Arithmetic, Assignment, and Type Compatibility

Introduction to arithmetic, assignment, and type conversion rules for Java primitive data types

James Brucker

Arithmetic Operators

Arithmetic operators:

-b	Negation
a * b	Multiplication
a / b	Division.
a % b	Remainder of a / b, may be negative
a + b	Addition
a - b	Subtraction
a + b * c	Multiplication then addition

Example: 12 % 5 is 2, 13 % 5 is 3, -12 % 5 is -2,
2 % 5 is 2, 0 % 5 is 0, 20 % 5 is 0.

Arithmetic Using Integers

- These operations apply to integer data, including "`int`" and "`long`" types.

Expression	Result
`int a = 7;`	
`int b = 10;`	
`a + b`	17
`a - b`	-3
`a * b`	70
`a / b`	0
`b / a`	1
`a % b`	7
`b % a`	3
`a / b * b`	0
`b / a * a`	7

Expression	Result
`int c =-12;`	
`int d =  7;`	
`c + d`	-5
`d - c`	19
`c * d`	-84
`c / d`	-1
`d / c`	0
`c % d`	-5
`d % c`	7
`a + b - c / d`	18
`a - b * c + d`	??

Order of Arithmetic Operations

(a *op* b)	expression in parenthesis is performed first,
-*a*	negation is done next,
*** / %**	are done next, left-to-right,
+ -	are done next, left-to-right.

```
a = 12; b = 6; c = 3;
x = a + b / 2 * c;
y = a + b / (2 * c);
z = (a + b) / (2 * c);
```

```
a = 3
x = 12 + (6/2)*3 = 21
y = 12 + 6 / (2*3) = 13
z = (12 + 6)/(2*3) = 3
```

```
a = 3;
x = 4 + 2 * 9 / 6 / a - 1;
y = 2 + 12 * 2 / 6 % a - 1;
```

```
a = 3
x = 4 + (18/6)/3 - 1 = 4
y = 2 + (24/6)%3 - 1 = 3
```

Quiz on Order of Operations

What are the resulting values for the following?

```
a = 4; b = 12; c = 4; d = 2;
n1 = a + b * c + d;
n2 = a + b * (c + d);
n3 = b / a * c / d;
n4 = b / a + c / d;
n5 = -a + 15 % c - d;
```

n1 =

n2 =

n3 =

n4 =

n5 =

Type of Results

What is data type of the result of an operation?

Examples: what is...

15 * 200 = 3000 (int)

15F * 200F = 3000F (float)

0.5 * 7.0 = 3.5 (double)

7 / 2 = 3 (int) ... *NOT* 3.5

Rule: For primitive numeric types, the result of + - * / % is the same type!

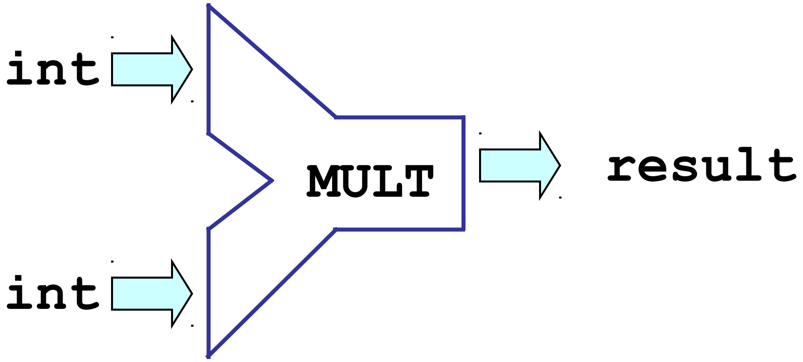
a	b	a *op* b	- a
int	int	int	int
long	long	long	long
float	float	float	float
double	double	double	double

How is Arithmetic Done?

QUESTION:

- Does the CPU have hardware instructions for + - * / involving integer data, or does it use software?
- Does the CPU have hardware instructions for + - * / involving float point data?
- What is the name of the CPU component that performs + - * / ?

is multiplication done using hardware or software?

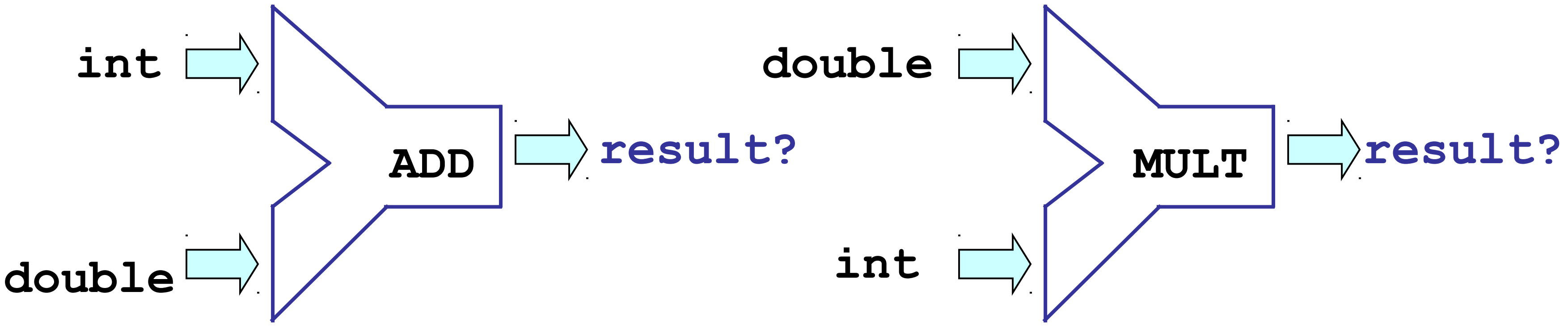


Arithmetic and Type Conversion

The **BIG** Question:

The CPU can't *directly* add int + double or int * double, because they are different types.

- So, *what does Java do when we write 2 + 0.5 or 1.1 * 8 ?*
- What is the *data type* of the result?



Arithmetic and Type Conversion (1)

- Operations are defined for *each* data type
- When Java performs arithmetic (+ - * / %) on two values, both values *must be the same data type*.

a ***op*** b.

a and b must be same data type

Example	Data Types	Result
4 + 1000L	int + long	?
5 * 0.1F	int * float	?
2.5 * 0.8F	double * float	?
'4' + 100	char + int	?

operation on mixed types is not defined.

Type Promotion

- If a and b are different types, Java will try to *promote* one of the values to make them the same type

Example	Data Types	Promotion	Result
4 + 1000L	int + long	promote 4 to long	4L+1000L
5 * 0.1F	int * float	promote 5 to float	5.0F * 0.1F
2.5 * 0.8F	double * float	promote 0.8F to double	2.5 * 0.8
'4' + 100	char + int	promote char to int	52 + 100

Don't do this! The int value of '4' (char) is 52.

Automatic Type Promotion

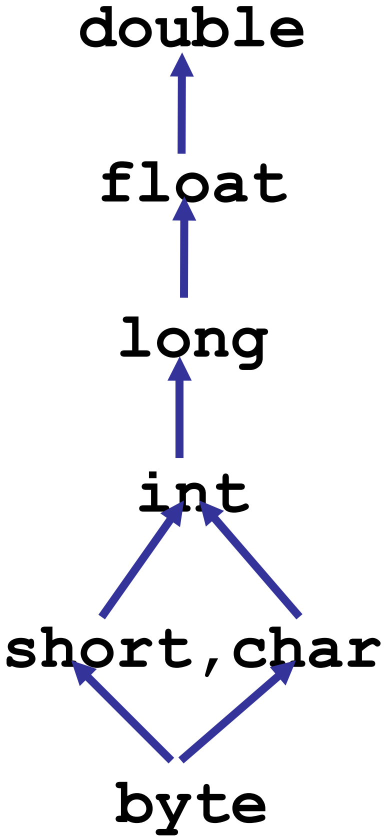
1. to perform arithmetic, Java **always** *promotes* `byte` and `short` values to "`int`".

   ```
   short a = 100;
   byte b = 50;
   a + b result is (int) 150
   a * a result is (int)
   ```

2. In other cases, Java performs a "widening" conversion. (see next slide)

Why use int?
The ALU in most
CPUs is designed
for 32-bit
or 64-bit data.

List of Automatic Promotions



double

float

long

int

short,char

byte

Rules

- The "higher" types can store any value that was stored in the lower types. But...
- There are some *loss of precision* in these cases:
  int -> float
  long -> float
  long -> double
- Conversion byte -> char, char -> int is mostly for I/O involving character data. Be careful!

Widening Conversions

These promotions are called *widening conversions* because the higher data types have larger ("wider") range of possible values.

Automatic Conversions

The widening conversions are easy to remember if you remember the size & range of each data type:

Data Type	**Size in Memory**	**Range of Values**
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 to 2,147,483,647
long	8 bytes	-9,223,372,036,854,775,808L 9,223,372,036,854,775,807L
float	4 bytes	±3,402823E+38
double	8 bytes	±1.79769313462157E+38

More Type Promotion

1. If one argument is integer ("`int`" or "`long`") and the other is "`float`" then integer is promoted to "`float`"

   50 * 2.5f   result is (float) 125.0f

   2.98E-5 * 1000L   result is (double) 0.029800...

2. if either operand is "`double`", then the other operand is converted to "`double`" and the result "`double`"

   double x = 0.25;

   8 * x result is (double) 2.0

   x * 0.5f result is (double) 0.125

   1 / 2 * x result is (double) 0.0   Why?

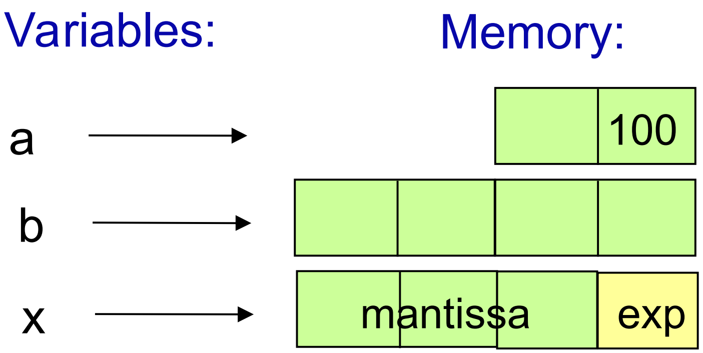
   x * 1 / 2 result is (double) 0.125 Why?

Assignment and Type Compatibility

- When assigning a value to a variable (a = 2*b + c), the left side must be *type compatible* with the right side.
- An assignment that requires a *widening conversion (type promotion)* is considered type compatible.

Example:

```
short a = 100;
int b = 1000;
float x = 2E+30;
b = a;
x = a;
a = b;
b = x;
```



no problem: b can store any "short" value

no problem: x has store any "short" value

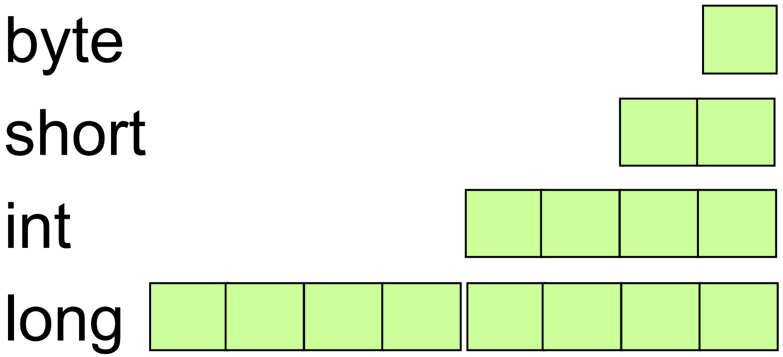
error! a is too small to store all "int" values

error! b cannot store some large "float" values

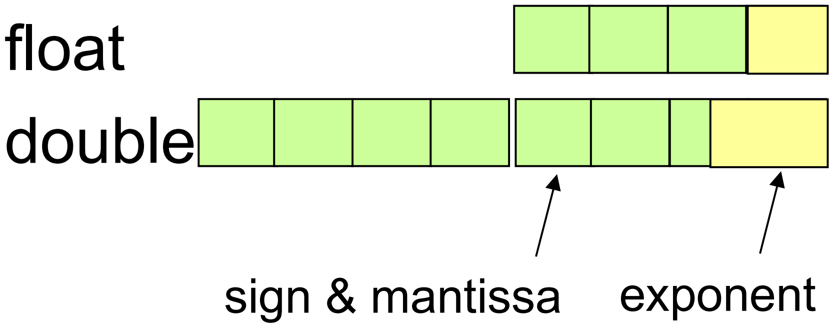
Automatic Conversions (2)

Value	Can be converted and assigned to:
byte	short, int, long, float, double
short	int, long, float, double
int	long, float, double
long	float, double
float	double

Integer Data Types and Memory



Floating Point Data Types



Examples

`int ax = 100;`	
`float fx;`	
`double dx;`	
`fx = 2;`	OK. Convert 2 to 2.0F (**`float`**) then assign to fx.
`fx = 2.0;`	Error: 2.0 is a **`double`**. Can't assign to **`float`** fx.
`ax = fx;`	Error. can't assign a **`float`** in an **`int`** variable.
`dx = ax;`	OK. *Promote* value of ax to **`double`**, then assign.
`dx = 0.5F * ax;`	OK. *Promote* ax to **`float`**, then multiply (**`float`**), then promote result to a **`double`** and assign.
`ax = 0.5 * 100;`	Error. *Promote* 100 to **`double`** (0.5 is **`double`**) then multiply. But can't assign the result (**`double`**) to **`int`** variable ax.

The Type of Numeric Literals

Value	Is Automatically of Type:
0 1 -8000 123456789	int
0L 1L -8000L 123456789L	long
0. 2.5 2.98E+8 -1E-14	double
0F 2.5F 2.98E+8F -1E-14F	float
2.5L	Error: incompatible

The "default double" is one of Java's most annoying "features".

```
float x, y;
x = 100;     // OK.  Integer 100 can be converted to "float"
y = 0.5 * x;  // Error!  "0.5" is a double, so the result is a double
y = 0.5F * x;// OK.  Both operands are float, so result is float
```

Examples

Expression	Result
15 / 2	7 (int)
15 / 2.0F	7.5F (float)
15 / 2.0	7.5 (double!)
int VAT = 7; // tax rate	
5000 * (1 + VAT/100)	5000 (no tax!)
5000 * (1 + VAT/100.)	5350. (tax)
int a; float x; double d;	0
x = 3.14159;	Error: float <-- double
a = 2.5F * x;	Error: int <-- float
d = 12345678901121314L;	OK: double <-- long
a = Math.sqrt(2);	Error: int <-- double

"L" denotes a "long" constant

Common Errors

1. Create a double variable with value 1/2.

```
double x;
x = 1 / 2;
out.println( x );
```

Bug: 1 and 2 are "int", so integer arithmetic is used.

Output is 0

2. Compute 1/3 of the sum

```
int sum = 90;
int part;
part = (1/3) * sum;
out.println( part );
```

Bug: 1 and 3 are "int", so integer arithmetic is used.

Output is 0

How to Fix these Common Errors

1. Create a double variable with value 1/2.

```
double x;
x = 1.0 / 2.0;
out.println( x );
```

Fixed: 1 and 2 are double. Easier: x = 0.5.

Output value is 0.5

2. Compute 1/3 of the sum (sum can be int, float, ...).

```
int sum = 90;
int part;
part = sum / 3;
out.println( part );
```

Fixed: use data type of sum for arithmetic.

Output value is 30

Example: Area of a Circle

Problem:

given the radius of a circle, find its area.

Algorithm for Solution:

1. Read the radius from the input
2. Compute area using $A = \pi * r^2$
3. Display the result.

Project budget:

- ❑ Development: 1 day (including testing!)
- ❑ Training the user: 0.5 day
- ❑ Budget: 15,000 Baht

Example: Area of Circle

```
import java.util.Scanner;
/**
 *  Compute the area of a circle
 */
public class Circle {
  public static void main( String [ ] args) {
     Scanner console =
              new Scanner( System.in );
     System.out.print("Input radius of circle: ");
     double radius = scan.nextDouble( );
     double area = Math.PI * radius * radius;
     System.out.println("The radius is "+radius);
     System.out.println("The area is "+area);
  }
}
```

Java classes are grouped into "packages" to help organize.

This import says "Scanner" is in package java.util.

Name of this class is Circle. The filename must be **`Circle.java`**

Increment/Decrement Operators

Java has increment and decrement operators:

`x++` use the value of x, then add 1

`++x` add 1 to x, *then* use the value

`x--` use the value of x, then subtract 1

`--x` subtract 1 from x, *then* use the value

Examples:

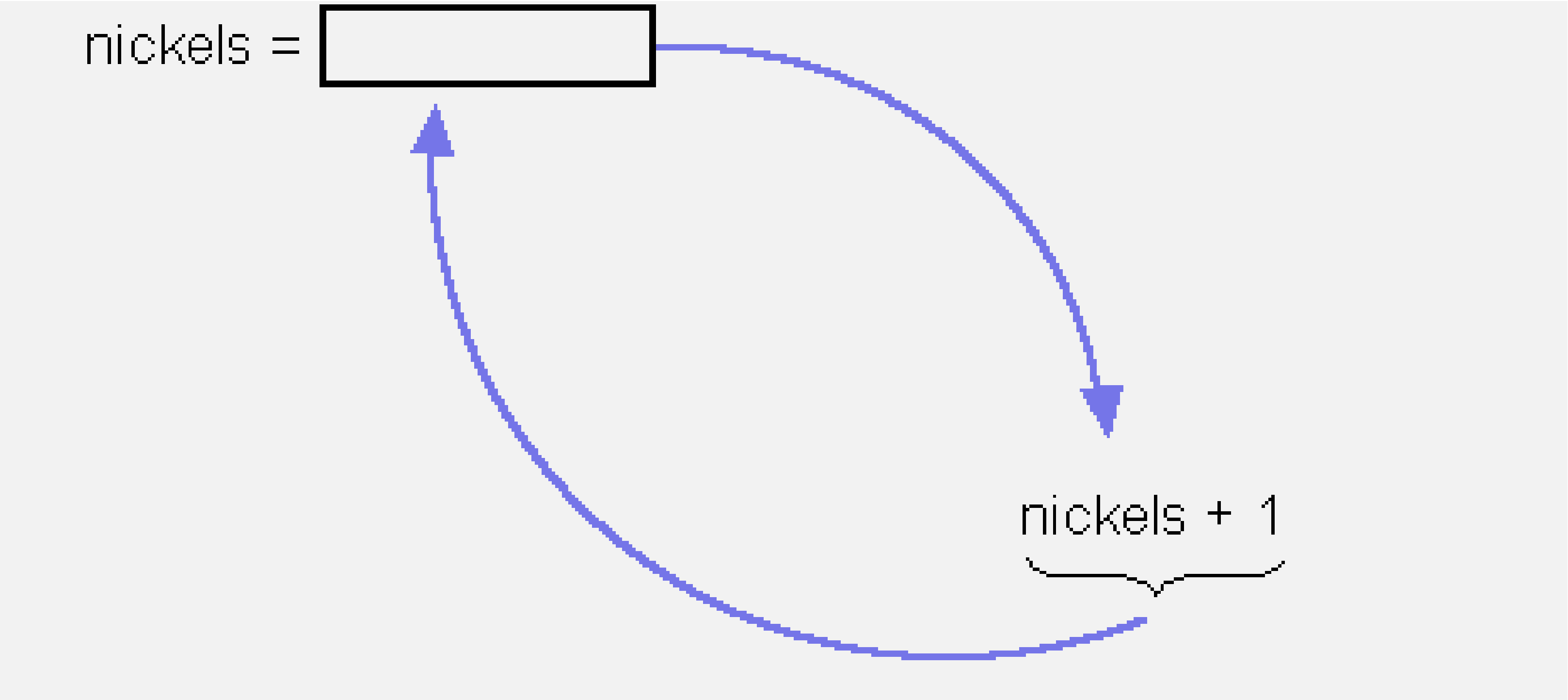
```
int x = 10;
int w, y, z;
w = x++;  // now w = 10 and x = 11
y = 2 * ++x;    // increment x, then use: y = 2 * 12 = 24
x++; // can increment x as a statement by itself!
```

Increment: `nickels++`

`nickels++` means give me another nickel!

(1) return the current value of nickels

(2) then, add one to the value



Increment/Decrement Operators (2)

Often used to increment a loop index or keep a count, like this:

```
int count = 1;
while ( count < 4 ) {
   System.out.println("count = " + count);
   count++;
}
System.out.println("Done. count = "+count);
```

```
count = 1
count = 2
count = 3
Done. count = 4
```

Increment/Decrement Operators (3)

Increment is also used in counting things, like this:

```
// read numbers and compute the average
int count = 0;
long sum =  0;
Scanner scanner = new Scanner( System.in );
while ( scanner.hasNextInt() ) {
   sum = sum + scanner.nextInt( );
   count++;
}
double average = ((double)sum)/count;
System.out.println("The average is "+average);
```

```
Input some numbers: 10 15 20 25
The average is 17.5
```

What are the results?

```
a = 5;

k1 = a++;

k2 = ++a;
```

What are the values of a, k1, k2 ?

```
x = y = 5;

n1 = x++ * y--;

n2 = ++x * y--;

n3 = x++ * --y;

n4 = ++x * --y;
```

What are the values of n1, n2, n3, n4 ?

Compound Assignment Operators

Combine an operation and assignment.

Expression	Meaning
`sum += x;`	`sum = sum + x;`
`sum -= x;`	`sum = sum - x;`
`prod *= x;`	`prod = prod * x;`
`prod /= x;`	`prod = prod / x;`
`prod %= x;`	`prod = prod % x;`

Assignment operators were introduced in the C language,
to help the compiler create more efficient machine code.
Efficiency is also the reason for the n++ and n-- syntax.

Compound Assignment Example

The previous summation example could be rewritten as:

```
// read numbers and compute the average
int count = 0;
long sum =  0;
Scanner scanner = new Scanner( System.in );
while ( scanner.hasNextInt() ) {
   sum += scanner.nextInt( );
   count++;
}
double avarage = ((double)sum)/count;
System.out.println("The average is "+ average);
```

```
Input some numbers: 20 30 10 80
The average is 35.0
```

Operator Precedence (order)

Operations are performed in this order (top to bottom):

Operator	Associativity
[ ], (...), *method*(...)	left to right
! ~ ++ -- +*a* -*a* (*cast*)	right to left
* / %	left to right
+ -	left to right
< <= > >= instanceof	left to right
== !=	left to right
& (bitwise *and*)	left to right
^ (bitwise *xor*)	left to right
| (bitwise *or*)	left to right
&& (*boolean and*)	left to right
|| (*boolean or*)	left to right
= += -= *= /= %=	right to left

Quiz: Operator Precedence

What are the resulting values for the following?

```
double a = 24, b = 12, c = 4, d = 2;
x1 = a + b / c * d
x2 = a / b / c /  d;
x3 = b / a * c / d;
x4 = b / a + c / d;
x5 = (a++ - -b) / 2*c;
x6 = 2*++b;
```

```
x1 =
x2 =
x3 =
x4 =
x5 =
x6 =
```