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

- 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

- 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.
Recall…

- **Computer program**: A sequence of statements (instructions) designed to accomplish a task.

- **Programming**: The process of planning and creating a program.
## Syntax and Semantic Rules

<table>
<thead>
<tr>
<th>How can we learn French!!</th>
<th>How can we learn Java!!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alphabet</td>
<td>Symbols</td>
</tr>
<tr>
<td>Words</td>
<td>Words</td>
</tr>
<tr>
<td>Grammar</td>
<td>Syntax</td>
</tr>
</tbody>
</table>
Syntax and Semantic Rules

- **Syntax rules** tell you which statements (instructions) are legal, or accepted by the programming language and which are not:
  - *A compiler will complain about programs with invalid syntax.*

- **Semantic rules** determine the meaning of the instruction:
  - *A compiler will complain about many (but not all) semantic errors in programs.*

Why can’t the compiler “catch” all errors?
- So a program may compile without errors
- But not run correctly - i.e., do the right thing
- Don’t become reliant on the compiler.
- **NOTE:** A syntactically valid program is not necessarily meaningful!

NOTE: A syntactically valid program is not necessarily meaningful!
Syntax and Semantic Examples

- **Some Java syntax rules:**
  - Statements must be terminated by a semicolon.
  - Parentheses, braces and brackets must balance.
    - $3 + 4 + 6$ is valid, but,
    - $3 + 4 +$ is invalid.

- **Some semantic rules:**
  - Subtraction is only meaningful on numbers so:
    - $3 - 5$ is valid, but
    - $3 - “five”$ is invalid.
The Basics of a Java Program

- Java program: A collection of classes.
- There is a main method in every Java application program.
- Token: The smallest individual unit of a program. It is either special symbols, word symbols, or identifiers.
Special Symbols

1. public class Message
2. {
3.   public static void main(String[] arg)
4.   {
5.     System.out.println("This is a message");
6.   }
7. }

*Note*: Blank is a special symbol.
Other Special Symbols

+, -, *, /, ., ;, ?, ==, >=
Word Symbols (reserved words)

- Also called reserved words or keywords.
- They are words that mean something special to Java.
- Cannot be redefined.
- Always lowercase.
- Complete list in Appendix A (second ed.).

```java
1. public class Message
2. {
3.   public static void main(String[] arg)
4.   {
5.     System.out.println("This is a message");
6.   }
7. }
```
# Java Reserved Words

<table>
<thead>
<tr>
<th>Java Keywords</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
<td>finally</td>
<td>public</td>
</tr>
<tr>
<td>boolean</td>
<td>float</td>
<td>return</td>
</tr>
<tr>
<td>break</td>
<td>for</td>
<td>short</td>
</tr>
<tr>
<td>byte</td>
<td>if</td>
<td>static</td>
</tr>
<tr>
<td>case</td>
<td>implements</td>
<td>super</td>
</tr>
<tr>
<td>catch</td>
<td>import</td>
<td>switch</td>
</tr>
<tr>
<td>char</td>
<td>instanceof</td>
<td>synchronized</td>
</tr>
<tr>
<td>class</td>
<td>int</td>
<td>this</td>
</tr>
<tr>
<td>continue</td>
<td>interface</td>
<td>throw</td>
</tr>
<tr>
<td>default</td>
<td>long</td>
<td>throws</td>
</tr>
<tr>
<td>do</td>
<td>native</td>
<td>transient</td>
</tr>
<tr>
<td>double</td>
<td>new</td>
<td>true</td>
</tr>
<tr>
<td>else</td>
<td>null</td>
<td>try</td>
</tr>
<tr>
<td>extends</td>
<td>package</td>
<td>void</td>
</tr>
<tr>
<td>false</td>
<td>private</td>
<td>volatile</td>
</tr>
<tr>
<td>final</td>
<td>protected</td>
<td>while</td>
</tr>
</tbody>
</table>

Java Programming: From Problem Analysis to Program Design, Third Edition
Java Identifiers

They are names that we introduce in our program
Some are predefined; others are defined by the user.
Consists of:
- Letters: (a → z) (A → Z)
- Digits (0 → 9)
- The underscore character (_)
- The dollar sign ($)
Must begin with a letter, underscore, or the dollar sign.

1. public class Message
2. {
3.   public static void main(String[] arg)
4.   {
5.     System.out.println("This is a message");
6.   }
7. }

Java Programming: From Problem Analysis to Program Design, Third Edition
Java Identifiers

- Java identifiers can be any length.
- Unlike reserved words, predefined identifiers can be redefined, but it would not be wise to do so.
- Some predefined identifiers:
  - `print`, `println`, `next`, `nextLine`
- Names should be descriptive:
  - `Message` – the name of a program that prints out a message.
  - `System.out.println` – the name for a part of Java that prints a line of output to the screen.
Illegal Identifiers

**Note:**

*White space*, breaks up the program into words, e.g. the two reserved words `static` and `void`, rather than `staticvoid`, which would be assumed to be an identifier!
Illegal Identifiers

Table 2-1  Examples of Illegal Identifiers

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>employee Salary</td>
<td>There can be no space between employee and Salary.</td>
</tr>
<tr>
<td>Hello!</td>
<td>The exclamation mark cannot be used in an identifier.</td>
</tr>
<tr>
<td>one+two</td>
<td>The symbol + cannot be used in an identifier.</td>
</tr>
<tr>
<td>2nd</td>
<td>An identifier cannot begin with a digit.</td>
</tr>
</tbody>
</table>

Note:

_White space_, breaks up the program into words, e.g. the two reserved words _static_ _void_, rather than _static void_, which would be assumed to be an identifier!
Data Types

- The objective of a Java program is to manipulate data.
- Different programs manipulate different data.
- A Data type is a set of values together with a set of operations.
- Only certain operations can be performed on a particular type of data.
Primitive Data Types
(fundamental DT)

Figure 2-1  Primitive Data Types

Integers
Ex: 1, 5, 10, -3

Decimal numbers
Ex: 1.5, -3.0, +6.1

Logical values
Ex: true or false
Primitive Data Types

Integral data types:

Figure 2-2  Integral Data Types
Primitive Data Types

**Integral data types:**

- deals with integers, or numbers without a decimal part.
- The type depends on how big the number is.

**int Data Type:**

- -6347, +90, 10, 0
- Positive integers do not have to have a + sign.
- No commas are used:
  
  36,782 → 2 integers 36 and 782
Encoding schemes

- The digit 0 or 1 is called binary digit or bit.
- Byte: sequence of 8 bits (11000110).
  - Byte = 8 bits
  - KiloByte (KB) = $2^{10}$ bytes.
  - MegaByte (MB) = $2^{20}$ bytes.
  - GigaByte (GB) = $2^{30}$ bytes.
  - TeraByte (TB) = $2^{40}$ bytes.
- Every letter, number or special symbol on the keyboard is encoded as a sequence of bits, each having a unique representation.
- ASCII(128), EBCDIC (256) and Unicode(65536) are different encoding schemes.
- ASCII is a subset of Unicode.
Primitive Data Types

 Integral data types:

char Data Type:
- is used to represent single characters. It can represent any key on your keyboard. Ex: ‘a’, ‘+’, ‘7’
- ‘abc’, ‘!=’ are NOT char value.
- Java uses the Unicode character set.
- Each character has a predefined order in the set ➔ collating sequence
- Value 13 = 14th character = ‘\n’ = new line
- Value 65 ➔ ‘A’
- Value 43 ➔ ‘+’
- Value 66 ➔ ‘B’
  - ‘A’ < ‘B’; ‘+’ < ‘A’
# Primitive Data Types

## Values and Memory Allocation for Integral Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Values</th>
<th>Storage (in bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>0 to 65535</td>
<td>2 (16 bits)</td>
</tr>
<tr>
<td>byte</td>
<td>-128 to 127</td>
<td>1 (8 bits)</td>
</tr>
<tr>
<td>short</td>
<td>-32768 to 32767</td>
<td>2 (16 bits)</td>
</tr>
<tr>
<td>int</td>
<td>-2147483648 to 2147483647</td>
<td>4 (32 bits)</td>
</tr>
<tr>
<td>long</td>
<td>-9223372036845477588808 to 922337203684547758807</td>
<td>8 (64 bits)</td>
</tr>
</tbody>
</table>
Primitive Data Types

❖ **Floating-point data types:**
  ❖ Represent real numbers in scientific notation.
  ❖ Ex: 75.924 ➞ 7.5924 * 10¹ ➞ 7.592400E1 in Java
  ❖ Has two types:
    ❖ **float:**
      ❖ Values: -3.4E+38 ➞ 3.4E+38
      ❖ Precision: 6 or 7 (4 bytes)
    ❖ **double:**
      ❖ Values: -1.7E+308 ➞ 1.7E+308
      ❖ Precision: 15 (8 bytes)
int or double?

When do we know to use int and when do we use double?

- If the data value you are going to use might be fractional then choose double.
- If it will always going to be a whole number choose int.

Consider the following cases. What would you choose?

- Counting how many people have used a computer during a day.
- The area of the lecture room in meters.
- Average age of the students in CSC112 :) .
Primitive Data Types

- **Boolean Data Types:**
  - Has Two values:
    - true
    - false
  - These are called the **logical (Boolean) values**.
  - The central purpose of this data type is to manipulate logical (Boolean) expressions.
  - Ex: ‘a’ != ‘A’ → true
Guidelines for selecting data types

- To represent integral values use type `int`. If a larger range of values is needed, use type `long`.
- On some computers, `long` will take longer to execute, so care may be needed if a lot of arithmetic is being performed.
- To represent non-integral values, use type `double`. Type `float` has similar properties but less precision and a smaller range.
- If speed of execution is very important, the `float` type may offer advantages on some computers.
- These are all what we call *primitive* types in Java. There are other types available. *You'll learn about these when later in this chapter*.
Arithmetic Operators and Operator Precedence

- Five arithmetic operators:
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - % mod (modulus)
- / with integral data types ➞ integer results.
- **Unary operator**: An operator that has one operand. Ex: -5
- **Binary operator**: An operator that has two operands. Ex: 7 - 5
- +, - can be unary or binary; *, /, % are always binary.
## Example

<table>
<thead>
<tr>
<th>Arithmetic expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 / 2</td>
<td>2</td>
</tr>
<tr>
<td>5.0 / 2.0</td>
<td>2.5</td>
</tr>
<tr>
<td>14 / 7</td>
<td>2</td>
</tr>
<tr>
<td>34 % 5</td>
<td>4</td>
</tr>
<tr>
<td>- 34 % 5</td>
<td>-4</td>
</tr>
<tr>
<td>34 % -5</td>
<td>4</td>
</tr>
<tr>
<td>-34 % -5</td>
<td>-4</td>
</tr>
<tr>
<td>4 % 6</td>
<td>4</td>
</tr>
</tbody>
</table>
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 (i.e. associativity of arithmetic operators from left to right)
To avoid confusion use parentheses () to group arithmetic expressions.

Ex: 3 + 4 * 5 ➔ (3 +4) * 5 ➔ 35
Character Arithmetic

- `char` data type is an integer type
- Hence integer arithmetic is allowed on `char` data
- The integer value is the Unicode collating sequence.
  - $8 + 7 = 15$
  - ‘8’ + ‘7’ = 56 + 55 = 111 !!!
- If you must use arithmetic operations on the `char` type, do so WITH caution.
Expressions

1. Integral expressions
2. Floating-point or decimal expressions
3. Mixed expressions
1. Integral Expressions

- All operands are integers.
- Examples:
  
  \[ 2 + 3 \times 5 \]
  
  \[ 3 + x - y \div 7 \]
  
  \[ x + 2 \times (y - z) + 18 \]
2. Floating-Point Expressions

♦ All operands are floating-point numbers.

♦ Examples:

\[
12.8 \times 17.5 - 34.50
\]

\[
x \times 10.5 + y - 16.2
\]
3. Mixed Expressions

- Operands of different types.
- 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.
- Examples:
  \[
  2 + 3.5 \rightarrow 5.5 \\
  4 + 5/2.0 \rightarrow 4+ 2.5 \rightarrow 6.5 \\
  3 / 2 + 5.0 \rightarrow 1+ 5.0 \rightarrow 6.0 \\
  4*3+7/5-25.5 \rightarrow 12 + 7/5 - 25.5 \rightarrow 12 +1 -25.5 \rightarrow 13 -25.5 \rightarrow -12.5 \\
  \]
- Integer is not converted to fp number unless there is one fp operand.
Type Conversion (Casting)

- Used:
  - to change one data type to another.
  - to avoid implicit type coercion as $(1 + '8' = 57)$

- By the use of cast operator.

- Syntax:
  
  $$(\text{dataType}Name) \text{ expression}$$

- Expression evaluated first, then the value is converted to dataTypeName
Type Conversion (Casting)

Examples:
1. \((\text{int})(7.9) + (\text{int})(6.7) = 7 + 6 = 13\)
2. \((\text{int})(7.9 + 6.7) = (\text{int}) 14.6 = 14\)
3. \((\text{double})(17) = 17.0\)
4. \((\text{double})(8+3) =\)
5. \((\text{double})(7) / 2 =\)
6. \((\text{double})(7/2) =\)
7. \((\text{int})(7.8 + (\text{double})(15)/2) =\)
Examples:

1. \((\text{int})(7.9) + (\text{int})(6.7) = 7+6= 13\)
2. \((\text{int})(7.9 + 6.7) = (\text{int})14.6 = 14\)
3. \((\text{double})(17) = 17.0\)
4. \((\text{double})(8+3) = (\text{double})11 = 11.0\)
5. \((\text{double})(7) / 2 = 7.0/2 = 3.5\)
6. \((\text{double})(7/2) = 3.0\)
7. \((\text{int})(7.8+(\text{double})(15)/2) = \)
   \((\text{int})15.3 = 15\)
8. \((\text{int})(7.8 + (\text{double})(15/2)) = \)

9. \(x = 15\ ,\ y = 23\ ,\ z = 3.75\)
   
   \((\text{double})\ (y/x) + z = \)
   
   \((\text{double})\ (y) /x + z = \)

10. \((\text{int})\ (\text{\textquote{A}}) = \)

11. \((\text{int})\ (\text{\textquote{8}}) = \)

12. \((\text{char})\ (65) = \)

13. \((\text{char})\ (56) = \)
Type Conversion (Casting)

8. \((\text{int})(7.8+(\text{double})(15/2))=\text{(int)}14.8 = 14\)

9. \(x=15\), \(y = 23\), \(z = 3.75\)
   \(\text{(double)} \ (y/x) + z = \text{(double)}(1) + 3.75 = 4.75\)
   \(\text{(double)} \ (y) /x + z = 1.5333 + 3.75 = 5.28333\)

10. \((\text{int})(‘A’)) = 65\)
11. \((\text{int})(‘8’)) = 56\)
12. \((\text{char}) \ (65) = ‘A’\)
13. \((\text{char}) \ (56) = ‘8’\)
The class String

- We've seen almost all of the primitive types in use.
- Java also defines a lot of types in addition to the primitive types.
- Let's say you want a value to which is more than one character. In English we'd call this a string. But there is NO string primitive type!!
- In Java, there is a class called String. It provides a lot of methods that allow you to manipulate sequences of characters.
- A type that comes from a class always starts with a capital letter (String).
- Have you noticed that all primitive type names start with lower case letters? (int, short, long, double, float, byte, char...)

Java Programming: From Problem Analysis to Program Design, Third Edition
The class String

- Contains operations to manipulate strings.
- String:
  - Sequence of zero or more characters.
  - Enclosed in double quotation marks "".
  - Is processed as a single unit.
  - Null or empty strings have no characters. ""
  - Every character in a string has a relative position in that string, the first character is in position 0.
The class String

- **Length** of the string is the number of characters in it.
- Numeric strings consist of integers or decimal numbers.
- When determining the length of a string, **blanks count**.
- **Example**:
  - "" ➔ Empty String has length = 0
  - "abc" ➔ has length = 3, position of a = 0, b = 1, c = 2
  - "a boy" ➔ has length = 5
The class String

More examples:

- String: “William Jacob”
- Position of ‘W’:
- Position of second ‘i’:
- Position of ‘ ‘:
- Length of the Sting:
Input

Recall, data must be loaded into main memory before it can be manipulated.

Allocating Memory

- What names to use for each memory location
- What type of data to store in those memory locations
- Whether the data must remain fixed or should be changed throughout the program execution.

Memory can be allocated to store **constants** and **variables**.

Named constant

- A memory location whose content cannot be changed during program execution.
- Declared by using the reserved word **final**.
- Initialized when it is declared.
The syntax to declare a named constant:

```java
static final datatype IDENTIFIER = value;
```

*static* here *may or may not appear, later we will see when it might be required.*

**Example 2-11**

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

- The default type of floating point numbers is `double`.
- The declaration: `final float rate = 15.5f;`
  without the `f`, the compiler will generate an error.
Input

Why using constants?

♦ If the fixed data changes, no need to edit the entire program.
♦ Avoid typing the same value again and again.
Input

Variable (name, value, data type, size)

- A memory location whose content may change during program execution.
- Must be declared before it can be used.
- Java programmers typically use lowercase letters to declare variables.
- If new value is assigned, old one is destroyed.
- Syntax:

  ```java
  dataType identifier1, identifier2,..., identifierN;
  ```

Example 2-12

```java
double amountDue;
int counter;
char ch;
int x, y;
```
Putting data into Variables:

Two common ways to place data into a variable are:

1. an assignment statement (=)
2. an input (read) statement.
The Assignment Statement

- Syntax: `variable = expression;`
- Value of expression **should match** the data type of the variable.
- Expression on right is evaluated, value is assigned to variable on the left.
- **Java is strongly typed;** you cannot assign a value to a variable that is not **compatible** with its data type.
- Associativity of assignment operator is from right to left. **Example:**
  
  ```
  x = y = z;
  ```
Input

Example 2-13

```java
int i, j;
double sale;
char first;
String str;

Assignment Statements:

i = 4;
j = 4 * 5 - 11;
sale = 0.02 * 1000;
first = 'D';
str = "It is a sunny day.";
```
public class Example2_13
{
    public static void main (String[] args)
    {
        int i, j;
        double sale;
        char first;
        String str;

        i = 4;
        System.out.println("i= " + i);

        j = 4 * 5 - 11;
        System.out.println("j= " + j);

        sale = 0.02 * 1000;
        System.out.println("sale= " + sale);

        first = 'D';
        System.out.println("first= " + first);

        str = "It is a sunny day.";
        System.out.println("str= " + str);
    }
}
Declaring and initializing variables

- A variable is said to be *initialized* the first time a value is placed in that variable.
- May not be automatically initialized.
- Using a variable without initializing it, might produce errors.
- Java allows initializing with declaring.

Example 1 - declare then initialize:

```java
int first, second;
char ch;

first = 13;
second = 10;
ch = ' ';```

Example 2 - declare and initialize:

```java
int first = 13, second = 10;
char ch = ' ';```
Input

Input (read) statement

- To read data into variables (interactively):
  1. Create an input stream object of the class `Scanner`.
  2. Associate it with the standard input device. The following statement accomplishes this:

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

- `System.in` = is an object that provides methods to allow you to get input from the user into a program.

- `Scanner` is a predefined Java class *(only from JDK version 5.0 & higher)* and `console` is the created input stream object from that class.
Input

The object console reads input using the following methods

A. `console.nextInt()`: to read integer.
B. `console.nextDouble()`: to read floating-point numbers. (double & float)
C. `console.next()`: to read a string.
D. `console.nextLine()`: to read a string until the end of the line.

Note:
`nextInt()`, `nextDouble`, `next()` skip any whitespace characters (such as blank, newline and tab).
1. import java.util.*;

2. public class Example2_16
3. {
4.   static Scanner console = new Scanner(System.in);
5.   public static void main(String[] args)
6.   {
7.     int feet;
8.     int inches;
9.     System.out.println("Enter two integers separated by spaces.");
10.    feet = console.nextInt(); // reads int
11.    inches = console.nextInt(); // reads int
12.    System.out.println("Feet = " + feet);
13.    System.out.println("Inches = " + inches);
14.   }
15.}
Example 2-16

1. import java.util.*;

2. public class Example2_16
3. {
4.     static Scanner console = new Scanner(System.in);
5.     public static void main(String[] args)
6.     {
7.         int feet;
8.         int inches;
9.         System.out.println("Enter two integers separated by spaces.");
10.        feet = console.nextInt(); // reads int
11.        inches = console.nextInt(); // reads int
12.        System.out.println("Feet = " + feet);
13.        System.out.println("Inches = " + inches);
14.    }
15.}
Example 2-16

1. import java.util.*;

2. public class Example2_16
3. {
4.     static Scanner console = new Scanner(System.in);
5.     public static void main(String[] args)
6.     {
7.         int feet;
8.         int inches;
9.         System.out.println("Enter two integers separated by spaces.");
10.        feet = console.nextInt(); // reads int
11.        inches = console.nextInt(); // reads int
12.        System.out.println("Feet = " + feet);
13.        System.out.println("Inches = " + inches);
14.     }
15. }

// single line comment
/* multi
line
comment */
⇒ all ignored by the complier
Input

Example 2-16 - Run

Enter two integers separated by spaces.
> 23 7
Feet = 23
Inches = 7
Input

Example 2-16 - Run

```java
Enter two integers separated by spaces.
> 23 7
Feet = 23
Inches = 7
```

If the user enters a non integer number for example 24w5 or 3.4 ➔
`console.nextInt()` will cause a program termination.
Example 2-17

1. import java.util.*;
2. public class Example2_17
3. {
4.     static Scanner console = new Scanner(System.in);
5.     public static void main(String[] args)
6.     {
7.         String firstName;
8.         String lastName;
9.         int age;
10.        double weight;
11.        System.out.println("Enter first name, last name, "
12.            +"age, and weight separated by spaces.");
13.        firstName = console.next();
14.        lastName = console.next();
15.        age = console.nextInt();
16.        weight = console.nextDouble();
17.        System.out.println("Name: " + firstName + " " + lastName);
18.        System.out.println("Age: " + age);
19.        System.out.println("Weight: " + weight);
20.    }
21.}
Input

Example 2-17 - Run

Enter first name, last name, age, and weight separated by spaces.
> Sheila Mann 23 120.5
Name: Sheila Mann
Age: 23
Weight: 120.5
Variable Initialization

- When a variable is declared, Java might not automatically put a meaningful value into it.
- If you declare a variable and then use it in an expression without first initializing it, when you compile the program you are likely to get an error. Therefore Java allows you to initialize variables while they are being declared.
- Consider the following declaration:

```java
int feet;
```

You can initialize the variable feet to a value of 35 either by using the assignment statement:

```java
feet = 35;
```

or by executing the following statement and entering 35 during program execution:

```java
feet = console.nextInt();
```
Input

♦ Reading a Single Character
if \( ch \) is a \texttt{char} variable. To input A into \( ch \), you can use the following statement:

\[
ch = \text{console}.next().\text{charAt}(0);
\]
Example2_18

```java
import java.util.*;
public class Example2_18{
    static Scanner console = new Scanner(System.in);
    public static void main(String[] args) {
        int firstNum, secondNum;
        char ch;   double z;
        firstNum = 4;
        System.out.println("Line 2: firstNum = "+ firstNum);
        System.out.println("firstNum = "+ firstNum + " , secondNum = "+ secondNum);
        z = (firstNum + 1) / 2.0;
        ch = 'A';
        System.out.println("Line 6: firstNum = " + firstNum + " , secondNum = " + secondNum + " , z = " + z);
        secondNum = console.nextInt();
        System.out.println("Line 10: firstNum = " + firstNum + " , secondNum = " + secondNum + " , ch = " + ch + " , z = " + z);
        z = console.nextDouble();
    }
}
```

System.out.println("Line 8: firstNum = " + firstNum + " , secondNum = " + secondNum + " , ch = " + ch + " , z = " + z);
secondNum = console.nextInt();
System.out.println("Line 10: firstNum = " + firstNum + " , secondNum = " + secondNum + " , ch = " + ch + " , z = " + z);
```
System.out.println("Line 12: firstNum = " + firstNum + ", secondNum = " + secondNum + ", ch = " + ch + ", z = " + z);

firstNum = 2 * secondNum + (int)(z);
System.out.println("Line 14: firstNum = " + firstNum + ", secondNum = " + secondNum + ", ch = " + ch + ", z = " + z);

secondNum = secondNum + 1;
System.out.println("Line 16: firstNum = " + firstNum + ", secondNum = " + secondNum + ", ch = " + ch + ", z = " + z);

ch = console.next().charAt(0);
System.out.println("Line 18: firstNum = " + firstNum + ", secondNum = " + secondNum + ", ch = " + ch + ", z = " + z);

firstNum = firstNum + (int)(ch);
// ‘D’ = 68
System.out.println("Line 20: firstNum = " + firstNum + ", secondNum = " + secondNum + ", ch = " + ch + ", z = " + z);

z = firstNum - z;
System.out.println("Line 22: firstNum = " + firstNum + ", secondNum = " + secondNum + ", ch = " + ch + ", z = " + z);
Input

Example2_18

Suppose the input is 8 16.3 D what should be stored in firstNum, secondNum, ch and z after the program executes?
Input

Example2_18

Suppose the input is 8 16.3 D what should be stored in `firstNum`, `secondNum`, `ch` and `z` after the program executes?

<table>
<thead>
<tr>
<th>firstNum</th>
<th>secondNum</th>
<th>ch</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>9</td>
<td>D</td>
<td>83.7</td>
</tr>
</tbody>
</table>
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--`
Increment and Decrement Operators

- **Example:**
  ```java
  int count = 1;
  count ++ ; or  ++ count ; // same as count = count +1
  ```

- The meaning of pre and post differ when the variable using these operators is used in an expression.
  - The **pre**-increment adds 1 to the variable **before** the expression is evaluated. Similarly, the **pre**-decrement subtracts 1 from the variable before it is evaluated in an expression while.
  - The **post**-increment adds 1 to the variable **after** the expression is evaluated. Similarly, post-decrement subtracts the value 1 from the variable after the expression is evaluated.
Increment and Decrement Operators

**Example:**

```java
int x, y;
1. x = 5;
   y = ++x;  //the value of x is incremented
             //first then it is assigned to y.
             //( x =6 , y =6 )

2. x = 5;
   y = x++ ;  //the current value of x (5) is used
              //to evaluate the exp. then the
              //value of x is incremented.,
              //( x =6 , y =5)
```
Increment and Decrement Operators

Example:

```java
int a, b;
3. a = 5;
   b = 2 + (++a); // a = 6, b = 8
4. a = 5;
   b = 2 + (a++); // a = 5 during the exp.
   // Evaluation then its
   // incremented to 6  b = 7
```
Strings and the Operator +

- Operator + can be used to concatenate (join) two strings, or a string and a numeric value or character.

Example 2-20(a)

```java
String str;
int num1, num2;
num1 = 12;
num2 = 26;
str = "The sum = " + num1 + num2;
```

After this statement executes, the string assigned to `str` is:
"The sum = 1226";
Example 2-20(b)
Consider the following statement:

```
str = "The sum = " + (num1 + num2);
```

- 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";
Example

Consider the following statement:

```java
str = num1 + num2 + " is the sum ";
```

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

"38 is the sum";
Output

- Standard output object is System.out.

- Methods:
  - print: leaves insertion point after last char in the line.
  - println: moves insertion point to beginning of next line.

- Syntax:
  System.out.print(stringExp);
  System.out.println(stringExp);
  System.out.println();
### Statement vs. Output

<table>
<thead>
<tr>
<th>Statement</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.out.println('A');</td>
<td>A</td>
</tr>
<tr>
<td>System.out.println(&quot;Hello \n there.&quot;);</td>
<td>Hello there.</td>
</tr>
<tr>
<td>System.out.print(&quot;Hello&quot;); System.out.println(&quot; there.&quot;);</td>
<td>Hello there.</td>
</tr>
</tbody>
</table>
Commonly Used Escape Sequences

In Java, \ is called **escape character**.

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>Newline</td>
</tr>
<tr>
<td>\t</td>
<td>Tab</td>
</tr>
<tr>
<td>\b</td>
<td>Backspace</td>
</tr>
<tr>
<td>\r</td>
<td>Return</td>
</tr>
<tr>
<td>\</td>
<td>Backslash</td>
</tr>
<tr>
<td>'</td>
<td>Single quotation</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quotation</td>
</tr>
</tbody>
</table>

**Example:**

```java
System.out.println(" The tab character is represented as \'\t\'");
```

The tab character is represented as `\t`
Output

How to fit the following statement in one line as part of the output statement?

It is sunny, warm, and not a windy day. Let us go golfing.

Check Example 2-24 (text book 2nd Ed.)
Packages, Classes, Methods, and the import Statement

- **Package**: A collection of related classes.
- **Class**: Consists of methods.
- **Method**: Designed to accomplish a specific task.
- **Example**:
  - **Method**: `pow`
  - **Class**: `Math`
  - **Package**: `java.lang`
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:

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

- Syntax of the `main` method:

```java
public static void main (String[] args) {
    statement1
    
    statementn
}
```
Creating a Java Application Program

1. **import** statements if any
2. **public class** ClassName
3. {
4. declare CONSTANTS and/or stream objects
5. **public static void** main(String[] args)
6. {
7. variable declaration
8. executable statements
9. }
10.}
Programming Style and Form

- Know common syntax errors and rules.
- Use blanks appropriately.
- Use a semicolon as a statement terminator.
- Important to have well-documented code.
- Good practice to follow traditional rules for naming identifiers.
- Use prompt lines to inform the user what to do.
- Add multi-line comment at the top of a program to briefly explain the program and to give information about the programmer.
- Take a look at example2-29.
More on Assignment Statements

- Simple assignment statements: \( x = x \times y; \)
- Compound assignments: \( x *= y; \)
- \( +=, -=, *=, /=, %= \)

Syntax:

\[
\text{variable} = \text{variable} \times (\text{expression});
\]

is equivalent to:

\[
\text{variable} *= \text{expression};
\]

Similarly,

\[
\text{variable} = \text{variable} + (\text{expression});
\]

is equivalent to:

\[
\text{variable} += \text{expression};
\]
### Example 2-30

<table>
<thead>
<tr>
<th>Simple assignment</th>
<th>Compound assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i = i + 5; )</td>
<td>( i += 5; )</td>
</tr>
<tr>
<td>( \text{sum} = \text{sum} + \text{number} )</td>
<td>( \text{sum} += \text{number}; )</td>
</tr>
<tr>
<td>( x = x / (y + 5); )</td>
<td>( x /= y + 5; )</td>
</tr>
</tbody>
</table>
Programming Examples

- Convert Length program: *(Conversion.java)*
  - Input: Length in feet and inches.
  - Output: Equivalent length in centimeters.

- Make Change program: *(MakeChange.java)*
  - 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

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.