# CEN 419
# Introduction to Java Programming

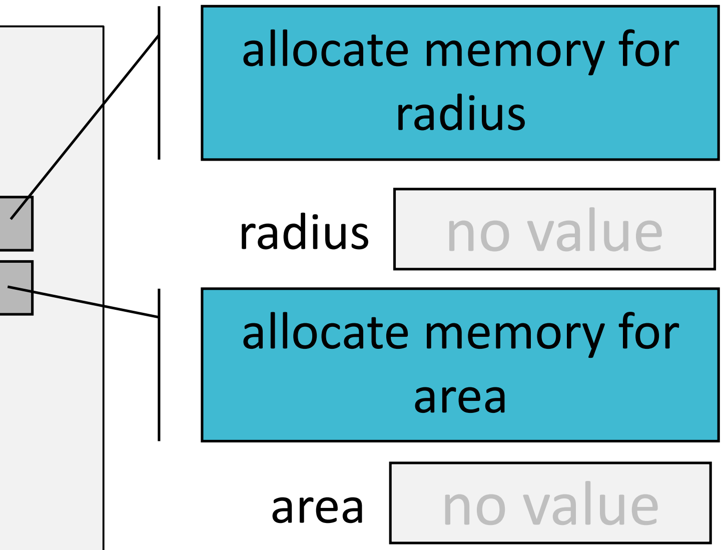
Dr. H. Esin ÜNAL

FALL 2021

*Slides are modified from original slides of Y. Daniel Liang*

## Computing the Area of a Circle

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



# Computing the Area of a Circle

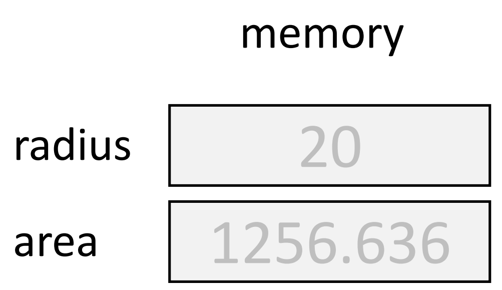
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

| | |
|---|---|
| radius | 20 |
| area | no value |

assign 20 to radius

## Computing the Area of a Circle

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



compute area and assign it to variable area

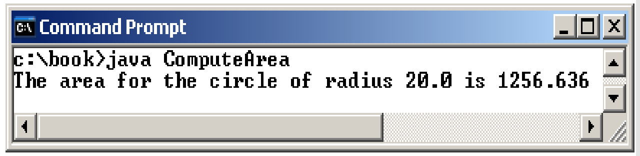
## Computing the Area of a Circle

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

memory

| | |
|---|---|
| radius | 20 |
| area | 1256.636 |

print a message to the console



```
c:\book>java ComputeArea
The area for the circle of radius 20.0 is 1256.636
```

String concatenation operator

## Reading Input from the Console

1. Import the Scanner class in the java.util package:

```
import java.util.Scanner;
```

2. Create a Scanner object:

```
Scanner input = new Scanner(System.in);
```

3. Use the method nextDouble() to obtain a double value.

For example,

```
System.out.print("Enter a double value: ");
Scanner input = new Scanner(System.in);
double d = input.nextDouble();
```

## Now It’s Your Turn!!!

Compute the Area with console input

## Compute Area with Console Input

```
import java.util.Scanner; // Scanner is in the java.util package
public class ComputeAreaWithConsoleInput {
    public static void main(String[] args) {
    // Create a Scanner object
    Scanner input = new Scanner(System.in);
    // Prompt the user to enter a radius
    System.out.print("Enter a number for radius: ");
    double radius = input.nextDouble();
     // Compute area
     double area = radius * radius * 3.14159;
     // Display result
     System.out.println("The area for the circle of radius " +
          radius + " is " + area);
     }
}
```

**Compute Area with Console Input**

Intro to Java Programming, Y. Daniel Liang - ComputeAreaWithConsoleInput.java (pearsoncmg.com)

## Import Statement

- There are two types of import statements:
  - ***Specific Import:*** Specifies a single class in the import statement.

```
import java.util.Scanner;
```

  - ***Wildcard Import:*** Imports all the classes in a package by using the asterisk as the wildcard.

```
import java.util.*;
```

- The information for the classes in an imported package is not read in at compile time or runtime unless the class is used in the program. No performance difference.

## Identifiers

- ***Identifiers <u>are the names</u> that identify the elements such as classes, methods, and variables in a program.***
- An identifier is a sequence of characters that consist of ***letters, digits, underscores (_), and dollar signs ($)***.
- An identifier must start with a letter, an underscore (_), or a dollar sign ($). **<u>It cannot start with a digit.</u>**
- An identifier cannot be a reserved word.
- An identifier cannot be `true`, `false`, or `null`.

## Identifiers

- An identifier can be of any length.
- Give descriptive names to identifiers
- If identifiers are written wrong than syntax error occurs
- Since Java is case sensitive, area, Area and AREA are all different identifiers.
- Examples:
  - ✓ *$2, ComputeArea, radius* are legal identifiers
  - ✓ *2A , d+4* are NOT legal identifiers

## Variables

- ***Variables are used to represent values that may be changed in the program.***
- Variables are for representing data of a certain type.
- To use a variable, you declare it by telling the compiler its name as well as what type of data it can store. The *variable declaration* tells the compiler to allocate appropriate memory space for the variable based on its data type.
- The syntax for declaring a variable is:

*datatype* variableName;

## Declaring Variables

```
int x;          // Declare x to be an
                // integer variable;
double radius;  // Declare radius to
                  // be a double variable;
char a;         // Declare a to be a
                 // character variable;
```

- If variables are of the same type, they can be declared together, as follows:

  *datatype* variable_1, variable_2, ..., variable_n;

```
int i, j, k; // Declare i, j, and k as int variables
```

## Assignment Statements

- ***An assignment statement designates a value for a variable.***
- ***An assignment statement can be used as an expression in Java.***
- After declaring a variable you  can assign a value to it:

variable = expression;

- Examples:

```
x = 1;              // Assign 1 to x;
radius = 1.0;       // Assign 1.0 to radius;
a = 'A';            // Assign 'A' to a;
area = radius * radius * 3.14159; // Compute area
```

## Assignment Statements

- You can use a variable in an expression. A variable can also be used in both sides of the = operator.

```
    x = x + 1;
```

- If a value is assigned to multiple variables, you can use this syntax:

```
   i = j = k = 1;
```

- You can declare and initialize in one step:

```
  int x = 1;
  double d = 1.4;
```

## Named Constants

- ***A named constant is an identifier that represents a permanent value***.
- Do not change during the execution of a program

  *final datatype* CONSTANTNAME = VALUE;

- A constant must be declared and initialized in the same statement.
- The word **final** is a Java keyword for declaring a constant.

```
final double PI = 3.14159;
final int SIZE = 3;
```

## Naming Conventions

- Choose meaningful and descriptive names.
- **Variables and method names**:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name.
    - For example, the variables `radius` and `area`, and the method `computeArea`.

## Naming Conventions

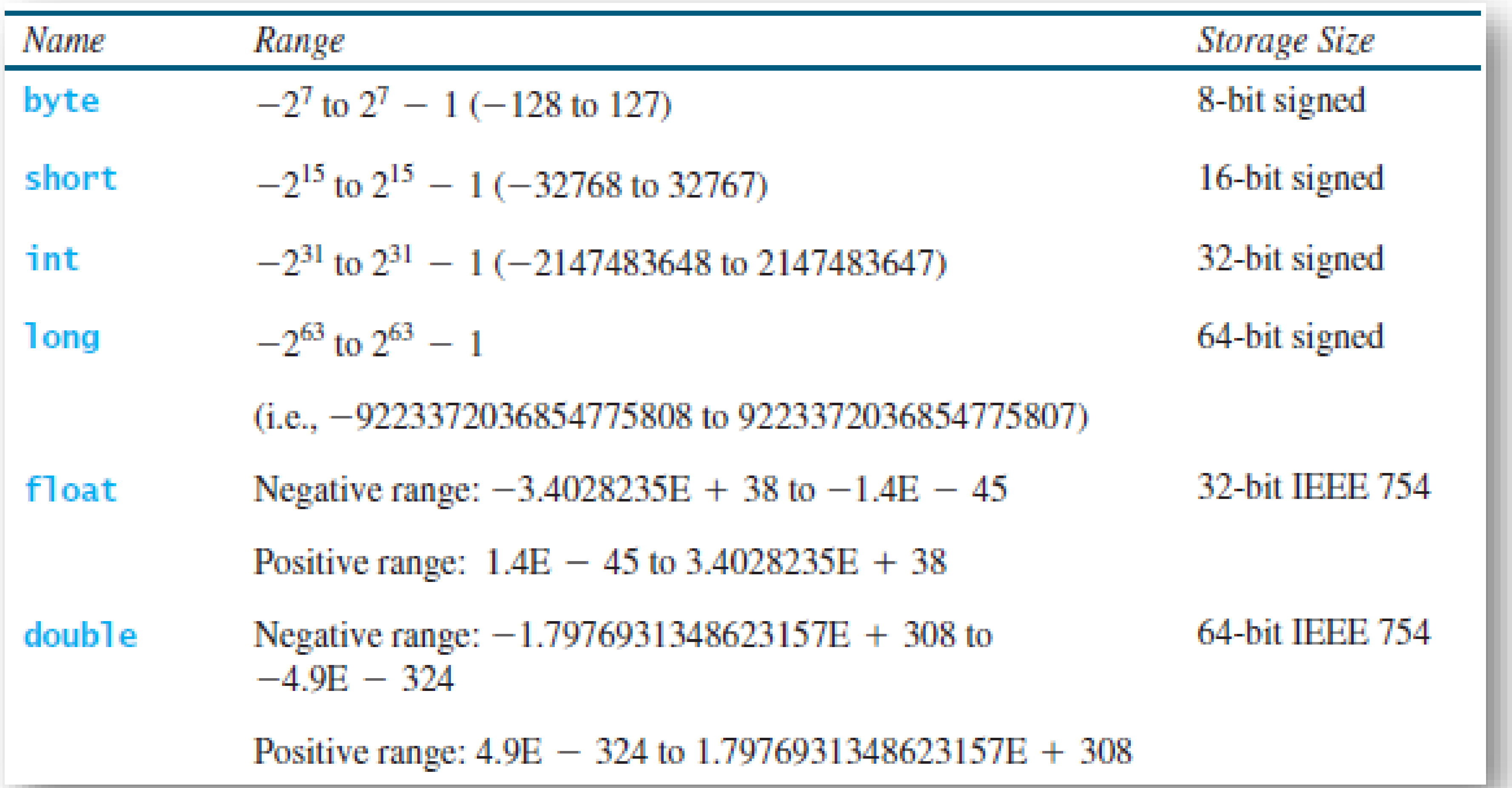
- **Class names**:
  - Capitalize the first letter of each word in the name.
    - For example, the class name `ComputeArea` and `System`

- **Constants**:
  - Capitalize all letters in constants, and use underscores to connect words.
    - For example, the constant `PI` and `MAX_VALUE`

## Numeric Data Types and Operations

- ***Java has six numeric types for integers and floating-point numbers with operators +, -, *, /, and %.***
- Java uses four types for integers: **byte**, **short**, **int**, and **long**.
- Java uses two types for floating-point numbers: **float** and **double**.

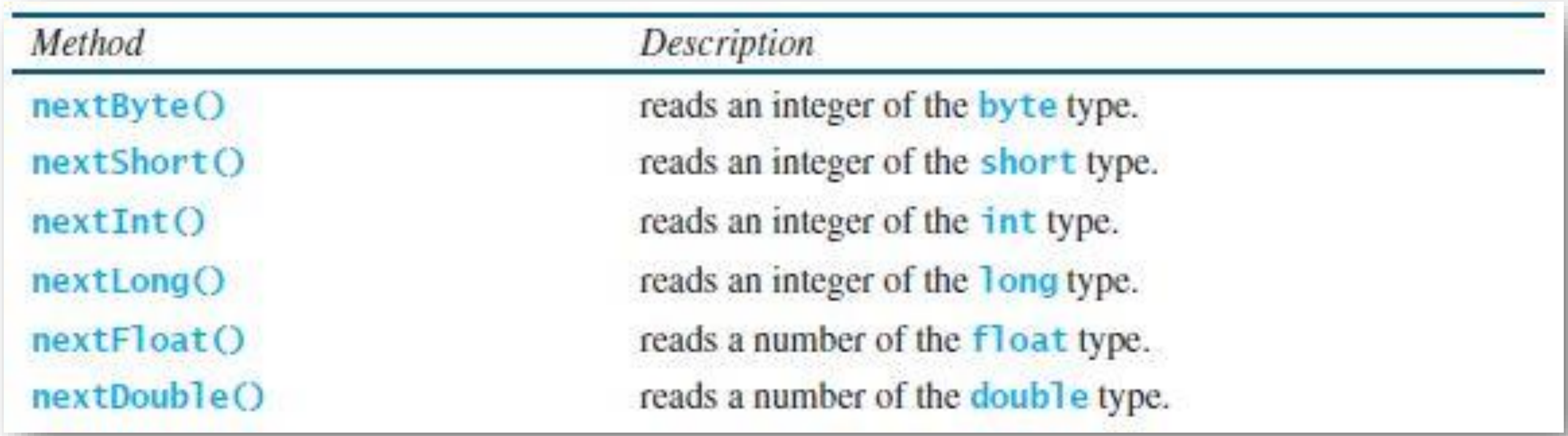
# Numeric Data Types



| *Name* | *Range* | *Storage Size* |
|---|---|---|
| `byte` | $-2^7$ to $2^7 - 1$ (−128 to 127) | 8-bit signed |
| `short` | $-2^{15}$ to $2^{15} - 1$ (−32768 to 32767) | 16-bit signed |
| `int` | $-2^{31}$ to $2^{31} - 1$ (−2147483648 to 2147483647) | 32-bit signed |
| `long` | $-2^{63}$ to $2^{63} - 1$ | 64-bit signed |
| | (i.e., −9223372036854775808 to 9223372036854775807) | |
| `float` | Negative range: −3.4028235E + 38 to −1.4E − 45 | 32-bit IEEE 754 |
| | Positive range: 1.4E − 45 to 3.4028235E + 38 | |
| `double` | Negative range: −1.7976931348623157E + 308 to −4.9E − 324 | 64-bit IEEE 754 |
| | Positive range: 4.9E − 324 to 1.7976931348623157E + 308 | |

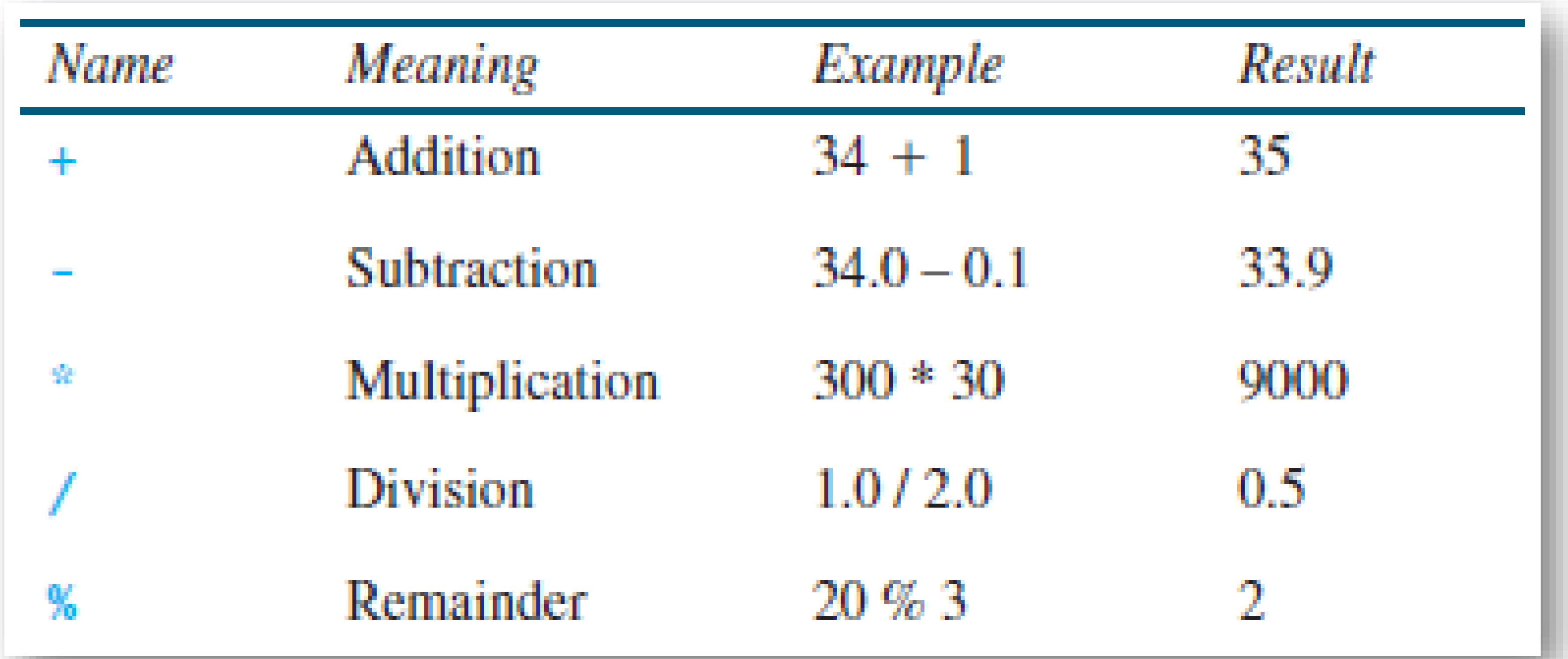
## Reading Numbers from the Keyboard

```
Scanner input = new Scanner(System.in);
int value = input.nextInt();
```



| *Method* | *Description* |
|---|---|
| `nextByte()` | reads an integer of the `byte` type. |
| `nextShort()` | reads an integer of the `short` type. |
| `nextInt()` | reads an integer of the `int` type. |
| `nextLong()` | reads an integer of the `long` type. |
| `nextFloat()` | reads a number of the `float` type. |
| `nextDouble()` | reads a number of the `double` type. |

## 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 |

## Numeric Operators (/)

- When both operands of a division are integers, the result of the division is the quotient and the fractional part is truncated.
  - ✓**5 / 2** yields **2**, not **2.5**
  - ✓**–5 / 2** yields **-2**, not **–2.5**

- To perform a float-point division, one of the operands must be a floating-point number.
  - ✓**5.0 / 2** yields **2.5**

## Problem: Converting Temperatures

Remember the program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\tfrac{5}{9})(fahrenheit - 32)$$

You have to write this expression as:

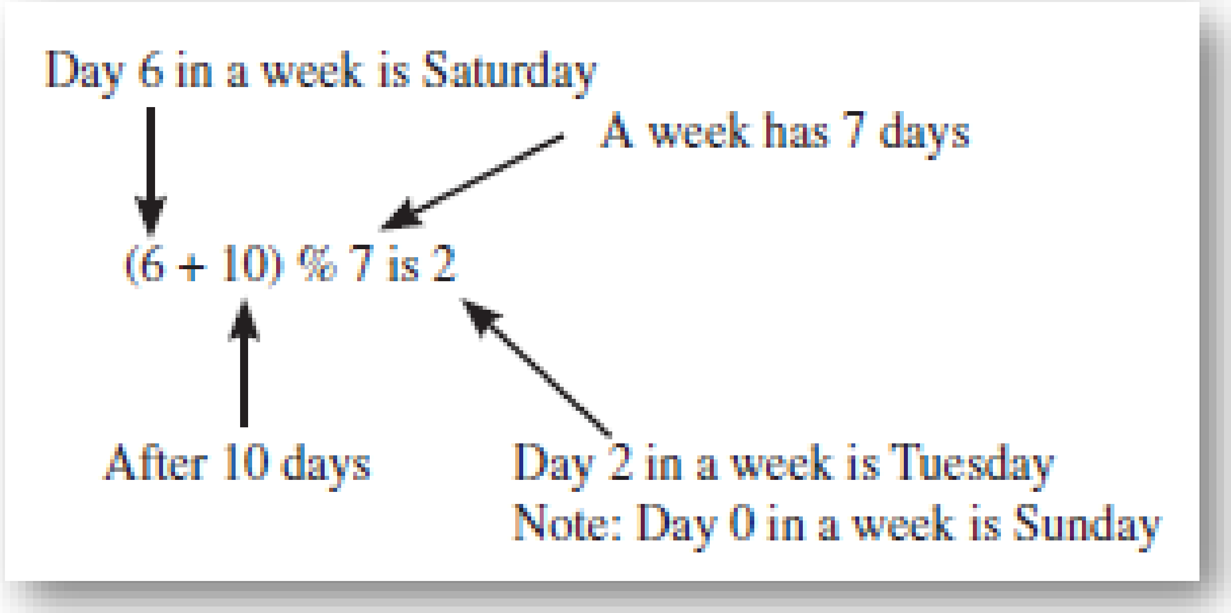
```
celsius = (5.0 / 9) * (fahrenheit – 32)
```

## Numeric Operators (%)

- The **%** operator is often used for positive integers
- The remainder is negative only if the dividend is negative
  - ✓**-7 % 3** yields **-1**
  - ✓**-12 % 4** yields **0**
  - ✓**-26 % -8** yields **-2**
  - ✓**20 % -13** yields **7**

## Numeric Operators

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:


Day 6 in a week is Saturday
A week has 7 days
(6 + 10) % 7 is 2
After 10 days
Day 2 in a week is Tuesday
Note: Day 0 in a week is Sunday

## Pop-Up 2

Displaying Time:

Write a program that obtains minutes and seconds from seconds.

## Problem: Displaying Time

Write a program that obtains minutes and seconds from seconds.

**Display Time**

Intro to Java Programming, Y. Daniel Liang - DisplayTime.java (pearsoncmg.com)

## Exponent Operations

- The **Math.pow(a, b)** method can be used to compute $a^b$.
- The **pow** method is defined in the **Math** class in the Java API.
- You invoke the method using the syntax **Math.pow(a, b)**
  - 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

## Numeric 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 $-2^{31}$ (-2147483648) to $2^{31}-1$ (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). (2147483648L)

## 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.

## 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
  - 1.23456e-2 is equivalent to 0.0123456
- E (or e) represents an exponent and it can be either in lowercase or uppercase.

## Arithmetic Expressions

*Java expressions are evaluated in the same way as arithmetic expressions.*

$$\boxed{\frac{3+4x}{5}-\frac{10(y-5)(a+b+c)}{x}+9(\frac{4}{x}+\frac{9+x}{y})}$$

is translated to

```
(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)
```

## Operator Precedence

- Operators contained within pairs of parentheses are evaluated first.
- After that, multiplication, division, and remainder operators are applied. If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.
- Addition and subtraction operators are applied last. If an expression contains several addition and subtraction operators, they are applied from left to right.

## Augmented Assignment Operators

- The operators **+**, **-**, *****, **/**, and **%** can be combined with the assignment operator to form augmented operators.

| **Operator** | **Example** | **Equivalent** |
|---|---|---|
| `+=` | `i += 8` | `i = i + 8` |
| `-=` | `f -= 8.0` | `f = f - 8.0` |
| `*=` | `i *= 8` | `i = i * 8` |
| `/=` | `i /= 8` | `i = i / 8` |
| `%=` | `i %= 8` | `i = i % 8` |

## Augmented Assignment Operators

- The augmented assignment operator **is performed last after all the other operators** in the expression are evaluated. For example,

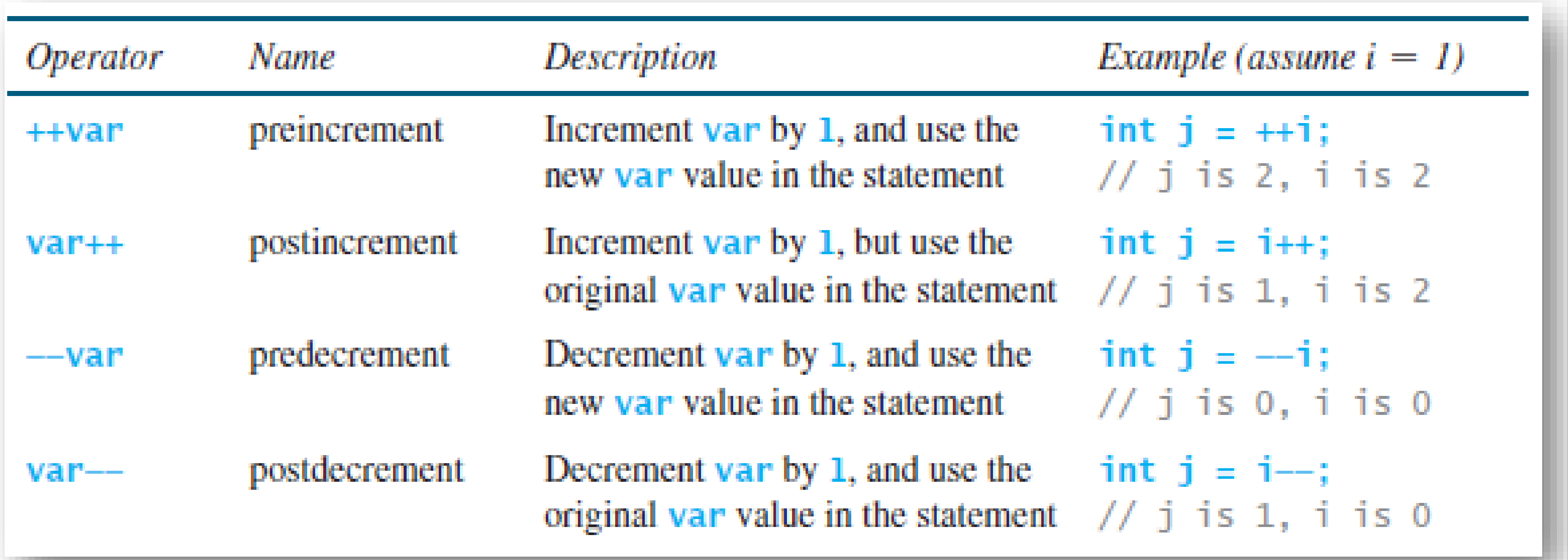
  x /= **4** + **5.5** * **1.5**;

  is same as

  x = x / (**4** + **5.5** * **1.5**);

## Increment and Decrement Operators

- *The increment operator (++) and decrement operator (– –) are for incrementing and decrementing a variable by 1.*



| *Operator* | *Name* | *Description* | *Example (assume i = 1)* |
|---|---|---|---|
| `++var` | preincrement | Increment `var` by `1`, and use the new `var` value in the statement | `int j = ++i;` `// j is 2, i is 2` |
| `var++` | postincrement | Increment `var` by `1`, but use the original `var` value in the statement | `int j = i++;` `// j is 1, i is 2` |
| `--var` | predecrement | Decrement `var` by `1`, and use the new `var` value in the statement | `int j = --i;` `// j is 0, i is 0` |
| `var--` | postdecrement | Decrement `var` by `1`, and use the original `var` value in the statement | `int j = i--;` `// j is 1, i is 0` |

## Increment and Decrement Operators

- Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read. Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this:

**int k = ++i + i**

```
int i = 10;
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;
int newNum = 10 * i;
```

## Numeric Type Conversion

- You can always assign a value to a numeric variable whose type supports a larger range of values
- Consider the following statements:

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

$$\xrightarrow{\texttt{range increases}}$$

`byte, short, int, long, float, double`

## Conversion Rules

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 Casting

- You cannot, however, assign a value to a variable of a type with a smaller range unless you use *type casting.*
- **Casting is an operation that converts a value of one data type into a value of another data type.**
- Casting a type with a small range to a type with a larger range is known as ***widening a type***.
  - `double d = 3;` (type widening)
- Casting a type with a large range to a type with a smaller range is known as ***narrowing a type***.
  - `int i = (int)3.9;` (Fraction part is truncated)

## Type Casting

- Java will automatically widen a type, but you must narrow a type explicitly.
- What is wrong?
  - int x = 5 / 2.0;
  - A compiler error will occur

$$\xrightarrow{\texttt{range increases}}$$

`byte, short, int, long, float, double`

## 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)**.

## Problem: Computing Loan Payments

This program lets the user enter the interest rate, number of years, and loan amount, and computes monthly payment and total payment.

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

**Compute Loan**

Intro to Java Programming, Y. Daniel Liang - ComputeLoan.java (pearsoncmg.com)

## Problem: Monetary Units

This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies. Your program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.

Compute Change

Intro to Java Programming, Y. Daniel Liang - ComputeChange.java (pearsoncmg.com)

## Common Errors and Pitfalls

- Common elementary programming errors often involve:
  - undeclared variables
  - uninitialized variables
  - integer overflow
  - unintended integer division
  - round-off errors

## Undeclared/ Uninitialized Variables and Unused Variables

```
double interestRate = 0.05;
double interest = interestrate * 45;
```

This code is wrong, because **interestRate** is assigned a value **0.05**; but **interestrate** has not been declared and initialized. Java is case sensitive, so it considers **interestRate** and **interestrate** to be two different variables.

```
double interestRate = 0.05;
double taxRate = 0.05;
double interest = interestRate * 45;
System.out.println("Interest is " + interest);
```

If a variable is declared, but not used in the program, it might be a potential programming error. So, you should remove the unused variable from your program. In this code **taxRate** is never used. It should be removed from the code

## Integer Overflow

- Numbers are stored with a limited numbers of digits. When a variable is assigned a value that is too large (*in size*) to be stored, it causes *overflow*.

```
int value = 2147483647 + 1;
```

Executing the statement causes overflow, because the largest value that can be stored in a variable of the **int** type is **2147483647**. **2147483648** will be too large for an **int** value.

## Round-off Errors

- *A round-off error*, also called a *rounding error*, is the difference between the calculated approximation of a number and its exact mathematical value.

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

displays **0.5000000000000001**, not **0.5**, and

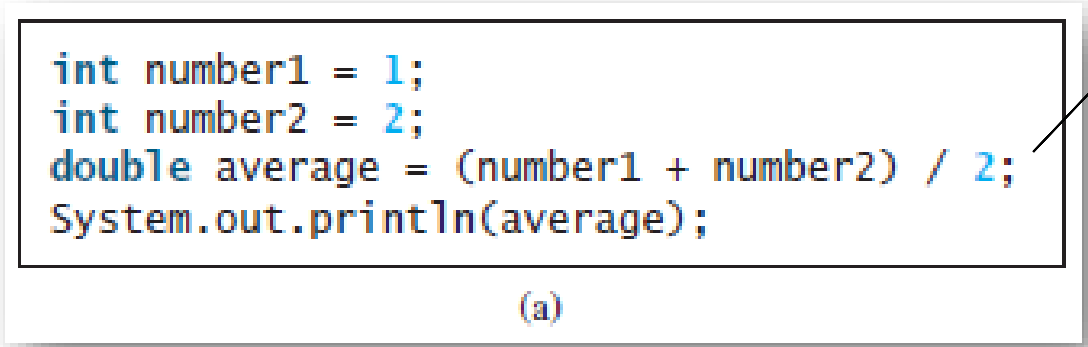
```
System.out.println(1.0 - 0.9);
```

displays **0.09999999999999998**, not **0.1**.

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

## Unintended Integer Division

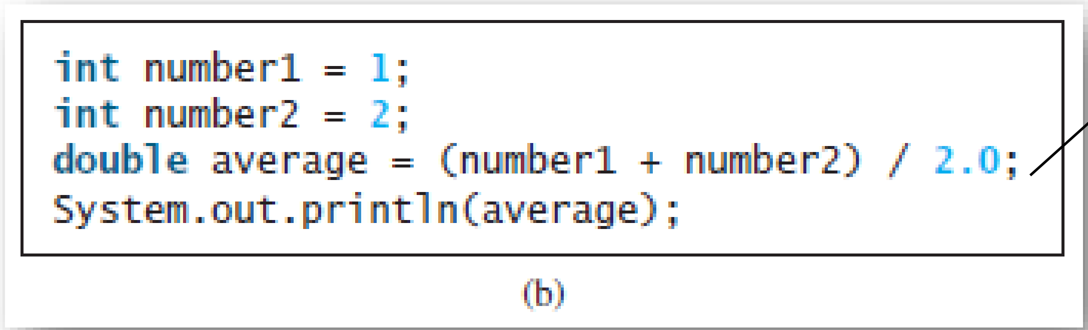
- Java uses the same divide operator, namely **/**, to perform both integer and floating-point division.



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

(a)

Average=1



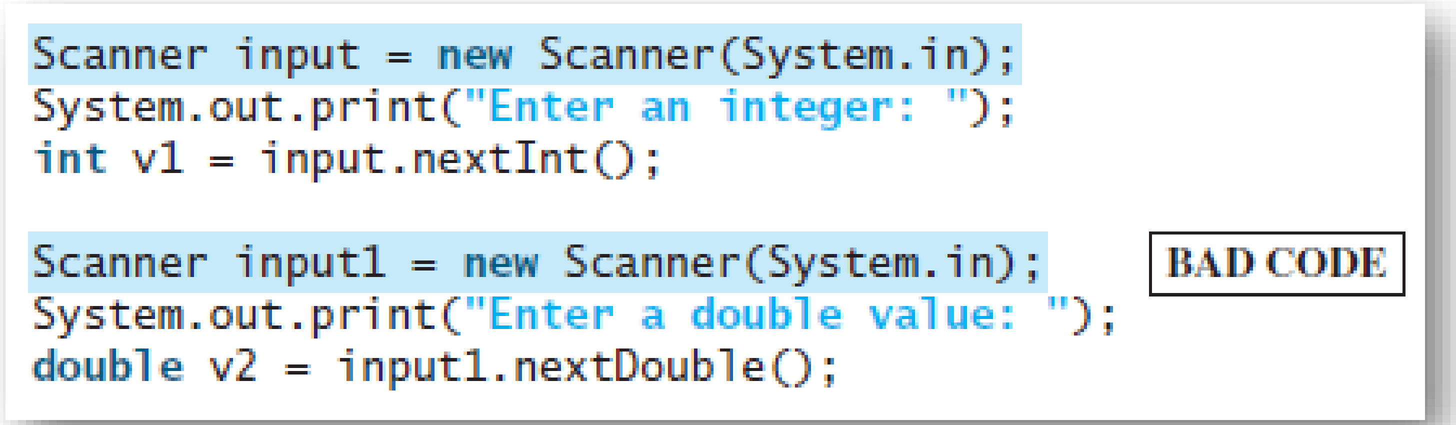
```java
int number1 = 1;
int number2 = 2;
double average = (number1 + number2) / 2.0;
System.out.println(average);
```

(b)

Average=1,5

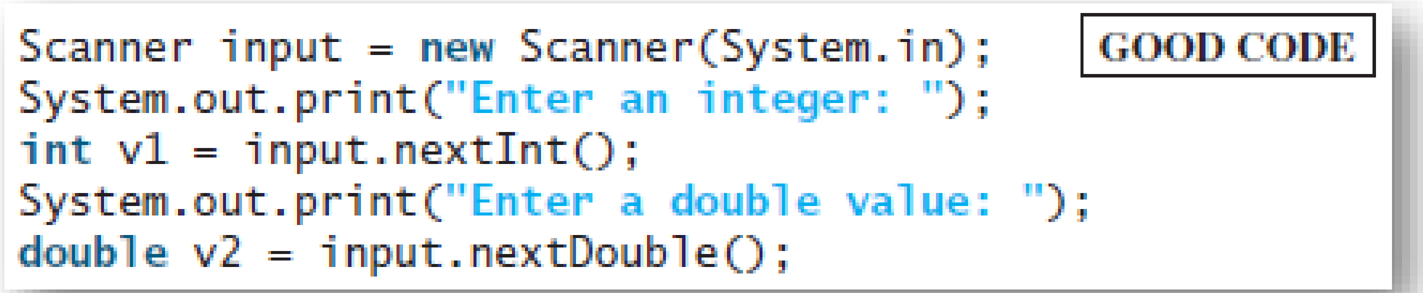
## Redundant Input Objects

New programmers often write the code to create multiple input objects for each input.



```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();

Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();
```

BAD CODE



```
Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();
System.out.print("Enter a double value: ");
double v2 = input.nextDouble();
```

GOOD CODE