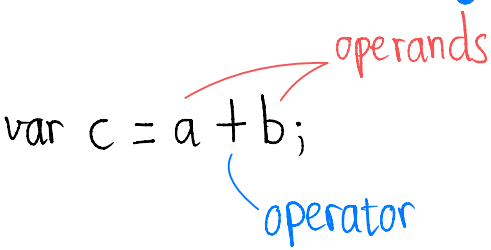
Chapter 2

Operators

Understanding Java Operators



Types of Operators

3 flavors of operators:

- unary (have the highest order of precedence)
- binary
- ternary

Operator Precedence

Operator	Symbols and examples	Evaluation
Post-unary operators	*expression*++, *expression*--	Left-to-right
Pre-unary operators	++*expression*, --*expression*	Left-to-right
Other unary operators	-, !, ~, +, (*type*)	Right-to-left
Cast	(Type)reference	Right-to-left
Multiplication/division/modulus	*, /, %	Left-to-right
Addition/subtraction	+, -	Left-to-right
Shift operators	<<, >>, >>>	Left-to-right
Relational operators	<, >, <=, >=, instanceof	Left-to-right
Equal to/not equal to	==, !=	Left-to-right
Logical AND	&	Left-to-right
Logical exclusive OR	^	Left-to-right
Logical inclusive OR	|	Left-to-right
Conditional AND	&&	Left-to-right
Conditional OR	||	Left-to-right
Ternary operators	*boolean expression* ? *expression1* : *expression2*	Right-to-left
Assignment operators	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=	Right-to-left
Arrow operator	->	Right-to-left

Applying Unary Operators

Operator	Examples	Description
Logical complement	`!a`	Inverts a boolean's logical value
Bitwise complement	`~b`	Inverts all 0s and 1s in a number (byte, short, char, int, long)
Plus	`+c`	Indicates a number is positive, although numbers are assumed to be positive in Java unless accompanied by a negative unary operator
Negation or minus	`-d`	Indicates a literal number is negative or negates an expression
Increment	`++e` `f++`	Increments a value by 1
Decrement	`--f` `h--`	Decrements a value by 1
Cast	`(String)i`	Casts a value to a specific type

~(value) = -1 * value - 1

```
int pelican = !5; // Error
boolean penguin = -true; // Error
boolean peacock = !0; // Error
```

Operator	Example	Description
Pre-increment	++w	Increases the value by 1 and returns the *new* value
Pre-decrement	--x	Decreases the value by 1 and returns the *new* value
Post-increment	y++	Increases the value by 1 and returns the *original* value
Post-decrement	z--	Decreases the value by 1 and returns the *original* value


```
int num = 0;
System.out.println(num); // 0
System.out.println(++num); // 1
System.out.println(num); // 1
System.out.println(num--); // 1
System.out.println(num); // 0
```

outputs the value **BEFORE** the decrement occurs

Working with Binary Arithmetic Operators (+, -, *, /, %)

```
int price = 2*5+3*4-8; //14
int price = 2*((5+3)*4-8); //48
long pigeon = 1+(3*5)/3; //Error
short robin = 3+[(4*2)+4]; //Error
```

% operation may also be applied to:

- Negative integers
- Floating-point numbers

Numeric Promotion Rules

1/ data type A | data type B (A < B)

A promoted → B

2/ A : integral | B : floating-point

A promoted → B

3/ A: byte, short, char | binary arithmetic operator | variable

A first promoted → int

4/ After all promotion has occurred & the operands have the same data type, the resulting value will have the same data type as its promoted operands

```
int x = 1;
long y = 33;
var z = x * y; // long
```

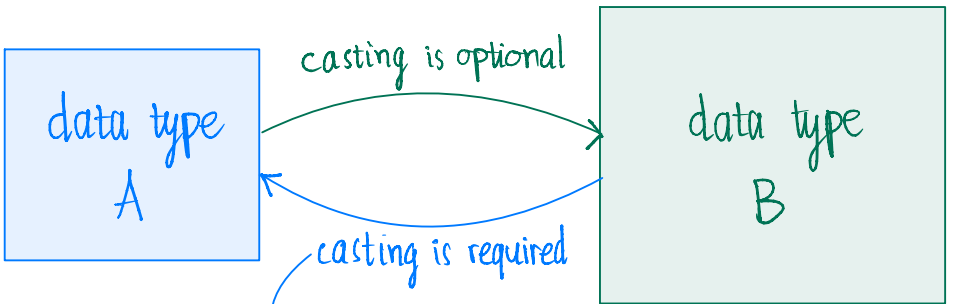
```
double x = 39.21;
float y = 2.1; // Error, 2.1 is double
var z = x + y;
```

```
short x = 10;
short y = 3;
var z = x * y; // 3rd rule, int
```

```
short w = 14;
float x = 13;
double y = 30;
var z = w * x / y; // double
```

Assigning Values

Casting Values



also required when casting floating-point → integral

```
int hair = (short) 2;
String type = (String) "Birds";
short tail = (short) (4 + 10);
float egg = 2.019; // Error
int tadpole = (int) 5 * 2L; // Error
short frog = 3 - 2.0; // Error
```

Casting Value vs. Variables

```
byte hat = 1;
byte gloves = 7 * 10; // constant folding
short scarf = 5;
short boots = 2 + 1;
```

```
short boots = 2 + hat; // Error, 3rd rule
byte gloves = 7 * 100; // Error, overflow
```

Compound Assignment Operators (+=, -=, *=, /=)

CANNOT be used to declare a new variable

```
Long goat = 10;
int sheep = 5;
sheep = sheep * goat; // Error
sheep *= goat; // OK
```

(automatically)
cast sheep to long
→ multiplication
→ cast result to int

```
long wolf = 5;
long coyote = (wolf = 3);
System.out.println(wolf); //3
System.out.println(coyote); //3
```

Comparing Values

Operator	Example	Apply to primitives	Apply to objects
Equality	`a == 10`	Returns `true` if the two values represent the same value	Returns `true` if the two values reference the same object
Inequality	`b != 3.14`	Returns `true` if the two values represent different values	Returns `true` if the two values do not reference the same object


```
boolean monkey = true == 3;
boolean ape = false != "Grape";
boolean gorilla = 10.2 == "Koko";
```

CANNOT MIX types

```
var monday = new File("schedule.txt");
var tuesday = new File("schedule.txt");
var wednesday = tuesday;
System.out.println(monday == tuesday); //false
System.out.println(tuesday == wednesday); //true
System.out.println(null == null); //true
```

instanceof operator

```
public void openZoo(Number time) {
  if (time instanceof Integer)
    System.out.print((Integer)time + " O'clock");
  else
    System.out.print(time);
}
```

```
public void openZoo(Number time) {
  if(time instanceof String) //Error
    System.out.print(time);
}
```

```
System.out.print(null instanceof Object); //false
Object noObjectHere = null;
System.out.print(noObjectHere instanceof String); //false
```

Calling instanceof on the null literal or a null reference always returns false

```
System.out.print(null instanceof null); //Error
```

Logical Operators

Operator	Example	Description
Logical AND	a & b	Value is true only if both values are true.
Logical inclusive OR	c | d	Value is true if at least one of the values is true.
Logical exclusive OR	e ^ f	Value is true only if one value is true and the other is false.

Apply to:

- boolean → Logical operators
- numeric → bitwise operators

Conditional Operators

Operator	Example	Description
Conditional AND	a && b	Value is true only if both values are true. If the left side is false, then the right side will not be evaluated.
Conditional OR	c || d	Value is true if at least one of the values is true. If the left side is true, then the right side will not be evaluated.


```
int hour = 10;
boolean zooOpen = true || (hour < 4);
System.out.println(zooOpen); //true
```

```
if (duck != null & duck.getAge()<5) {
  // Do something
}
```

Logical AND (&) evaluates both sides
→ Could throw a NullPointerException

```
if (duck != null && duck.getAge()<5) {
  // Do something
}
```

If duck is null, the evaluation of duck.getAge()<5 is never reached
→ Prevents a NullPointerException

Unperformed Side Effects

```
int rabbit = 6;
boolean bunny = (rabbit >= 6) || (++rabbit <= 7);
System.out.println(rabbit); // 6
```

Making Decisions with the Ternary Operator

```
int owl = 5;
int food = owl < 2 ? 3 : 4;
System.out.println(food); // 4
```

```
int sheep = 1;
int zzz = 1;
int sleep = zzz < 10 ? sheep++ : zzz++;
System.out.println(sheep + "," + zzz); // 2,1
```

Review Questions

1/ A D G ✓

2/ A B D ✓

3/ B C D F ✓

4/ B ✓

5/ ~~B~~ ~~G~~ **A C**

6/ ~~B~~ **F** (cast float → long)

7/ D ✓

8/ ~~C~~ **A** (pig=pig++; → no increment)

9/ A **D E** (choose all outputs)

10/ G ✓

11/ D ✓

12/ D ✓

13/ F ✓ (precedence of &<=>^)

14/ B E G ✓ (flip = negate)

15/ D ✓

16/ B ✓

17/ C F ✓

18/ C ✓

19/ B F ✓ (overflow → negative)

20/ D E **A**

21/ E ✓

76,19%