Chapter 5

Operators & Arithmetic Expressions

Objectives

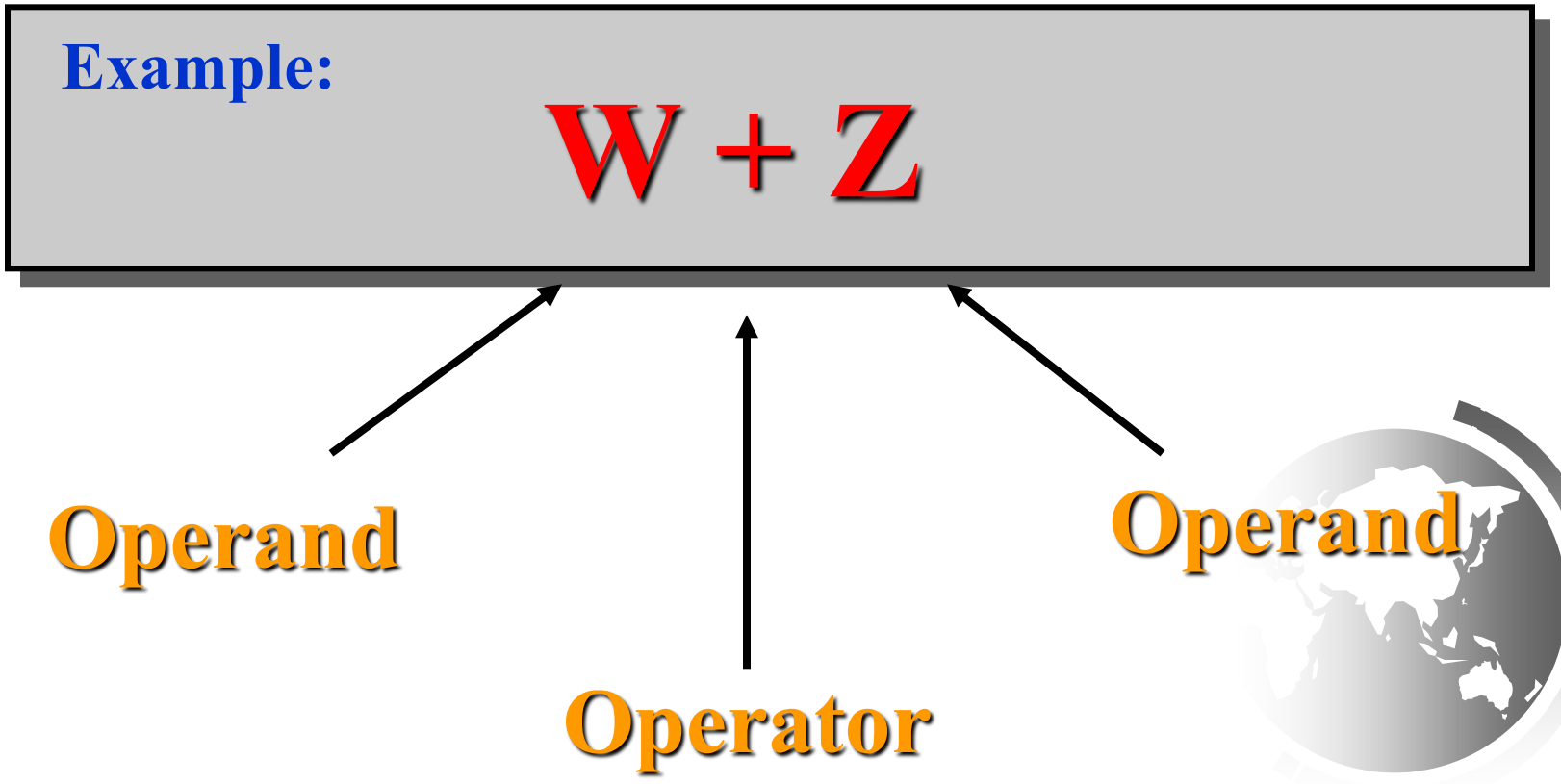
- At the end of this chapter, student should be able to:
  - Know the types of basic arithmetic operators and their order of precedence
  - Develop skills in computer arithmetic

Introduction

- We use arithmetic expressions to solve most programming problems
- Arithmetic expressions comprise of operators and operands
- Unary operator: An operator that has one operand.
- Binary operator: An operator that has two operands.
- There are rules for writing and evaluating arithmetic expressions

Operator and Operand

- What are operator and operand?



Operator and Operand

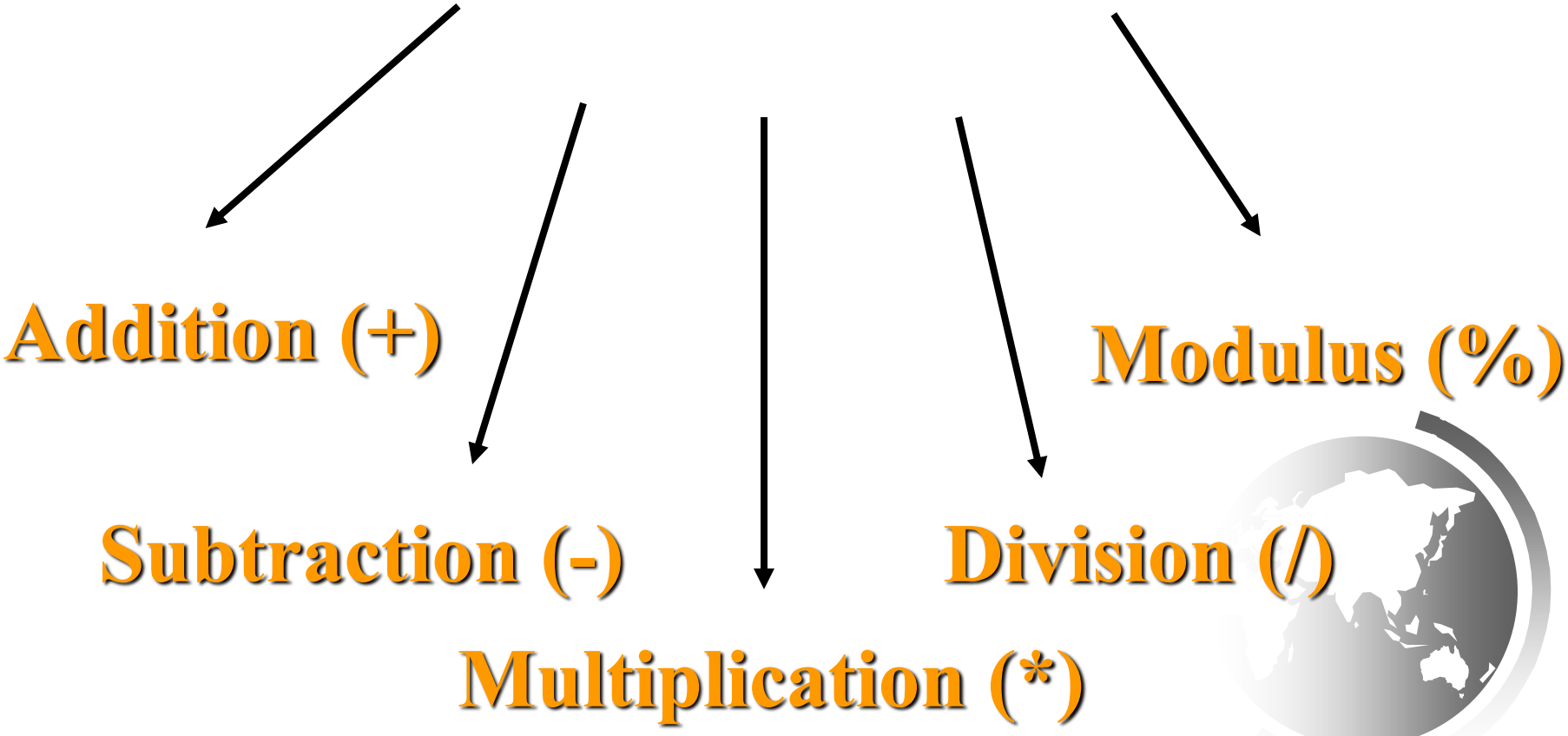
- Which is which?

Operator ??

Example: **A / B**

Operand??

Basic Arithmetic Operators



Basic Arithmetic Operators

- **Multiplication**, **addition** and **subtraction** are the simplest to use
- **Division** is easy, but some precautions need to be taken
- **Modulus** is the one that normally confuses novices

So, let's study in detail the **Division** and **Modulus**

Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Division

Example: **W / Z**

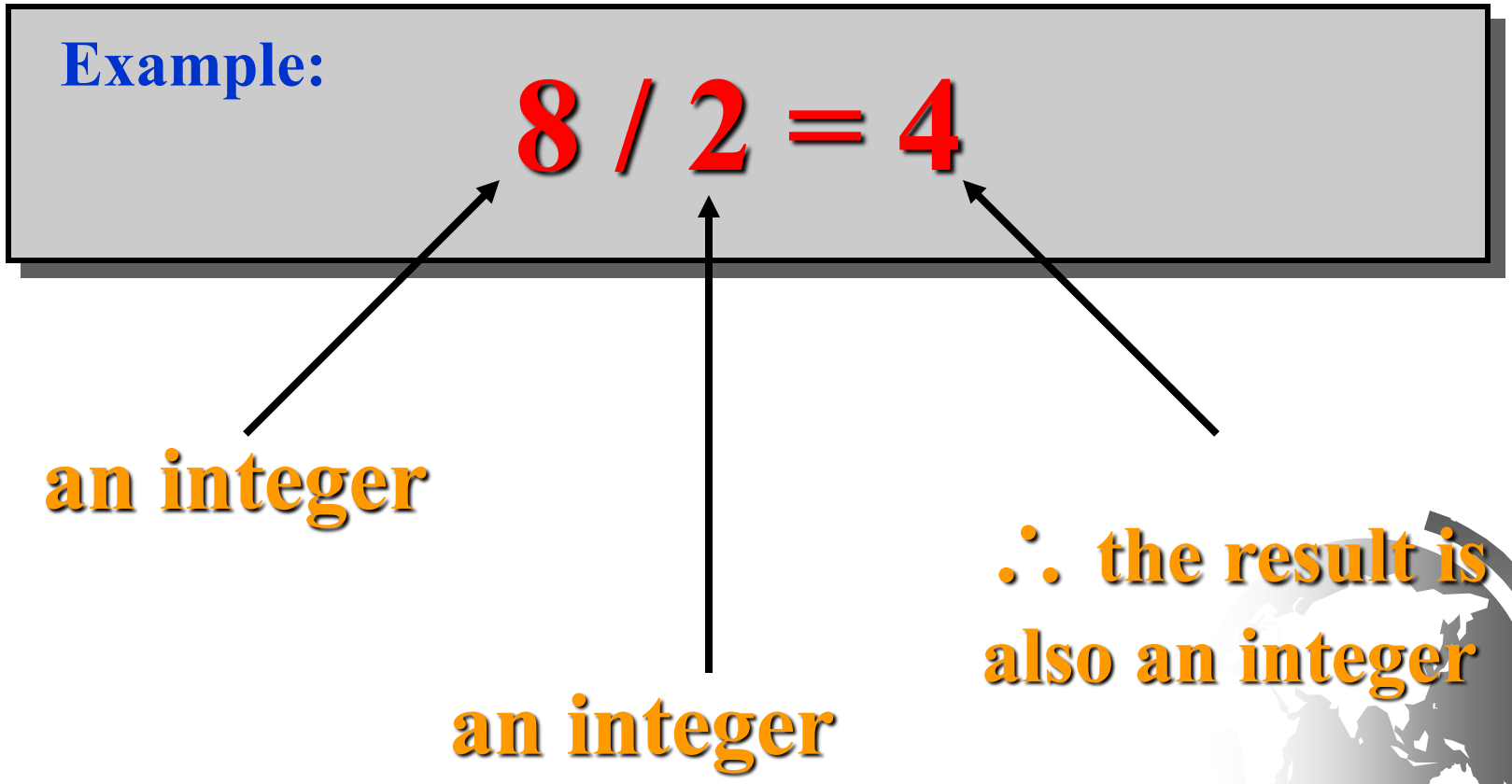
Integer Division

- **W** and **Z** are integers

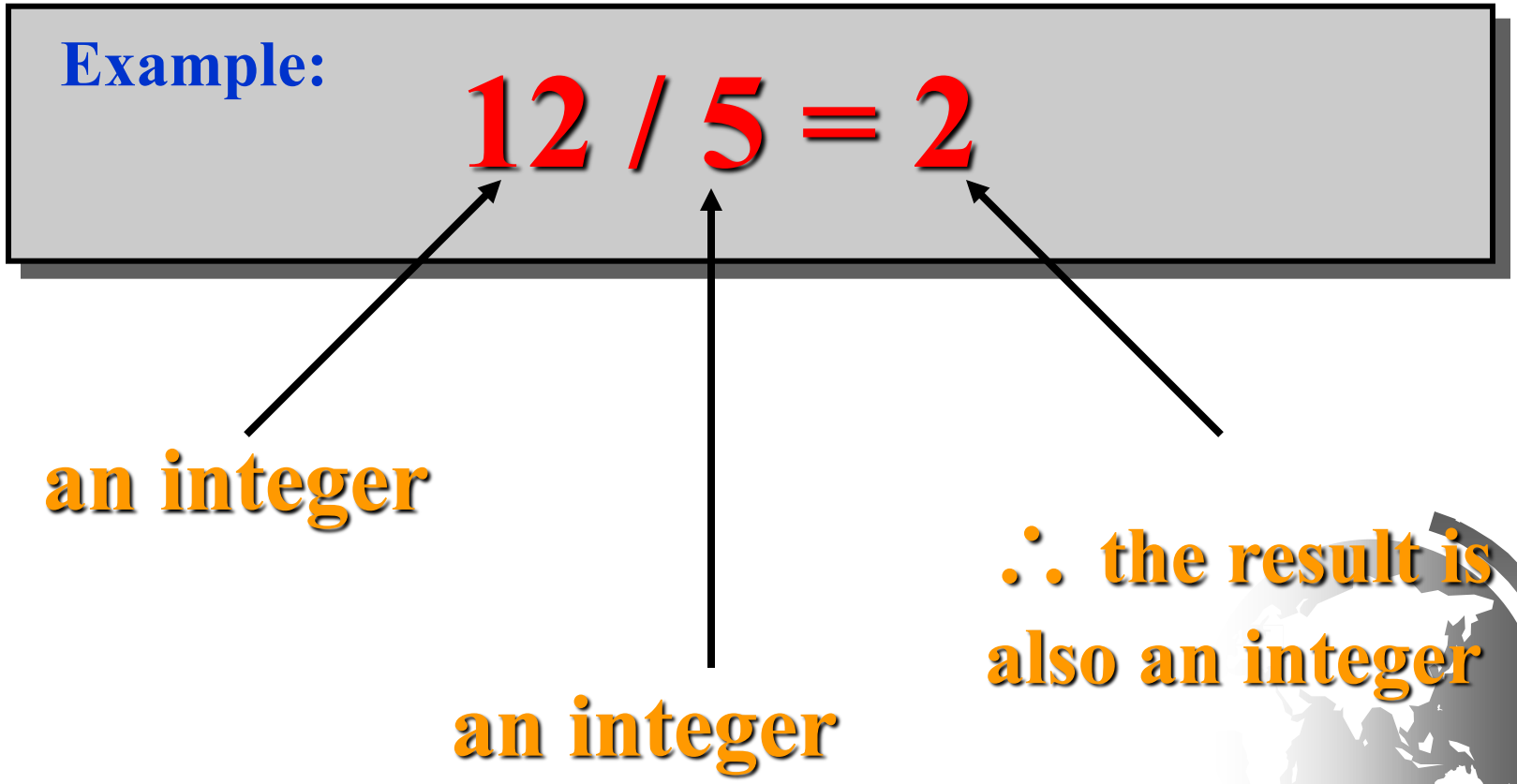
Floating Division

- **W** or **Z** or both are floats

Integer Division



Integer Division



Floating Division

Example:

12.0 / 5 = 2.0

a float

an integer

the result is a float

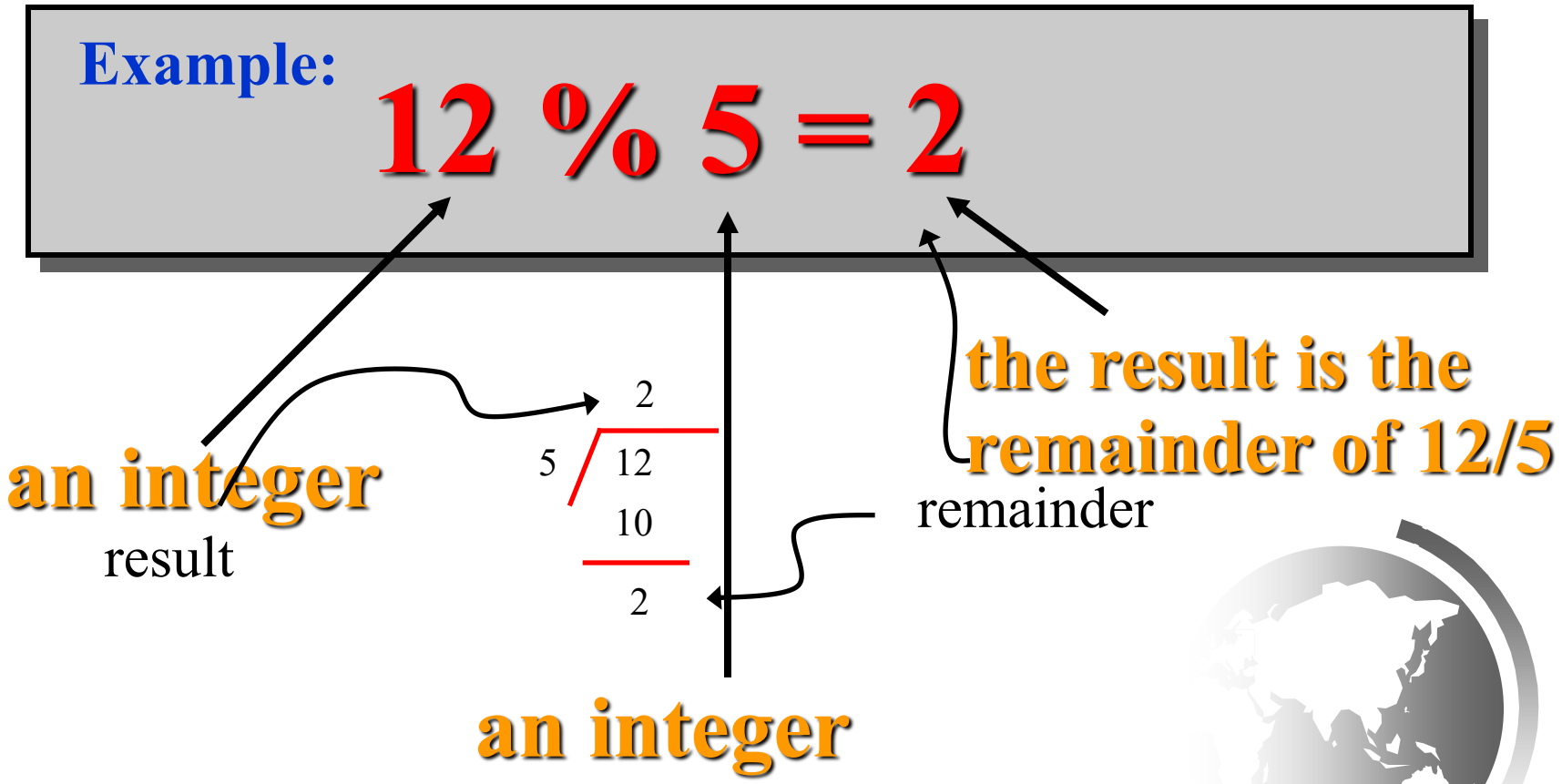
Something to ponder …

What will be the answer if an integer is divided by 0? How about if one of the operands is a negative integer?

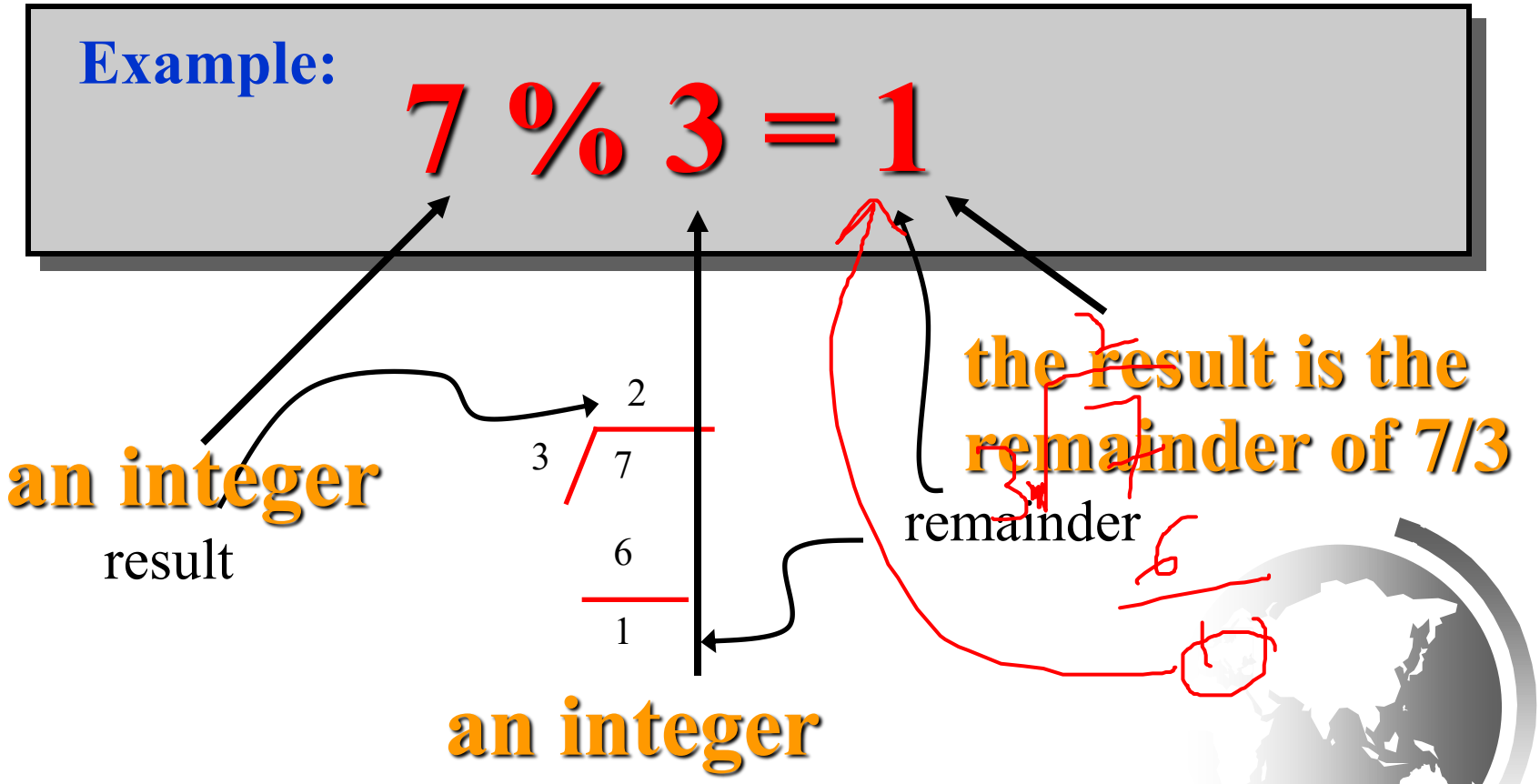
Modulus

- It returns the **remainder** that occurs after performing the division of 2 operands

Modulus

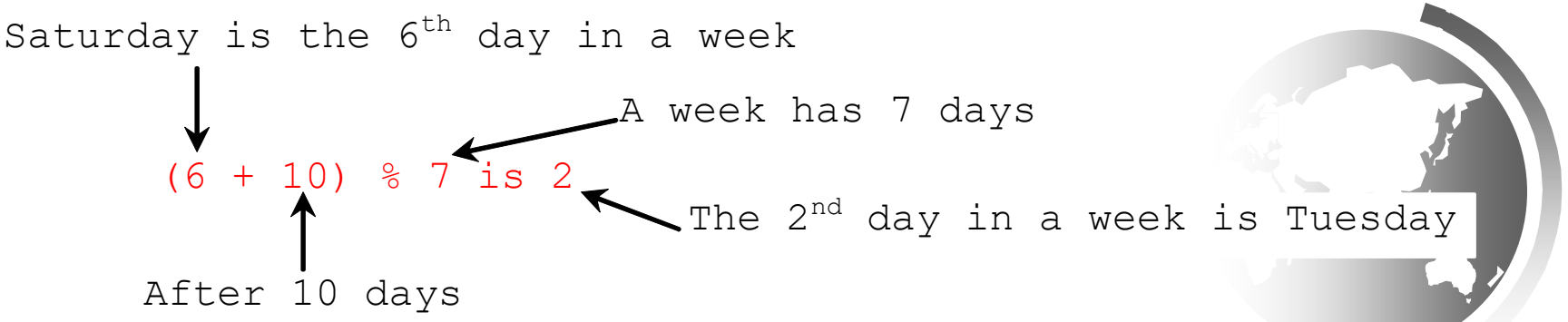

Example:
12 % 5 = 2
an integer
result
2
5 12
10
2
remainder
the result is the remainder of 12/5
an integer

Modulus


Example:
7 % 3 = 1
an integer
result
2
3
7
6
1
remainder
the result is the remainder of 7/3
an integer

Modulus Operator

- Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:



Problem: Displaying Time

- Write a program that obtains minutes and remaining seconds from seconds.

DisplayTime Run

Note!

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy. For example,

```
System.out.println(1.0-0.1-0.1-0.1-0.1-0.1);
```

displays 0.5000000000000001, not 0.5

-
```
System.out.println(1.0 - 0.9);
```

displays 0.09999999999999998, not 0.1.

Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.

Exponent Operations

```
System.out.println(Math.pow(2, 3));
// Displays 8.0
System.out.println(Math.pow(4, 0.5));
// Displays 2.0
System.out.println(Math.pow(2.5, 2));
// Displays 6.25
System.out.println(Math.pow(2.5, -2));
// Displays 0.16
```

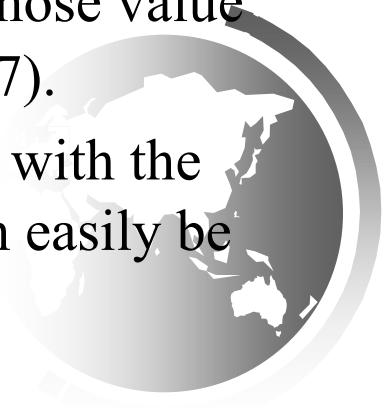
Number Literals

- A literal is a constant value that appears directly in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

```
int i = 34;
long x = 1000000;
double d = 5.0;
```

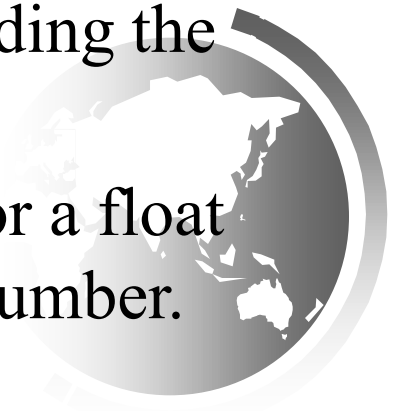
Integer Literals

- An integer literal can be assigned to an integer variable as long as it can fit into the variable.
- A compilation error would occur if the literal were too large for the variable to hold.
- For example, the statement byte b = 1000 would cause a compilation error, because 1000 cannot be stored in a variable of the byte type.
- An integer literal is assumed to be of the `int` type, whose value is between -231 (-2147483648) to 231–1 (2147483647).
- To denote an integer literal of the long type, append it with the letter L or l. L is preferred because l (lowercase L) can easily be confused with 1 (the digit one).



Floating-Point Literals

- Floating-point literals are written with a decimal point.
- By default, a floating-point literal is treated as a double type value.
- For example, 5.0 is considered a double value, not a float value.
- You can make a number a float by appending the letter f or F, and make a number a double by appending the letter d or D.
- For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.



double vs. float

- The double type values are more accurate than the float type values. For example,

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`

7 digits

Scientific Notation

- Floating-point literals can also be specified in scientific notation, for example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456.
- E (or e) represents an exponent and it can be either in lowercase or uppercase.

Something to ponder …

The earlier expressions contain only one operator at a time. What if the expression contains more than one operator?

Arithmetic Expression

- An expression may contain 2 or more arithmetic operators
- Main issue:

ORDER OF PRECEDENCE

Arithmetic Expression

Examples:

5 + 6 = 11

5 + 6 * 2 = 17 or 17?

2.5 + 6 – 2 * 2 = ??

12 / 6.0 – 2 * 2 = ??

Arithmetic Expression

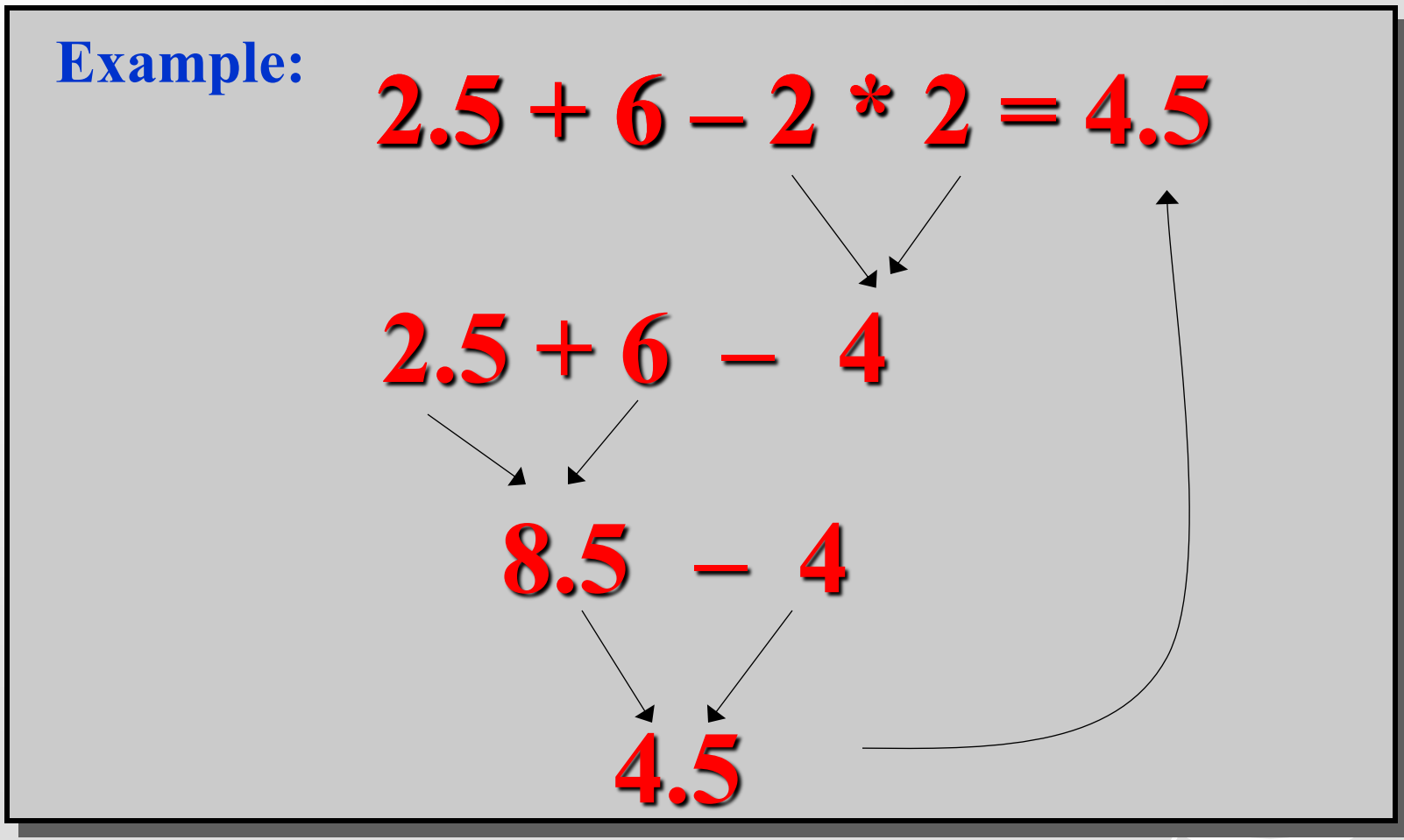
Order of Precedence:

High:	*** / %**
Low:	**+ -**

- All operators have a precedence level.
- High precedence level operators are evaluated before lower ones.
- Operators of the same precedence level are evaluated from left to right

Arithmetic Expression

Example:



Try it!

Example:

12 + 6.0 – 2 * 2 = ??

What's the answer??

Arithmetic Expression

- All expressions in parentheses (brackets) must be evaluated prior to values outside brackets
- Nested parenthesized expressions must be evaluated from the inside out, with the innermost expression evaluated first

Example:

$$(9 - (3 + 2)) * 3 = ??$$

Arithmetic Expression

Example:

$$\therefore (9 - (3 + 2)) * 3 = 12$$

Assignment Statement

- There are 3 types of assignment:
  - Simple
  - Multiple
  - Shorthand

Simple Assignment

Syntax:

variable = expression ;

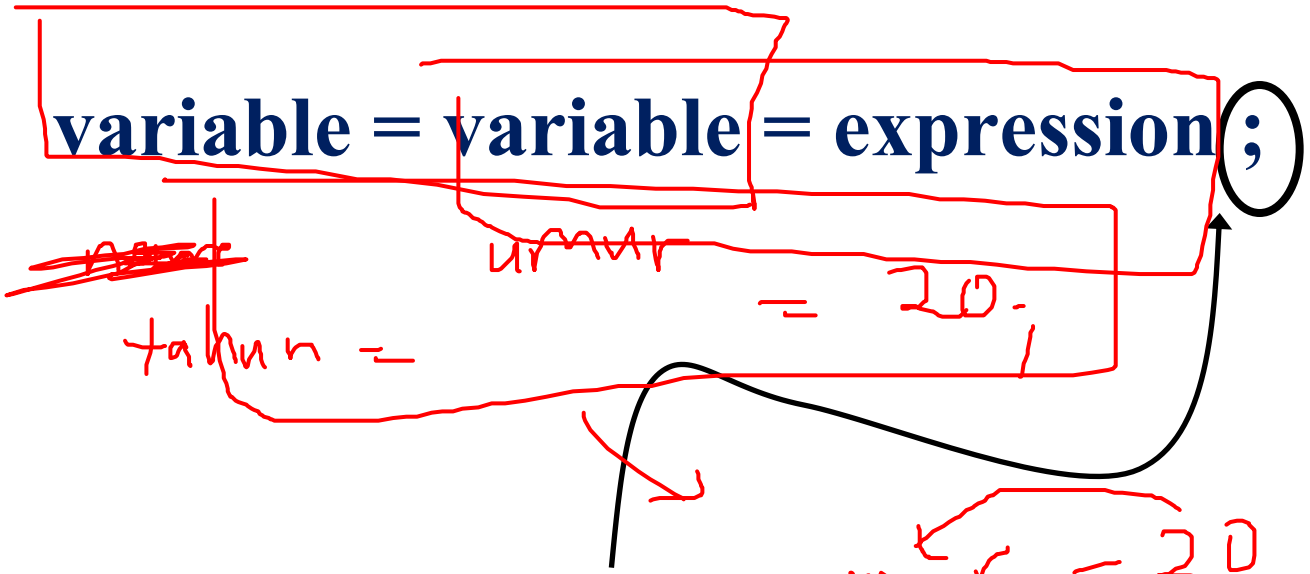
```
Buying price: 10.00
Discount rate: 0.25
For buying price RM10.00 and discount rate 0.25
The total price is RM7.50
```

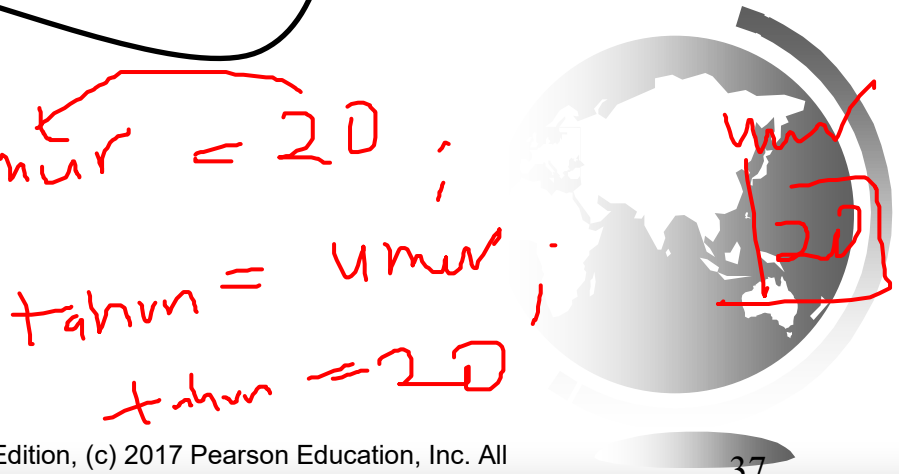
```
public static void main(String [] args ) {
  Scanner  in = new Scanner(System.in);
  float price, discount, total;
  System.out.print("Buying price : ");
  price = in.nextDouble();
  System.out.print("Discount rate :  ");
  discount=in.nextDouble();
  total = price – (price * discount);
  System.out.println("For buying price "+ price + " and discount
        rate" + discount);
  System.out.println("The total price is "+ total);
 }
}
```

price	10.00
discount	0.25
total	7.50

Multiple Assignment

Syntax:

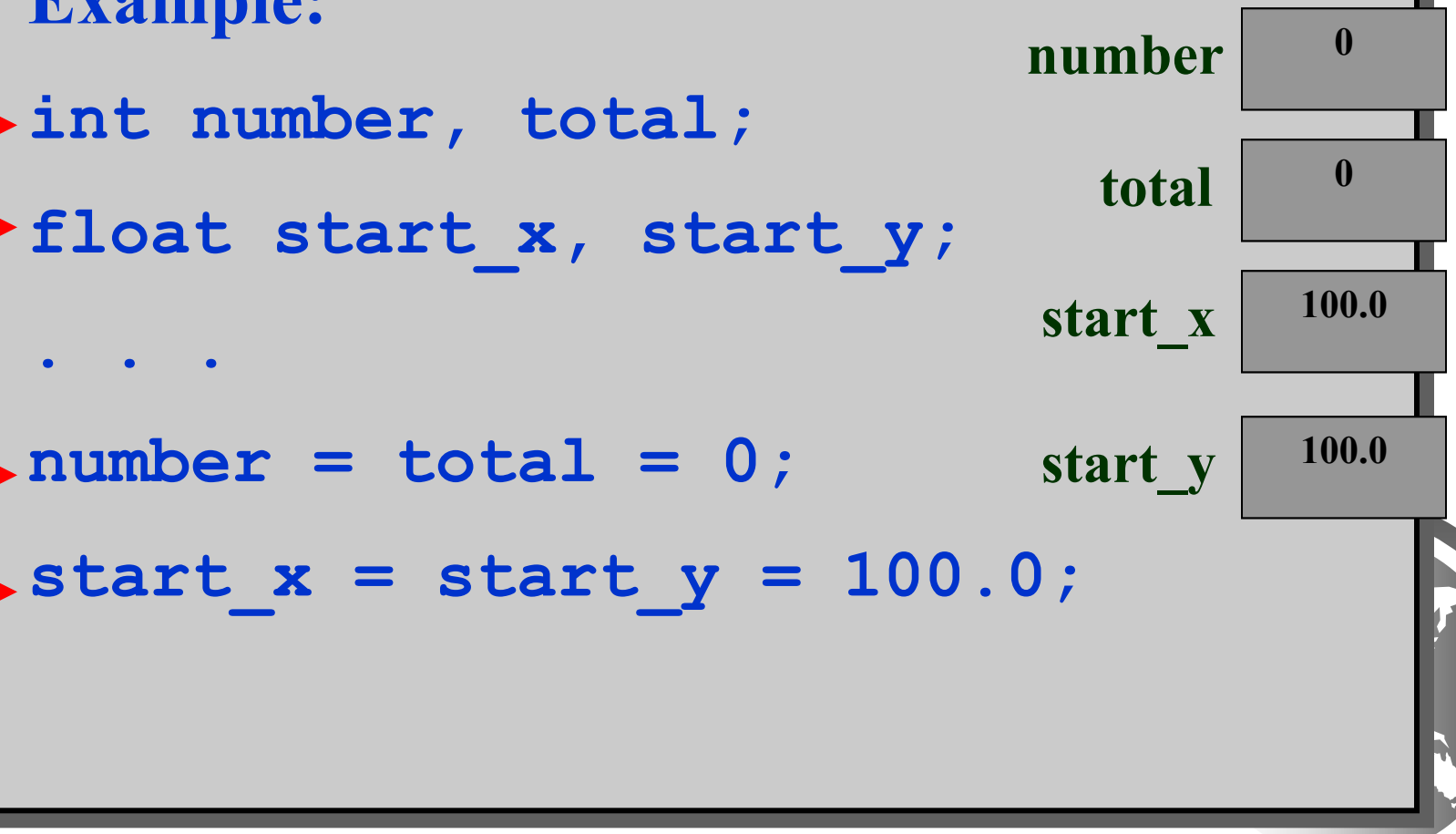




Multiple Assignment

Example:

```
int number, total;
float start_x, start_y;
. . .
number = total = 0;
start_x = start_y = 100.0;
```



Augmented Assignment Operators

Syntax:

variableX = variableX *op* expression ;

variableX *op*= expression;

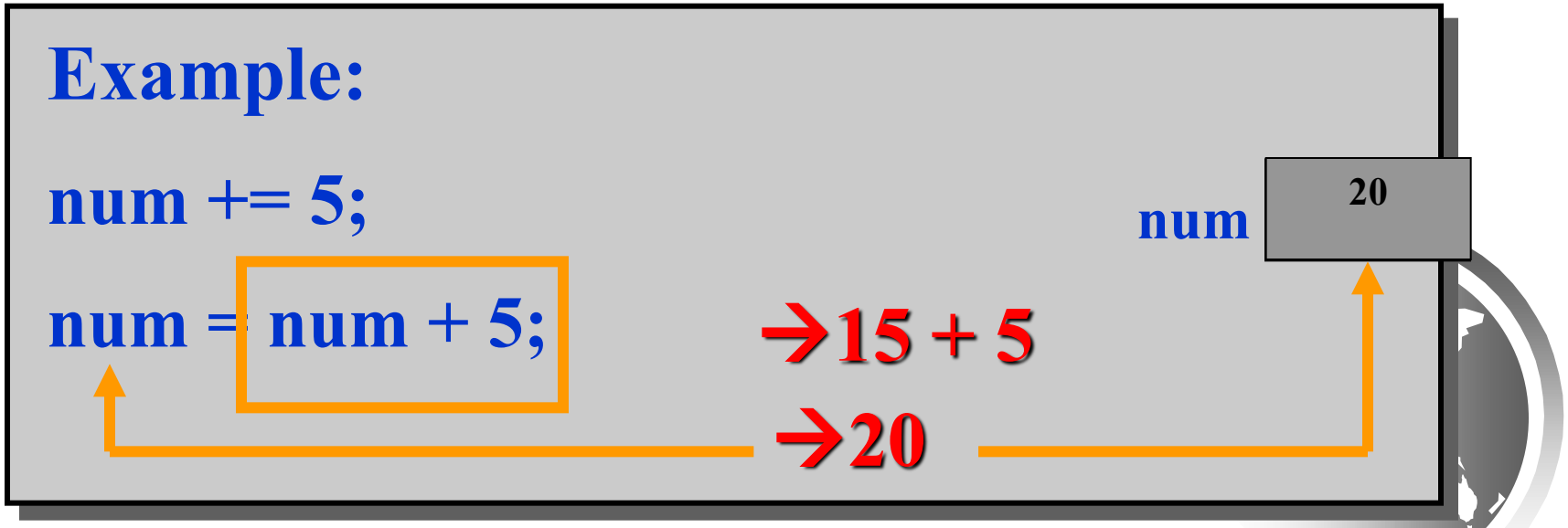
Also called augmented assignment operator

Augmented Assignment Operators

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Division assignment	`i /= 8`	`i = i / 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`

Augmented Assignment Operators

- Whenever the expression on the right contains the variable on the left (to which the value is assigned)



Shorthand Assignment

- Expressions can also be stated using shorthand assignment operators

Example:

num += 5; **similar to** **num = num + 5**

Shorthand assignment operators have the lowest order of precedence – the last one to be evaluated

Shorthand Assignment

Operation	Examples of expression	Description
+=	num += 5;	num = num + 5;
-=	num -= 5;	num = num – 5;
*=	num *= 5;	num = num * 5;
/=	num /= 5;	num = num / 5;
%=	num %= 5;	num = num % 5;

Shorthand Assignment

Example:

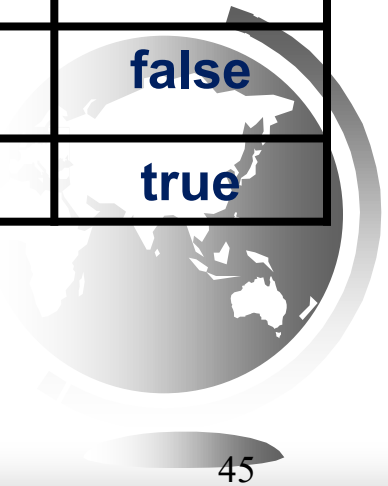
pay += hour * rate * 2

similar to pay = pay + (hour * rate * 2)

→ pay + (hour * rate * 2)
→ pay + (8 * 5.00 * 2)
→ 100.00 + 80.00
→ 180.00

Relational Operators

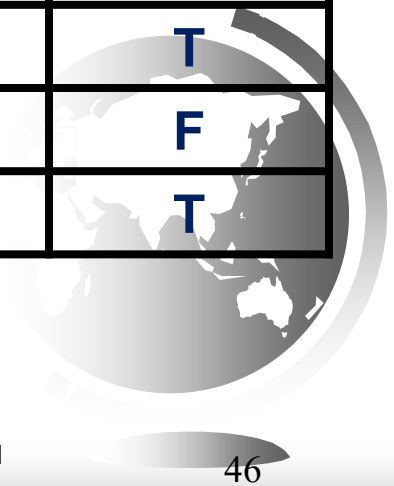
Operation	Description	Examples of Expression	Value
<	Less than	6 < 9	true
<=	Less than or equal to	5 <= 5	true
>	Greater than	2 > 6	false
>=	Greater than or equal to	9 >= 5	true
==	Equal to	7 == 5	false
!=	Not equal to	6 != 5	true



Mantic/Logical Operators

Symbol	Description
&&	AND
||	OR
!	NOT

a	b	a && b	a || b	!a	!b
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

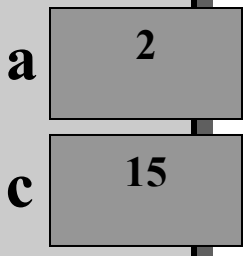


Compound Statement

- Arithmetic, relational and mantic operators can be integrated/combined in one expression

Example:

! (c > a)



→ ! (15 > 2)

→ ! (true)

→ false

Compound Statement

Example:

(a >= 1) && (b == 5)

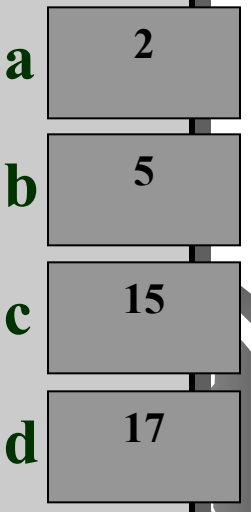
→ (2 >= 1) && (b == 5)

→ true && (b == 5)

→ true && (5 == 5)

→ true && true

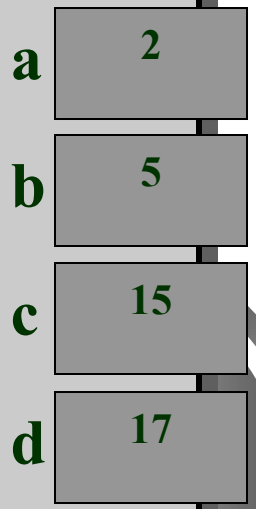
→ true



Compound Statement

Example:

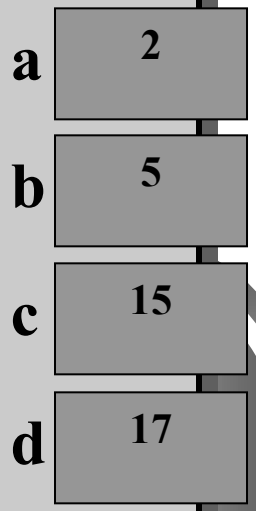
(c >= (b * 3)) || (a == 3)



Compound Statement

Example:

! ((a < b) || (c > d))



Increment and Decrement

- This operation contains only one operand, that is, the operand which value will be incremented/decremented

Symbol	Description	Examples of Expression	Description
++	Increment operand by 1	i++	i = i + 1
--	Decrement operand by 1	i--	i = i – 1

Prefix and Postfix

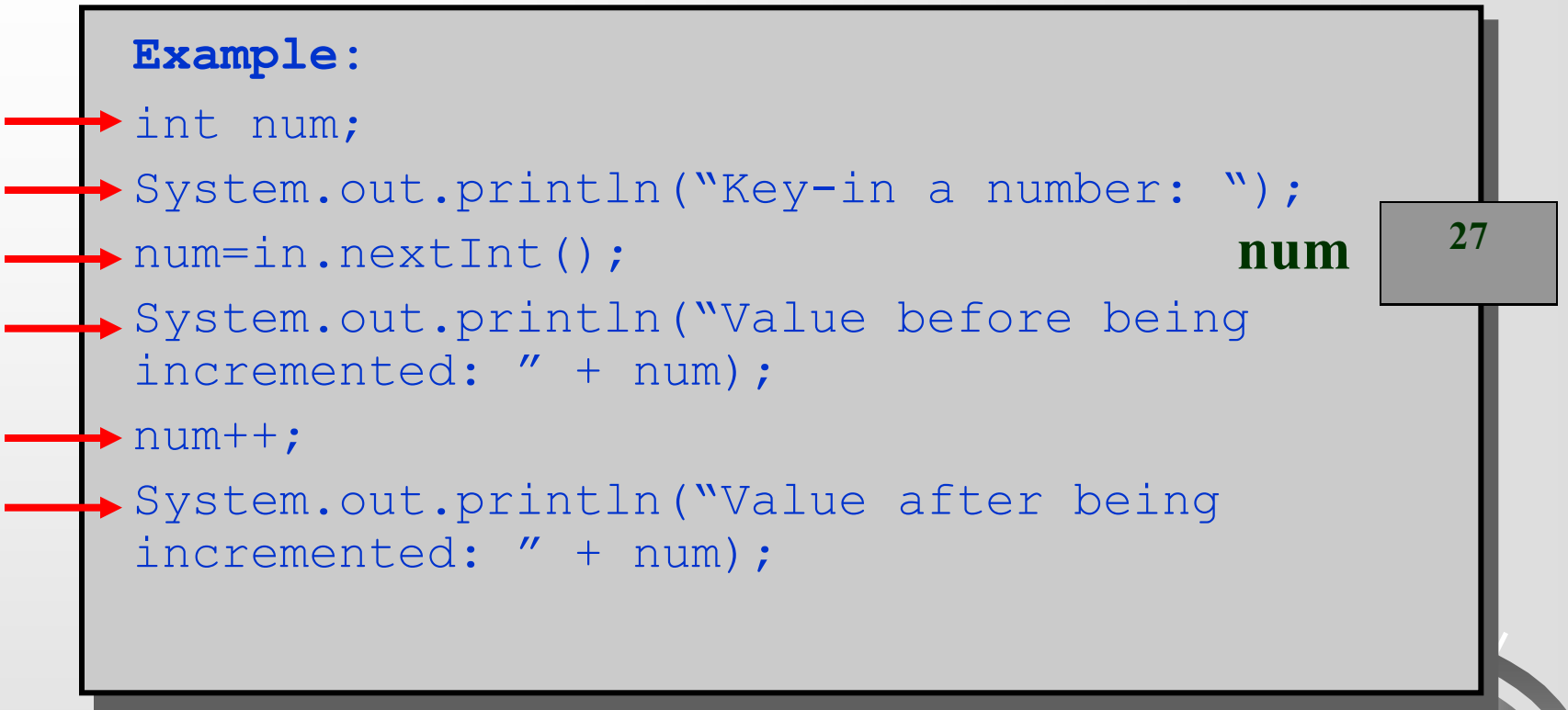
- Increment and Decrement operators can be either in prefix or postfix forms

Expression	Description
i++	Value of i is incremented after being used in the expression
++i	Value of i is incremented before being used in the expression
i--	Value of i is decremented after being used in the expression
--i	Value of i is decremented before being used in the expression

Increment and Decrement

Example:

```
int num;
System.out.println("Key-in a number: ");
num=in.nextInt();
System.out.println("Value before being
incremented: " + num);
num++;
System.out.println("Value after being
incremented: " + num);
```




```
Key-in a number: 26
Value before being incremented: 26
Value after being incremented: 27 _
```

Prefix and Postfix

Example:

j = i++ - 2

similar to

j = i – 2;

i = i + 1;

Prefix and Postfix

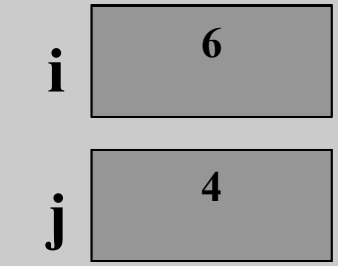
Example:

j = ++i - 2

similar to

i = i + 1;

j = i – 2;



Something to ponder …

The earlier expressions contain only one type at a time. What if the expression contains more than one type?

Conversion Rules

- When performing a binary operation involving two operands of different type, Java automatically converts the operand based on the following rules:
  - If one of the operands is double, the other is converted into double
  - Otherwise, if one of the operands is float, the other is converted to float
  - Otherwise, if one of the operands is long, the other is converted to long
  - Otherwise, both operands are converted to int

Conversion Problems Example

```
byte i = 100;

long k = i * 3 + 4;

double d = i * 3.1 + k / 2;
```

Type Conversion (Casting)

- Used to avoid implicit type coercion.
- Syntax:

  (dataTypeName) expression

- Expression evaluated first, then type converted to: dataTypeName
- Examples:
  - `(int)(7.9 + 6.7) = 14`
  - `(int)(7.9) + (int)(6.7) = 13`

Type Conversion (Casting)

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)

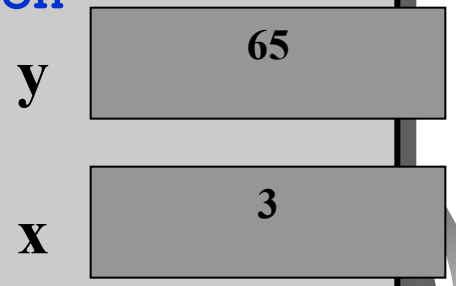
What is wrong? int x = 5 / 2.0;

Type Conversion (Casting)

- Casting is used to compute a value that is equivalent to its operand's value (based on the stated data type)

Example:

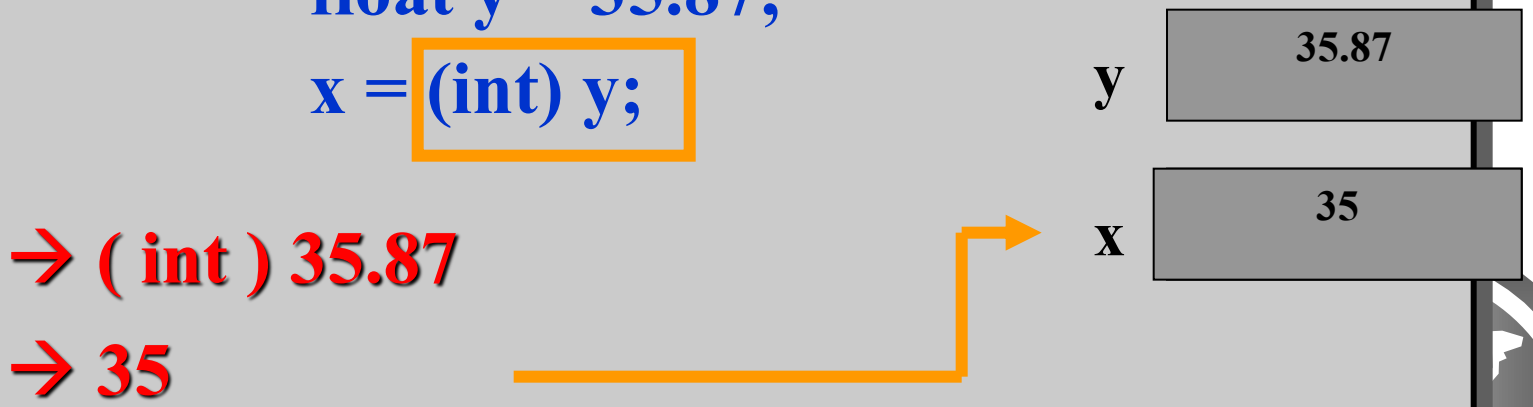
```
int x = (int) 3.0; // type narrowing
int y = (int) 65.76;//truncation
```



Type Casting Example

Example:

```
int x;
float y = 35.87;
x = (int) y;
```



→ (int) 35.87

→ 35

Problem: Keeping Two Digits After Decimal Points

- Write a program that displays the sales tax with two digits after the decimal point.

SalesTax Run

Casting in an Augmented Expression

In Java, an augmented expression of the form **x1 op= x2** is implemented as **x1 = (T)(x1 op x2)**, where **T** is the type for **x1**. Therefore, the following code is correct.

```
int sum = 0;
sum += 4.5; // sum becomes 4 after this statement
```

sum += 4.5 is equivalent to **sum = (int)(sum + 4.5)**.

Exercise

1. What is wrong?

```
int i = 5;
double d = 2.0;
int  x = i/d;
```

Exercise

2. Assume that int a = 10 and double d = 10.0, and that each expressions is independent. What are the results of the following expressions?

a) a = 25 / 4;

b) a = 15 % 9 + 2 * 5 – 3;

c) d += 4.3 * 2 + (a--);

d) d -= 6.2 * 4 + ++a;

Exercise

3. What is the result of the following code:

a)

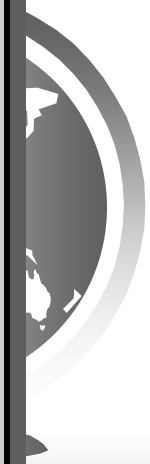
```
byte  x;
int  y;
byte  z;

x = 50;
y = 10;
z = x + y;
```

b)

```
byte  x;
int  y;
int  z;

x = 100;
y = 100;
z = x + y;
```



Exercise

4. What is the result of the following code:

a)

```
int x;
double  y;
float z;

x = 50;
y = 10.0;
z = x + y;
```

b)

```
byte  x;
double  y;
double  z;

x = 1000;
y = 1000.0;
z = x + y;
```

Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

double interestRate = **0.05**;

double interest = interestrate * **45**;

Common Error 2: Integer Overflow

```
int value = 2147483647 + 1;
// value will actually be -2147483648
```

Common Error 3: Round-off Errors

System.out.println(**1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1**);

System.out.println(**1.0 - 0.9**);

Common Error 4: Unintended Integer Division

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2;
System.out.println(average);
```

(a)

```
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
System.out.println(average);
```

(b)