out.println(“q = " + q);
    System.out.println(“r = " + r);
    System.out.println(“s = " + s);
    System.out.println(“t = " + t);
    }
}
```

Modulus Operator (%)

- returns the remainder of a division operation.
- can be applied to floating-point types as well as integer types.
- This differs from C/C++, in which the **%** can only be applied to integer types.

Arithmetic Assignment Operators

• Used to combine an arithmetic operation with an assignment.

Thus, the statements of the form

var = var op expression;

can be rewritten as

var op= expression;

• In Java, statements like

a = a + 5;

can be written as

a += 5;

Similarly:

b = b % 3; can be written as b %= 3;

Increment and Decrement

- ++ and the - - are Java's increment and decrement operators.
- The increment operator increases its operand by one.
  - x++; is equivalent to x = x + 1;
- The decrement operator decreases its operand by one.
  - y--; is equivalent to y = y – 1;

- They can appear both in
  - *postfix form, where they* follow the operand, and
  - *prefix form, where they precede the operand.*

- In the prefix form, the operand is incremented or decremented before the value is obtained for use in the expression.

  Example: x = 19; y = ++x;

  Output: y = 20 and x = 20

- In postfix form, the previous value is obtained for use in the expression, and then the operand is modified.

  Example: x = 19; y = x++;

  Output: y = 19 and x = 20

Bitwise Operators

- These operators act upon the individual bits of their operands.
- Can be applied to the integer types, long, int, short, char, and byte.

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
|=	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Bitwise Logical Operators

- The bitwise logical operators are
  - ~ (NOT)
  - & (AND)
  - | (OR)
  - ^ (XOR)

The Left Shift

- The left shift operator,<<, shifts all of the bits in a value to the left a specified number of times.
- It has this general form:
  - value << num

- Looking at the same operation in binary shows more clearly how this happens:
  - 01000001 65
  - << 2
  - 00000100 4

The Right Shift

- The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times.
- Its general form is shown here:
  - value >> num

- Looking at the same operation in binary shows more clearly how this happens:
  - 00100011 35
  - >> 2
  - 00001000 8

The Unsigned Right Shift

- In these cases, to shift a zero into the high-order bit no matter what its initial value was. This is known as an unsigned shift.

- Here is the same operation in binary form to further illustrate what is happening:
  - 11111111 11111111 11111111 11111111 –1 in binary as an int
  - >>>24
  - 00000000 00000000 00000000 11111111 255 in binary as an int

Bitwise Operator Compound Assignments

- For example, the following two statements, which shift the value in a right by four bits, are equivalent:
  - a = a >> 4;
  - a >>= 4;

- Likewise, the following two statements, which result in a being assigned the bitwise expression a OR b , are equivalent:
  - a = a | b;
  - a |= b;

Relational Operators

- The relational operators determine the relationship that one operand has to the other.

Operator	Result
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

- the following code fragment is perfectly valid:
  - int a = 4;
  - int b = 1;
  - boolean c = a < b;

Boolean Logical Operators

- The Boolean logical operators shown here operate only on boolean operands.

Operator	Result
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND assignment
|=	OR assignment
^=	XOR assignment
==	Equal to
!=	Not equal to
?:	Ternary if-then-else

A	B	A | B	A & B	A ^ B	~ A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Short-Circuit Logical Operators

- the OR operator results in true when A is true , no matter what B is. Similarly, the AND operator results in false when A is false, no matter what B is.

- For example,

  – **if (denom != 0 && num / denom > 10)**

- Since the short-circuit form of AND (&&) is used, there is no risk of causing a run-time exception when denom is zero.

The Assignment Operator

- The assignment operator is the single equal sign, =.
  - var = expression ;
- Here, the type of var must be compatible with the type of expression.
- The assignment operator does have one interesting attribute that you may not be familiar with: it allows you to create a chain of assignments. For example, consider this fragment:
  - int x, y, z;
  - x = y = z = 100; // set x, y, and z to 100
- This fragment sets the variables x , y, and z to 100 using a single statement.

The ? Operator

- Java includes a special ternary (three-way)operator that can replace certain types of if-then-else statements.

- The ? has this general form:
  - **expression1 ? expression2 : expression3**
- Here,expression1 can be any expression that evaluates to a boolean value.
- If expression1 is true , then expression2 is evaluated; otherwise, expression3 is evaluated.

- Here is an example of the way that the ? is employed:
  - **ratio = denom == 0 ? 0 : num / denom ;**
- When Java evaluates this assignment expression, it first looks at the expression to the left of the question mark.
- If denom equals zero, then the expression between the question mark and the colon is evaluated and used as the value of the entire ? expression.
- If denom does not equal zero, then the expression after the colon is evaluated and used for the value of the entire ? expression.
- The result produced by the? operator is then assigned to ratio.

Operator Precedence

Highest			
( )	[]	.	
++	--	~	!
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|			
&&			
||			
?:			
=	Op=		
Lowest			