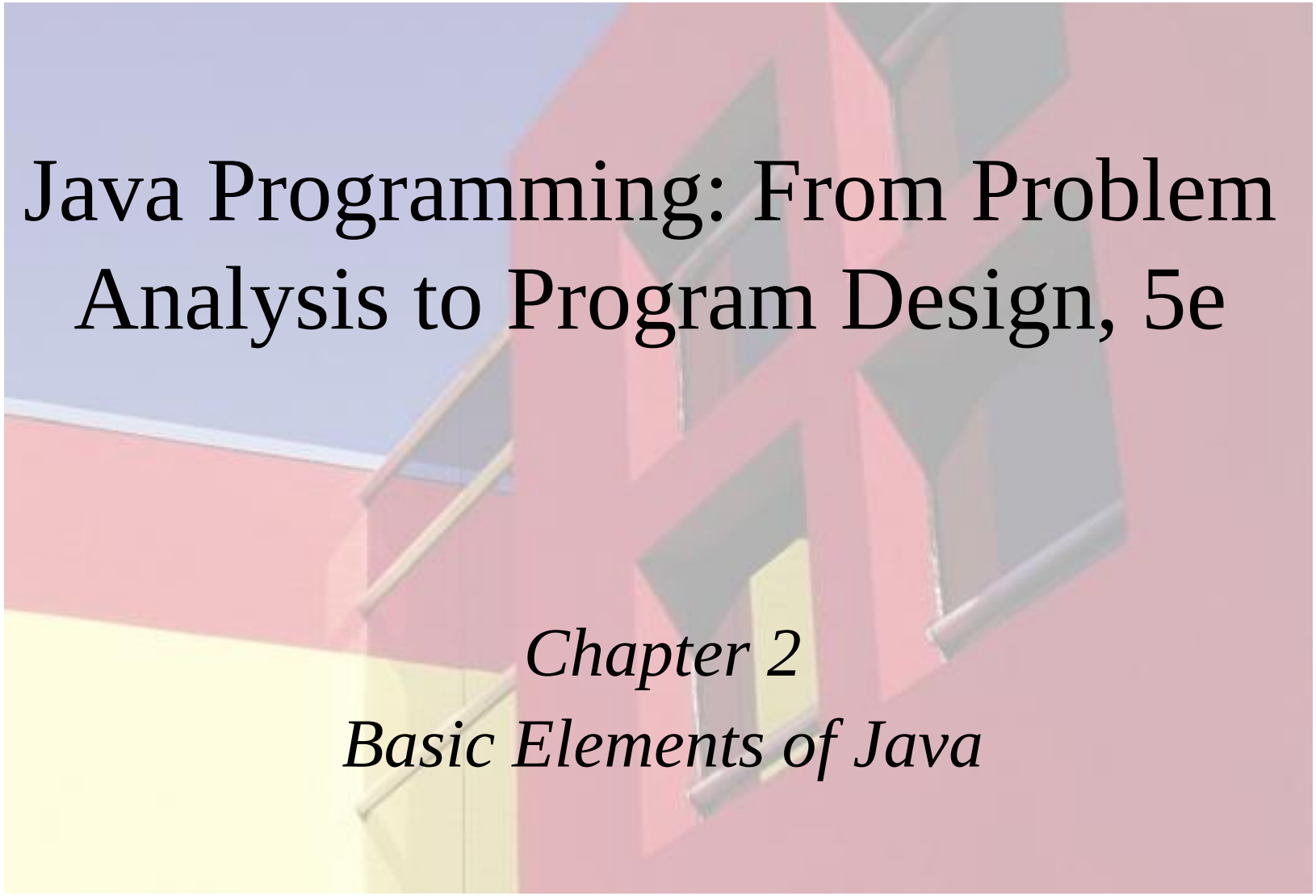


Java Programming: From Problem Analysis to Program Design, 5e

Chapter 2

Basic Elements of Java

Chapter Objectives

- Become familiar with the basic components of a Java program, including methods, special symbols, and identifiers
- Explore primitive data types
- Discover how to use arithmetic operators
- Examine how a program evaluates arithmetic expressions
- Explore how mixed expressions are evaluated

Chapter Objectives (continued)

- Learn about type casting
- Become familiar with the `String` type
- Learn what an assignment statement is and what it does
- Discover how to input data into memory by using input statements
- Become familiar with the use of increment and decrement operators

Chapter Objectives (continued)

- Examine ways to output results using output statements
- Learn how to import packages and why they are necessary
- Discover how to create a Java application program
- Explore how to properly structure a program, including using comments to document a program

Introduction

- **Computer program**: a sequence of statements whose objective is to accomplish a task
- **Programming**: process of planning and creating a program

The Basics of a Java Program

- Java program: collection of classes
- There is a main method in every Java application program
- Token: smallest individual unit of a program

A Java Program

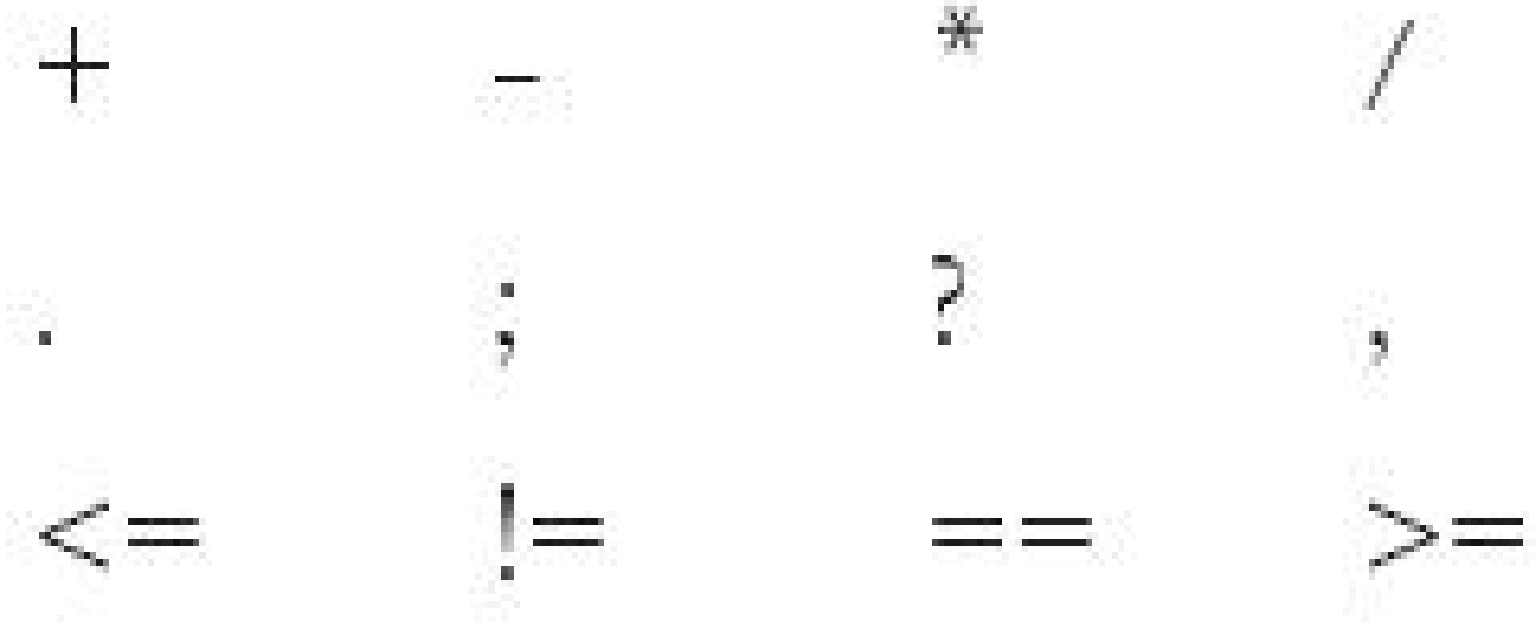
```
//*************************************************
// This is a simple Java program. It displays three lines
// of text, including the sum of two numbers.
//*************************************************

public class ASimpleJavaProgram
{
    public static void main(String[] args)
    {
        System.out.println("My first Java program.");
        System.out.println("The sum of 2 and 3 = " + 5);
        System.out.println("7 + 8 = " + (7 + 8));
    }
}
```

Sample Run:

```
My first Java program.
The sum of 2 and 3 = 5
7 + 8 = 15
```

Special Symbols



Reserved Words (Keywords)

- `int`
- `float`
- `double`
- `char`
- `void`
- `public`
- `static`
- `throws`
- `return`

Java Identifiers

- Names of things
- Consist of:
  - Letters
  - Digits
  - The underscore character (_)
  - The dollar sign ($)
- Must begin with a letter, underscore, or the dollar sign

Illegal Identifiers

TABLE 2-1 Examples of Illegal Identifiers

Illegal Identifier	Description
`employee Salary`	There can be no space between `employee` and `Salary`.
`Hello!`	The exclamation mark cannot be used in an identifier.
`one+two`	The symbol + cannot be used in an identifier.
`2nd`	An identifier cannot begin with a digit.

Data Types

- **Data type**: set of values together with a set of operations

Primitive Data Types

- Integral, which is a data type that deals with integers, or numbers without a decimal part (and characters)
- Floating-point, which is a data type that deals with decimal numbers
- Boolean, which is a data type that deals with logical values

Integral Data Types

- `char`
- `byte`
- `short`
- `int`
- `long`

Values and Memory Allocation for Integral Data Types

TABLE 2-2 Values and Memory Allocation for Integral Data Types

Data Type	Values	Storage (in bytes)
`char`	0 to 65535 (= $2^{16} - 1$)	2 (16 bits)
`byte`	-128 (= -2^7) to 127 (= $2^7 - 1$)	1 (8 bits)
`short`	-32768 (= -2^{15}) to 32767 (= $2^{15} - 1$)	2 (16 bits)
`int`	-2147483648 (= -2^{31}) to 2147483647 (= $2^{31} - 1$)	4 (32 bits)
`long`	-9223372036854775808 (= -2^{63}) to 9223372036854775807 (= $2^{63} - 1$)	8 (64 bits)

Primitive Data Types

- Floating-point data types
  - `float`: precision = 6 or 7
  - `double`: precision = 15
- `boolean`: two values
  - `true`
  - `false`

Literals (Constants)

- Integer literals, integer constants, or integers: 23 and -67
- Floating-point literals, floating-point constants, floating-point numbers: 12.34 and 25.60
- Character literals, character constants, or characters: 'a' and '5'

Arithmetic Operators and Operator Precedence

- Five arithmetic operators
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - % mod (modulus)
- Unary operator: operator that has one operand
- Binary operator: operator that has two operands

Order of Precedence

```
1.  *    /      %     (same precedence)
2.  +      -         (same precedence)
```

- Operators in 1 have a higher precedence than operators in 2
- When operators have the same level of precedence, operations are performed from left to right

Expressions

- Integral expressions
- Floating-point or decimal expressions
- Mixed expressions

Integral Expressions

- All operands are integers
- Examples

```
2 + 3 * 5
3 + x – y / 7
x + 2 * (y – z) + 18
```

Floating-Point Expressions

- All operands are floating-point numbers
- Examples

```
12.8 * 17.5 – 34.50
x * 10.5 + y - 16.2
```

Mixed Expressions

- Operands of different types
- Examples

```
2 + 3.5
6 / 4 + 3.9
```

- Integer operands yield an integer result; floating-point numbers yield floating-point results
- If both types of operands are present, the result is a floating-point number
- Precedence rules are followed

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
```

The `class` `String`

- Used to manipulate strings
- String
  - Sequence of zero or more characters
  - Enclosed in double quotation marks
  - Null or empty strings have no characters
  - Numeric strings consist of integers or decimal numbers
  - Length is the number of characters in a string

Strings and the Operator +

- Operator + can be used to concatenate two strings or a string and a numeric value or character
- Example 2-10

```
"The sum = " + 12 + 26
```

-After this statement executes, the string assigned to str is:

```
"The sum = 1226";
```

Strings and the Operator + (continued)

- Consider the following statement:

```
"The sum = " + (12 + 26)
```

- In this statement, because of the parentheses, you first evaluate `num1 + num2`
  - Because `num1` and `num2` are both `int` variables, `num1 + num2 = 12 + 26 = 38`
  - After this statement executes, the string assigned to `str` is:

```
"The sum = 38";
```

Input

- Named constant
  - Cannot be changed during program execution
  - Declared by using the reserved word `final`
  - Initialized when it is declared
- Example 2-11

```
final double CENTIMETERS_PER_INCH = 2.54;
final int NO_OF_STUDENTS = 20;
final char BLANK = ' ';
final double PAY_RATE = 15.75;
```

Input (continued)

- Variable (name, value, data type, size)
  - Content may change during program execution
  - Must be declared before it can be used
  - May not be automatically initialized
  - If new value is assigned, old one is destroyed
  - Value can only be changed by an assignment statement or an input (read) statement
- Example 2-12

```
double amountDue;
int    counter;
char   ch;
int    num1, num2;
```

Input (continued)

- The assignment statement

```
variable = expression;
```

- Example 2-13

```
int num1;
int num2;
double sale;
char first;
String str;

num1 = 4;
num2 = 4 * 5 - 11;
sale = 0.02 * 1000;
first = 'D';
str = "It is a sunny day.";
```

Input (continued)

- Example 2-14

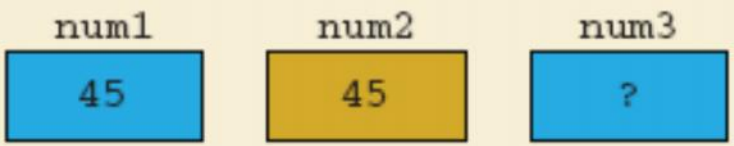
```
1. num1 = 18;
2. num1 = num1 + 27;
3. num2 = num1;
4. num3 = num2 / 5;
5. num3 = num3 / 4;
```

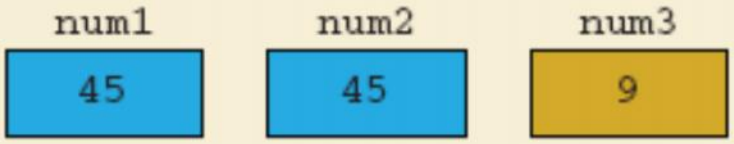
Input (continued)

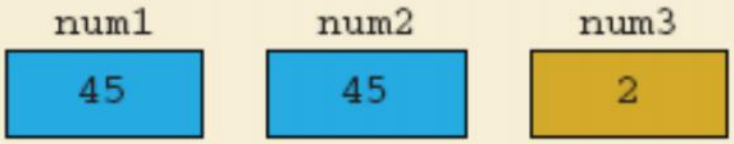
TABLE 2-4 Values of the Variables `num1`, `num2`, and `num3`

Values of the Variables	Variables	Statement/Explanation
Before Statement 1	num1: ? num2: ? num3: ?	
After Statement 1	num1: 18 num2: ? num3: ?	`num1 = 18;` Store `18` into `num1`.
After Statement 2	num1: 45 num2: ? num3: ?	`num1 = num1 + 27;` `num1 + 27 = 18 + 27 = 45.` This value is assigned to `num1`, which replaces the old value of `num1`.

Input (continued)

After Statement 3		`num2 = num1;` Copy the value of `num1` into `num2`.

After Statement 4		`num3 = num2 / 5;`
		`num2 / 5 = 45 / 5 = 9.`
		This value is assigned to `num3`. So `num3 = 9`.

After Statement 5		`num3 = num3 / 4;`
		`num3 / 4 = 9 / 4 = 2.`
		This value is assigned to `num3`, which replaces the old value of `num3`.

Input (continued)

- Standard input stream object: `System.in`
- Input numeric data to program
  - Separate by blanks, lines, or tabs
- To read data:
  1. Create an input stream object of the `class` `Scanner`
  2. Use the methods such as `next`, `nextLine`, `nextInt`, and `nextDouble`

Input (continued)

```
static Scanner console = new Scanner(System.in);
```

- Example 2-16

```
static Scanner console = new Scanner(System.in);
int feet;
int inches;

Suppose the input is

23 7

feet = console.nextInt();     //Line 1
inches = console.nextInt();   //Line 2
```

Increment and Decrement Operators

- `++` increments the value of its operand by 1
- `--` decrements the value of its operand by 1
- Syntax

  Pre-increment: `++variable`

  Post-increment: `variable++`

  Pre-decrement: `--variable`

  Post-decrement: `variable--`

Output

- Standard output object: `System.out`
- Methods
  `print`
  `println`
- Syntax

```
System.out.print(stringExp);
System.out.println(stringExp);
System.out.println();
```

Commonly Used Escape Sequences

TABLE 2-5 Commonly Used Escape Sequences

	Escape Sequence	Description
`\n`	Newline	Cursor moves to the beginning of the next line
`\t`	Tab	Cursor moves to the next tab stop
`\b`	Backspace	Cursor moves one space to the left
`\r`	Return	Cursor moves to the beginning of the current line (not the next line)
`\\`	Backslash	Backslash is printed
`\'`	Single quotation	Single quotation mark is printed
`\"`	Double quotation	Double quotation mark is printed

Packages, Classes, Methods, and the `import` Statement

- **Package**: collection of related classes
- **Class**: consists of methods
- **Method**: designed to accomplish a specific task

`import` Statement

- Used to import the components of a package into a program
- Reserved word
- `import java.io.*;`
  - Imports the (components of the) `package` `java.io` into the program
- Primitive data types and the `class` `String`
  - Part of the Java language
  - Don’t need to be imported

Creating a Java Application Program

- Syntax of a class

```
public  class ClassName
{
     classMembers
}
```

- Syntax of the `main` method

```
public static void main (String[] args)
{
     statement1
          .
          .
          .
     statementn
}
```

Debugging: Understanding and Fixing Syntax Errors

- When you type a program, typos and unintentional syntax errors are likely to occur
- Therefore, when you compile a program, the compiler will identify the syntax error

```
1.  import java.util.*;
2.
3.  public class ProgramNum1
4.  {
5.     static Scanner console = new Scanner(System.in);
6.
7.     public static void main(String[] args)
8.     {
9.        int num
10.
11.       num = 18;
12.
13.       tempNum = 2 * num;
14.
15.       System.out.println("Num = " + num + ", tempNum = " - tempNum);
16.    }
```

- The compiler produces the following errors:

```
ProgramNum.java:9: ';' expected
      int num
              ^
ProgramNum.java:16: reached end of file while parsing
   }
    ^
2 errors
```

```
1.  import java.util.*;
2.
3.  public class ProgramNum2
4.  {
5.      static Scanner console = new Scanner(System.in);
6.
7.      public static void main(String[] args)
8.      {
9.          int num;
10.
11.         num = 18;
12.
13.         tempNum = 2 * num;
14.
15.         System.out.println("Num = " + num + ", tempNum = " - tempNum);
16.     }
17. }
```

When you compile this program, it will generate the following errors:

```
ProgramNum2.java:13: cannot find symbol
symbol  : variable tempNum
location: class ProgramNum2
      tempNum = 2 * num;
      ^
ProgramNum2.java:15: cannot find symbol
symbol  : variable tempNum
location: class ProgramNum2
      System.out.println("Num = " + num + ", tempNum = " + tempNum);
                                                          ^
2 errors
```

```
1.  import java.util.*;
2.
3.   public class ProgramNum3
4.   {
5.       static Scanner console = new Scanner(System.in);
6.
7.       public static void main(String[ ] args)
8.       {
9.           int num;
10.          int tempNum;
11.
12.          num = 18;
13.
14.          tempNum = 2 * num;
15.
16.          System.out.println("Num = " + num + ", tempNum = " - tempNum);
17.     }
18.  }
```

When this program is compiled, it generates the following error:

```
ProgramNum3.java:16: operator - cannot be applied to java.lang.String, int
     System.out.println("Num = " + num + ", tempNum = " - tempNum);
                                                        ^
1 error
```

```
1.  import java.util.*;
2.
3.  public class ProgramNum4
4.  {
5.      static Scanner console = new Scanner(System.in);
6.
7.      public static void main(String[] args)
8.      {
9.          int num;
10.         int tempNum;
11.
12.         num = 18;
13.
14.         tempNum = 2 * num;
15.
16.         System.out.println("Num = " + num + ", tempNum = " + tempNum);
17.     }
18. }
```

Programming Style and Form

- Know common syntax errors and rules
- Use blanks appropriately
- Semicolon: statement terminator
- Important to have well-documented code
- Good practice to follow traditional rules for naming identifiers

Avoiding Bugs: Consistent, Proper Formatting and Code Walkthrough

- Java is a free-format language in the sense that programming instructions need not be typed in specific columns
- As you will discover, consistent and proper formatting will make it easier to develop, debug, and maintain programs
- Throughout the book, you will see consistent and predictable use of blanks, tabs, and newline characters to separate the elements of a program
- When you write programs, unintentional typos and errors are unavoidable
- The Java compiler will find the syntax rules and give some hints how to correct them; however, the compiler may not find logical (semantic) errors

Avoiding Bugs: Consistent, Proper Formatting and Code Walk-Through (continued)

- Typically, programmers try to find and fix these problems themselves by walking carefully through their programs
- Sometimes after multiple readings, a programmer may not be able to find the bug because the programmer may overlook the piece of the code that contains the bug and therefore may seek outside help
- If your program is properly formatted and you have used good names for identifiers, the person reading your program will have an easier time reading and debugging the program

Avoiding Bugs: Consistent, Proper Formatting and Code Walk-Through (continued)

- Before you seek outside help, you should be prepared to explain what your program intended to do and answer questions raised by the person reading your program
- The examination of your program by yourself, by another person, or a group of persons is a walk-through; a walk-through is helpful for all phases of the software development process

More on Assignment Statements

```
variable = variable * (expression);
```

is equivalent to

```
variable *= expression;
```

Similarly,

```
variable = variable + (expression);
```

is equivalent to

```
variable += expression;
```

Programming Examples

- Convert Length program
  - Input: length in feet and inches
  - Output: equivalent length in centimeters
- Make Change program
  - Input: change in cents
  - Output: equivalent change in half-dollars, quarters, dimes, nickels, and pennies

Chapter Summary

- Basic elements of a Java program include:
  - The main method
  - Reserved words
  - Special symbols
  - Identifiers
  - Data types
  - Expressions
  - Input
  - Output
  - Statements

Chapter Summary (continued)

- To create a Java application, it is important to understand:
  - Syntax rules
  - Semantic rules
  - How to manipulate strings and numbers
  - How to declare variables and named constants
  - How to receive input and display output
  - Good programming style and form