COMP20220
Programming II (Conversion)

Michael O'Mahony

Chapter 2 Elementary Programming

Objectives

✦ To write Java programs to perform simple computations (§2.2).

✦ To obtain input from the console using the **Scanner** class (§2.3).

✦ To use identifiers to name variables, constants, methods, and classes (§2.4).

✦ To use variables to store data (§§2.5–2.6).

✦ To program with assignment statements (§2.6).

✦ To use constants to store permanent data (§2.7).

✦ To name classes, variables, and constants by following their naming conventions (§2.8).

✦ To explore Java numeric primitive data types: **byte**, **short**, **int**, **long**, **float**, and **double** (§2.9.1).

✦ To perform operations using operators **+**, **-**, *****, **/**, and **%** (§2.9.3).

✦ To perform exponent operations using **Math.pow(a, b)** (§2.9.4).

✦ To write and evaluate numeric expressions (§2.11).

✦ To use augmented assignment operators (§2.13).

✦ To distinguish between postincrement and preincrement and between postdecrement and predecrement (§2.14).

✦ To cast the value of one type to another type (§2.15).

Motivations

In the preceding chapter, we looked at how to create, compile, and run a Java program…

Starting from this chapter, you will learn how to solve practical problems programmatically.

Through these problems, we will introduce Java primitive data types, variables, constants, operators, expressions, and input and output.

Example Program

This program computes the area of the circle and displays the result.

[ComputeArea]

Create/Trace a Program Execution

```
public class ComputeArea {




}
```

Create/Trace a Program Execution

```
public class ComputeArea {
  // Main method
  public static void main(String[] args) {



    }
}
```

Create/Trace a Program Execution

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




  }
}
```
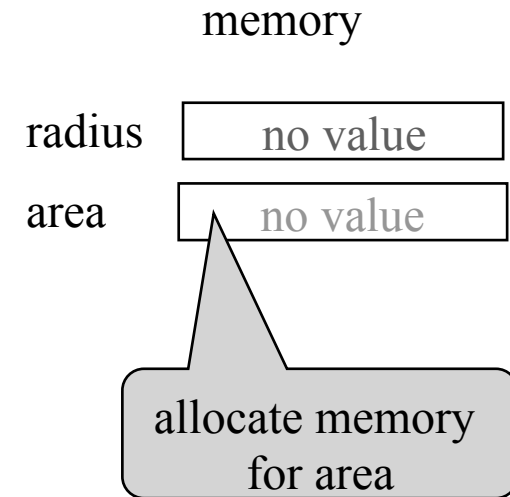
allocate memory for radius

radius | no value

Create/Trace a Program Execution

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



  }
}
```

memory

radius   no value

area   no value

allocate memory
for area

Create/Trace a Program Execution

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

    // Assign a radius
    radius = 20;



  }
}
```
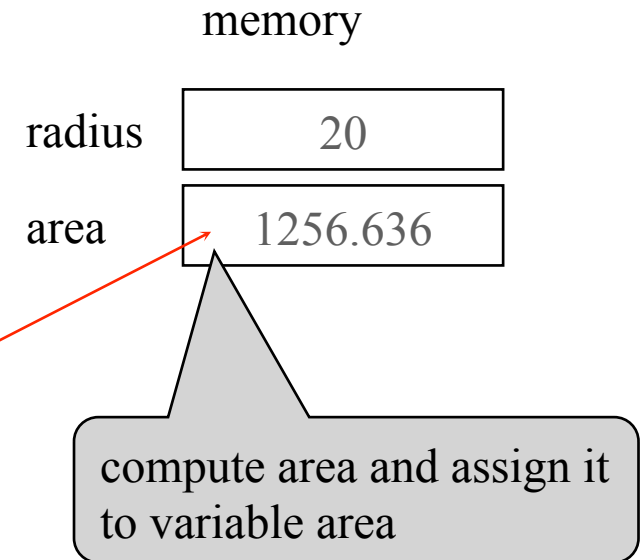
assign 20 to radius

radius   20

area   no value

Create/Trace a Program Execution

```
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;



  }
}
```

memory

radius   20

area   1256.636

compute area and assign it to variable area

Create/Trace a Program Execution

```java
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);
  }
}
```
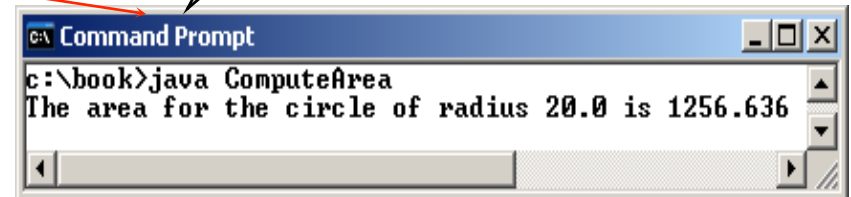
memory

| radius | 20 |

| area | 1256.636 |

print a message to the console

Command Prompt

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

Identifiers

✦ Identifiers are the names that identify the elements (classes, methods, 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 ($) – it cannot start with a digit.

✦ An identifier cannot be a reserved word (e.g. `int`, `for`, `if`, `else`, `class`…). See Appendix A, "Java Keywords," for a list of reserved words.

✦ An identifier cannot be `true`, `false`, or `null`.

✦ An identifier can be of any length.

Declaring Variables

Variables are used to represent values that may be changed in a program

In Java variables have types (!)

```
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
```

Assignment Statements

```
x = 1;            // Assign 1 to x;

radius = 1.0;     // Assign 1.0 to radius;

a = 'A';          // Assign 'A' to a;
```

Declaring and Initializing in One Step

```
int x = 1;


double radius = 1.0;


char a = 'A';
```

Assignment Statements

An *assignment statement* designates a value for a variable

The *assignment operator* is the equal size (=)

The syntax for assignment statements is:

```
variable = expression;
```

An *expression* represents a computation involving values, variables and operators that evaluates to a value:

```
double radius = 5.0;
double area = radius * radius * 3.14159;
```

Numerical 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 $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
float	Negative range: $-3.4028235E + 38$ to $-1.4E - 45$ Positive range: $1.4E - 45$ to $3.4028235E + 38$	32-bit IEEE 754
double	Negative range: $-1.7976931348623157E + 308$ to $-4.9E - 324$ Positive range: $4.9E - 324$ to $1.7976931348623157E + 308$	64-bit IEEE 754

Number Literals

A *literal* is a constant value that appears directly in the program.

For example, 34 and 5.0 are literals in the following statements:

```
int i = 34;
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 (generally use `int` for integers).

Floating-Point Literals

Floating-point literals are written with a decimal point – for example: 5.0

Java has two numeric types for floating-point numbers – `float` and `double`:

```
float num = 1.0f; or float num = 1.0F;

double num = 1.0d; or double num = 1.0D;
```

By default, a floating-point literal is treated as a `double` type value:

```
double num = 1.0;
```

Use `double` type – values are more accurate than `float` type values:

```
System.out.println(1.0F / 3.0F); // prints 0.33333334

System.out.println(1.0 / 3.0);   // prints 0.3333333333333333
```

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.

Named Constants

A named constant is a variable that represents a permanent value.

General form:

```
final datatype CONSTANTNAME = value;
```

Examples:

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

Note – a final variable can only be initialized once.

Naming Conventions

✦ Choose meaningful and descriptive names.

✦ Variable names:

  – Use lowercase: e.g. `radius` and `area`.
  – 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: e.g. `interestRate`.

✦ Class names:

  – Capitalize the first letter of each word in the name: e.g. `Welcome` and `ComputeArea`.

✦ Constants:

  – Capitalize all letters in constants, and use underscores to connect words: e.g. `PI` and `MAX_VALUE`

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

How to Evaluate an Expression

The value of a Java expression and its corresponding arithmetic expression are the same.

What is the value of the following expression?

```
3 + 4 * 4 + 5 * (4 + 3) - 1
```

How to Evaluate an Expression

3 + 4 * 4 + 5 * (4 + 3) - 1

     (1) inside parentheses first

3 + 4 * 4 + 5 * 7 - 1

     (2) multiplication

3 + 16 + 5 * 7 - 1

     (3) multiplication

3 + 16 + 35 - 1

     (4) addition

19 + 35 - 1

     (5) addition

54 - 1

     (6) subtraction

53

Arithmetic Expressions

Example – translate the following into a Java expression:

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

Answer:

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

Operator Precedence

When more than one operator is used in an expression, the following operator precedence rules are used to determine the order of evaluation:

- Multiplication, division, and remainder operators are applied first. 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.

Examples:

```
double d1 = 2.0 / 3 * 2; // d1 is 1.33

double d2 = 2.0 * 3 / 2; // d2 is 3.0
```

Type Casting

Casting is an operation that converts a value of one data type into a value of another data type.

Type widening refers to casting a type with a smaller range to a type with a larger range. Java automatically widens a type. For example:

```
double d = 3;
```

Type narrowing refers to casting a type with a larger range to a type with a smaller range. Must be done explicitly. For example:

```
int i = (int)3.9; // i is 3, fraction part is truncated
```

Numeric Type Conversion: Rules

When performing a binary operation involving two operands of different types (e.g. `x + y`), Java automatically converts operands 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.

Numeric Type Conversion: Rules

When performing a binary operation involving two operands of different types (e.g. `x + y`), 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.

What is wrong?   `int x = 5 / 2.0;`

Numeric Type Conversion: Rules

When performing a binary operation involving two operands of different types (e.g. `x + y`), 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.

What is wrong?   `int x = 5 / 2.0;`
Fix:

Numeric Type Conversion: Rules

When performing a binary operation involving two operands of different types (e.g. `x + y`), 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.

What is wrong?   `int x = 5 / 2.0;`
Fix: `int x = (int) (5 / 2.0);`

Numeric Type Conversion: Rules

When performing a binary operation involving two operands of
different types (e.g. `x + y`), 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.

What is wrong?   `int x = 5 / 2.0;`

Fix: `int x = (int) (5 / 2.0); // x = 2`

Numeric Type Conversion: Rules

When performing a binary operation involving two operands of different types (e.g. `x + y`), 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.

What is wrong?

```
int x = 5 / 2.0;
```

Fix:

```
int x = (int) (5 / 2.0); // x = 2
double x = 5 / 2.0;
```

Numeric Type Conversion: Rules

When performing a binary operation involving two operands of different types (e.g. `x + y`), 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.

What is wrong?

Fix:

```
int x = 5 / 2.0;
int x = (int) (5 / 2.0); // x = 2
double x = 5 / 2.0; // x = 2.5
```

Common Error: Unintended Integer Division

```
int n1 = 1;
int n2 = 2;
double average = (n1 + n2) / 2;
System.out.println(average);
```

Common Error: Unintended Integer Division

```
int n1 = 1;
int n2 = 2;
double average = (n1 + n2) / 2;
System.out.println(average);
```

```
int n1 = 1;
int n2 = 2;
double average = (n1 + n2) / 2.0;
System.out.println(average);
```

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

Increment and Decrement Operators

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, cont.

```
int i = 10;                    Same effect as    int newNum = 10 * i;
int newNum = 10 * i++;         ───────────►      i = i + 1;
```

```
int i = 10;                    Same effect as    i = i + 1;
int newNum = 10 * ++i;         ───────────►      int newNum = 10 * i;
```

Increment and Decrement Operators, cont.

Using increment and decrement operators makes expressions short, but it also can make them complex and difficult to read.

Avoid using these operators in expressions that modify multiple variables, or the same variable multiple times such as this:

```
int i = 1;
int k = ++i + i + i++;
```

What are the values of `i` and `k`??

Increment and Decrement Operators, cont.

Using increment and decrement operators makes expressions short, but it also can make them complex and difficult to read.

Avoid using these operators in expressions that modify multiple variables, or the same variable multiple times such as this:

```
int i = 1;
int k = ++i + i + i++;
```

What are the values of `i` and `k`?? (`i` is 3 and `k` is 6)

Standard Input/Output

`System.out` refers to the standard output device (monitor)

- To perform console output, you simply use the `println` method; e.g.

  ```
  System.out.println("Hello World!");
  ```

`System.in` refers to the standard input device (keyboard).

- Use the `Scanner` class to read input from `System.in`:

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

- Then, to read e.g. a double value from the keyboard, invoke the `nextDouble()` method:

  ```
  double d = input.nextDouble();
  ```

[ComputeAverage]

Reading Numbers from the Console

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

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.

See the Java API for more information – http://docs.oracle.com/javase/8/docs/api/

Common Pitfall: Redundant Objects

```
Scanner input1 = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input1.nextInt();


Scanner input2 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input2.nextDouble();
```

Common Pitfall: Redundant Objects

```java
Scanner input1 = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input1.nextInt();

Scanner input2 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input2.nextDouble();

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

Math Class

The Java API contains the `Math` class – useful for performing common mathematical functions.

Use the `Math.pow(a, b)` compute a^b – for example:

```
double d = Math.pow(2, 3); // d is 8.0
double d = Math.pow(4, 0.5); // d is 2.0
double d = Math.pow(2.5, 2); // d = 6.25
double d = Math.pow(2, -2); // d = 0.25
```

The Math class also provides the constant PI:

```
double area = radius * radius * Math.PI;
Double area = Math.pow(radius, 2) * Math.PI;
```

Compute the square root of a number as follows:

```
double d = 4.0;
double s = Math.sqrt(d); // s is 2.0
```

See the Java API for more information – http://docs.oracle.com/javase/8/docs/api/

Next Topics…

Chapter 3:

- boolean variables, relational operators, Boolean expressions
- if-else statements
- switch statements
- operator precedence