Basic Java- Operators & Assignments

Persistent University

PERSISTENT

Objectives:

- At the end of this module, you will be able to understand:
  - Types of Operators
  - Precedence and Associativity
  - Conversions
  - Promotions
  - Different Types of Operators

Operators

- Operator: produces a new value with 1, 2 or 3 operands.
  - Unary Operator
  - Binary Operator
  - Ternary Operator

```
a = x + y - 2/2 + z;
```

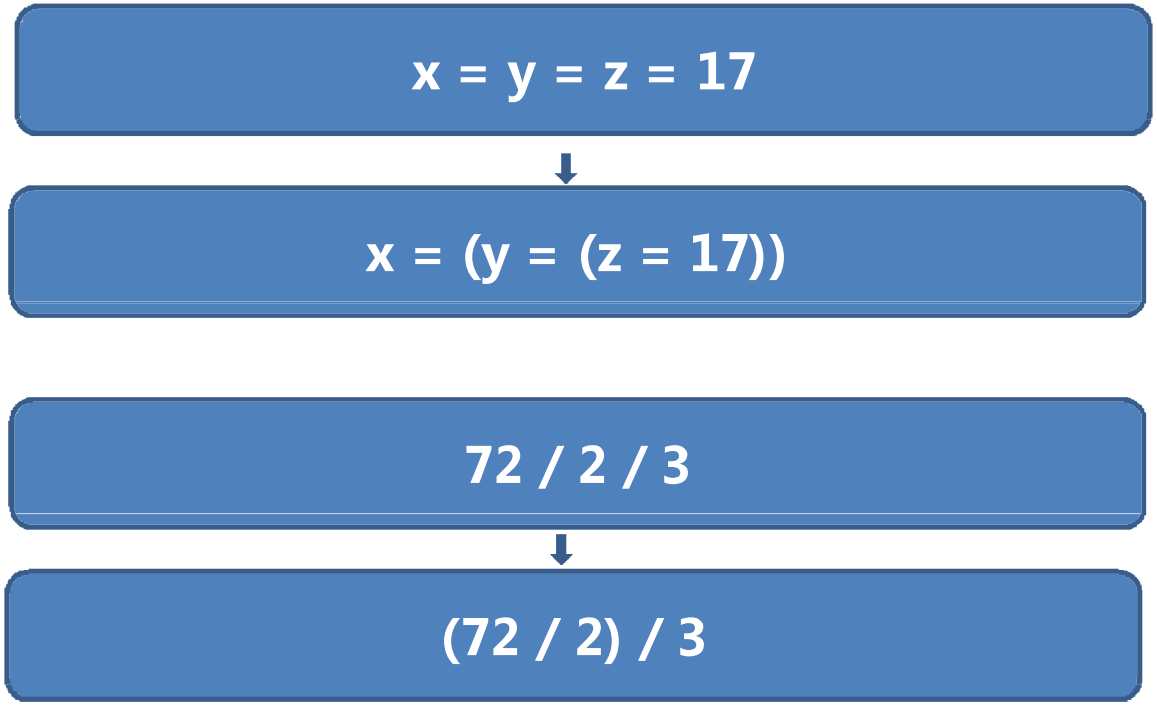
Precedence

Postfix operators	[ ] . (parameters) exp++ exp—
Unary prefix operators	++exp - -exp +exp –exp ~ !
Unary prefix creation & case	new (type)
Multiplicative	* / %
Additive	+ -
Shift	<< >> >>>
Relational	< <= > >= instanceof
Equality	== !=
Bitwise/logical AND	&
Bitwise/logical XOR	^
Bitwise/logical OR	|
Conditional AND	&&
Conditional OR	||
Conditional	?:
Assignment	= += -+ *= /= %= <<= >>= >>>= &= ^= |=

High Order of precedence

Associativity

- which operator should be applied first if there are two operators with same precedence.
  - Left – Right
  - Right - Left



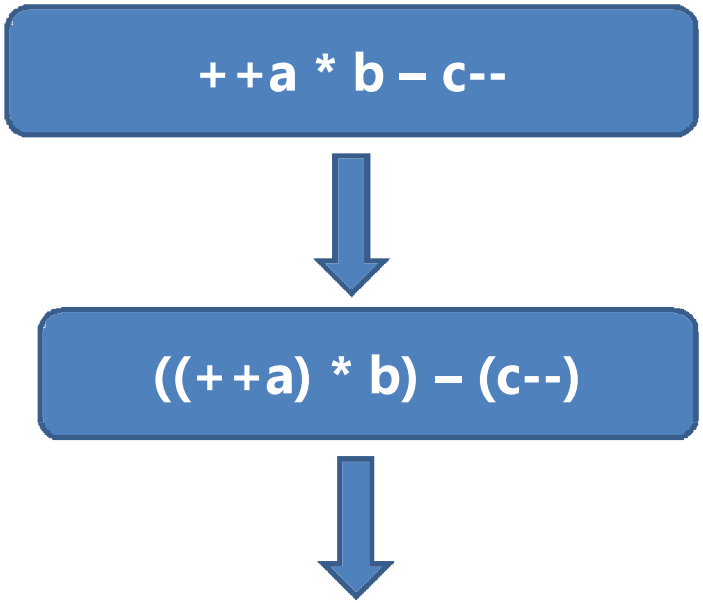
PERSISTENT

Precedence and Associativity

Postfix operators (L-R)	[ ] . (parameters) exp++ exp—
Unary prefix operators (R-L)	++exp - -exp +exp –exp ~ !
Unary prefix creation & case (R-L)	new (type)
Multiplicative (L-R)	* / %
Additive (L-R)	+ -
Shift (L-R)	<< >> >>>
Relational (L-R)	< <= > >= instanceof
Equality (L-R)	== !=
Bitwise/logical AND (L-R)	&
Bitwise/logical XOR (L-R)	^
Bitwise/logical OR (L-R)	|
Conditional AND (L-R)	&&
Conditional OR (L-R)	||
Conditional (R-L)	?:
Assignment (R-L)	= += -+ *= /= %= <<= >>= >>>= &= ^= |=

High Order of precedence

Example Expression



Conversions

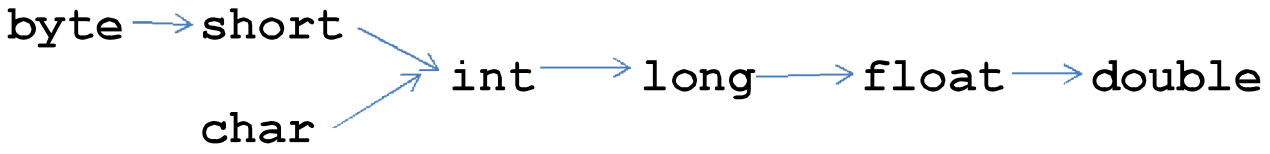
- Converting from one type to another.
  - Primitive Types
  - Reference Types

- Boolean values cannot be cast to other data values, and vice versa.

- Reference literal “null” can be cast to any reference type.

Widening Conversion

- Narrower to broader data type without loss of information.
- Usually done implicitly(cast can be used redundantly.)




```
int a = 100;
long b = a;
```

```
// Implicit cast, an
int value always fits
in a long
```

```
int a = 100;
long b = (long)a;
```

```
double d = 100L;
```

Narrowing Conversion

- Broader to Narrower data type with loss of information.
- Typically require cast.
- Cast required Conversion from char to short/byte, coz' char is unsigned.

```
float a = 100.001f;
int b = a;
```

```
float a = 100.001f;
int b = (int)a;
```

Conversions: Examples

```
long l = 130L;
byte b = (byte)l;
```

compiles & runs fine o/p is -126

```
char a = 0x892;    // hexadecimal literal
char b = 982;    // int literal
char c = (char) 70000;  // 70000 is out of char range
char d = (char) -98;  // Ridiculous, but legal
```

Unary Numeric Promotion

- If operand type is narrower than int, it is implicitly converted to int; otherwise not.
- Only byte, short, char are converted.

```
byte b=1;
int result = ++b;
```

```
byte b=1;
byte result = b + 1;
```

Compile time error

```
byte b=1;
byte result = (byte) (b + 1);
```

Binary Promotion

- implicitly applies widening conversions, to have the broadest numeric type (always at least int).

```
byte a1=1;
byte a2=1;
byte b1 = a1+a2;
```

```
int b1 = a1+a2;
```

```
byte b1 = (byte)a1+a2;
```

```
int i=1, j=2;
float f=1.1f;
double d1=1.2;
double d2 = i+j+f+d1;
```

Assignment Operator

```
<variable> = <expression>
```

- previous value of destination is overwritten.
- has lowest precedence.
- Assigning Primitive Values:

```
int i;
i=10;
```

```
int i, j;
i = j = 15;
```

```
//(i = (j = 15))
```

Numeric Type Conversion in Assignment

- Implicit widening conversion.
- Implicit narrowing conversion, if all
- Source is a constant expression of byte /short /char /int.
- Destination type is byte / short/ char.
- Value of source is in range of destination type.

```
short s1 = 10;
short s2 = 'a';
```

```
int i = 10;
short s1 = i;
```

Compile time error

- all other narrowing conversions explicitly require a cast.

Arithmetic Operators

Unary	+ Addition	- Subtraction		R
Binary	* Multiplication	/ Division	% Remainder	L
		+ Addition	- Subtraction	

7/5 => 1
7%5 => 2

```
System.out.println("put two & two together
                    & get "+ 2 + 2);
```

o/p- "put two & two together & get 22"

Increment and Decrement Operators

- **Increment Operator ++**
  - **Prefix Increment (++i**): Adds 1 to i & then uses new value of I
  - **Postfix Increment (i++):** uses current value of i first & then adds 1
- **Decrement Operator --**
  - **Prefix Decrement (--i): Subtracts 1 from i & then uses new value of i**
  - **Postfix Decrement (i--): uses current value of i first & then subtracts 1**

Relational Operators

- **<, <=, >, >=**
  - are binary operators.
  - binary numeric promotion is applied.
  - have left associativity.
  - return boolean values

Equality

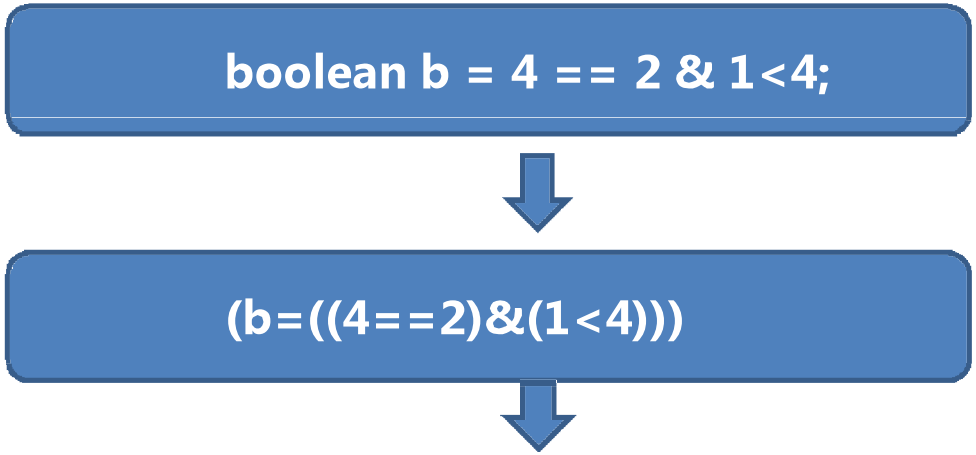
- Primitive Data Value Equality: ==, !=
  - to compare primitive data values, including boolean values.
  - have left associativity: (a==b==c) => ((a==b)==c)
  - results in boolean values (true / false).

```
int a, b, c;
boolean valid1 = a == b == c;
boolean valid2 = a==b && b==c;
boolean valid3 = a == b == true;
```

Compile time error

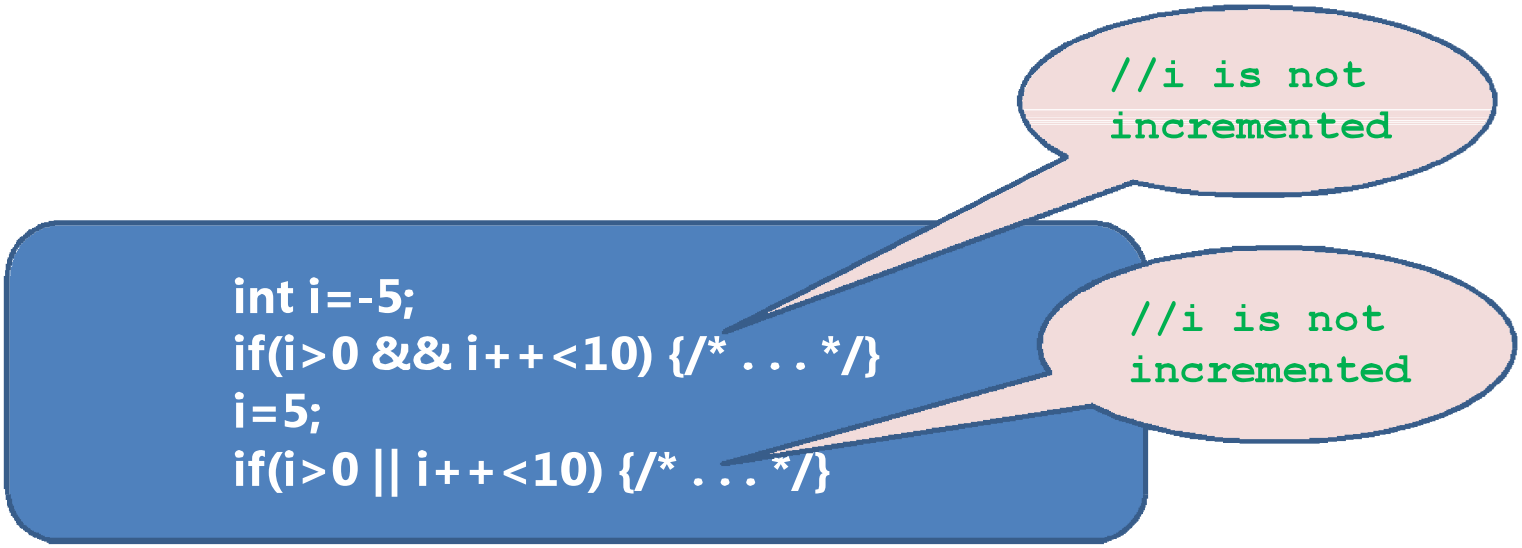
Boolean Logical Operators

- !, ^, &, |
  - can be applied to boolean operands.
  - results in boolean values (true / false).
  - There are compound assignment operators (&=, ^=, |=)



Short Circuited Operators

- **&&, ||**
  - Same as &,| except evaluation is ***short-circuited***.
  - only be applied to boolean operands (not to integral operands).
  - No compound assignment operators (&&=, ||=).



Bitwise Operators

- **~, &, |, ^**
  - perform bitwise operations between corresponding individual bit values in the operands.
  - Unary numeric promotion is applied to ~.
  - Binary numeric promotion is applied to &, |, ^.
  - Compound assignment operators exist : &=, ^=, |=.

Shift Operators: left shift

- **Shift left: <<**
  - a<<n : Shift all bits in a left n times; filling with 0 from right.
  - corresponds to multiplication of value by 2.

```
byte b = 32;
int j = b<<3;          //256
b = (byte)(a<<3)       //0
```

Shift Operators: right shift

- **Shift Right: >>**
  - a>>n : Shift all bits in a right n times; filling with the sign bit from the left.
  - 0 is filled for +ve operand; 1 is filled for –ve operand.
  - Each right shift corresponds to division by 2.
- **Shift Right with zero fill: >>>**
  - a>>n : Shift all bits in a right n times; filling with 0 from the left.
  - 0 is filled in any case.

Conditional Operator

- : ?

  <condition> ? <expression1>:<expression2>

  – if condition true , expression1 is evaluated otherwise expression2.

  – Equivalent to if-then-else

  – Conditional expressions can be nested.

  (a?b?c?d:e:f:g) ➔ (a?(b?(c?d:e):f):g)

new, [] , instanceof

Summary : Session

With this we have come to an end of our session, where we discussed about ….

- Operators in java
- Associativity of operators
- Conversion of data

At the end of this session, we see that you are now able to

- Use operators
- Perform data conversion

Reference Material : Websites & Blogs

- http://www.tutorialspoint.com/java/java_basic_operators.htm
- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op1.html
- http://www.javatpoint.com/operators-in-java

Reference Material : Books

- ***Head First Java***
  - *By: Kathy Sierra, Bert Bates*
  - *Publisher: O'Reilly Media, Inc.*

- ***Java Complete Reference***
  - *By Herbert Schildt*

Key Contacts :

Java Interactive :

- Asif Immanad

asif_immanad@persistent.co.in

- Nisha Waikar

nisha_waikar@persistent.co.in

- Varsha Mulik

varsha_mulik@persistent.co.in

Persistent University :

- Poorva Kulkarni

poorva_kulkarni@persistent.co.in

Vice President :

- Shubhangi Kelkar

shubhangi_kelkar@persistent.co.in

Thank You !!!

Persistent University