Chapter 2: Introduction t Java Applications

TASNIM SHARMIN ALIN
LECTURER,
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING,
LEADING UNIVERSITY

line

```
1  // Fig. 2.1: Welcome1.java
2  // Text-printing program.
3
4  public class Welcome1 {
5
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.println( "Welcome to Java Programming!" );
10
11    } // end method main
12
13 } // end class Welcome1
```

```
Welcome to Java Programming!
```

2.2 A First Program in Java: Printing a Line of Text

```
1  // Fig. 2.1: Welcome1.java
```

- Comments start with: //
  - Indicating that the remainder of the line is a comment
  - Comments ignored during program execution
  - A comment that begins with // is called an ***end-of-line*** comment
- Traditional comments: /* ... */

```
/* This is a traditional
   comment. It can be
   split over many lines */
```

```
2 // Text-printing program
```

- Line 2 is an end-of-line comment that describe the purpose of the program

2.2 A Simple Program: Printing a Line of Text

```
3
```

- Blank line
  - Blank lines, spaces, and tabs are white-space characters
    - Ignored by compiler
  - Space characters and tabs are known specifically as hite-space character.

```
4  public class Welcome1 {
```

- Begins class declaration for class `Welcome1`
  - Every Java program has at least one user-defined class
  - Keyword: words reserved for use by Java
    - `class` keyword followed by class name
  - Naming classes: capitalize every word
    - `SampleClassName`

2.2 A Simple Program: Printing a Line of Text

```
4   public class welcome1 {
```

- Name of class called identifier
  - Series of characters consisting of letters, digits, underscores (_) and dollar signs ($)
  - Does not begin with a digit, has no spaces
  - Examples: `welcome1`, `$value`, `_value`, `button7`
    - **`7button`** is invalid
  - Java is case sensitive (capitalization matters)
    - `a1` and `A1` are different

2.2 A Simple Program: Printing a Line of Text

```
4   public class Welcome1 {
```

- Saving files
  - File name must be class name with `.java` extension
  - `Welcome1.java`
- Left brace `{`
  - Begins body of every class
  - Right brace ends declarations (line 13)

```
7     public static void main( String args[] )
```

- Part of every Java application
  - Applications begin executing at `main`
    - Parenthesis indicate `main` is a method (ch. 6)
    - Java applications contain one or more methods

2.2 A Simple Program: Printing a Line of Text

```
7      public static void main( String args[] )
```

- Exactly one method must be called `main`

– Methods can perform tasks and return information

```
8      {
```

– Left brace begins body of method declaration
  - Ended by right brace } (line 11)

2.2 A Simple Program: Printing a Line of Text

```
9         System.out.println( "Welcome to Java Programming!" );
```

- Instructs computer to perform an action
  - Prints string of characters
    - String - series characters inside double quotes
  - White-spaces in strings are not ignored by compiler
- `System.out`
  - Standard output object
  - Print to command window (i.e., MS-DOS prompt)
- Method `System.out.println`
  - Displays line of text
  - Argument inside parenthesis
- This line known as a statement
  - Statements must end with semicolon `;`

2.2 A Simple Program: Printing a Line of Text

```
11     } // end method main
```

- Ends method declaration

```
13  } // end class Welcome1
```

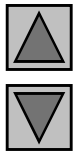
- Ends class declaration
- Can add comments to keep track of ending braces
- Lines 8 and 9 could be rewritten as:
- Remember, compiler ignores comments
- Comments can start on same line after code

2.3 Modifying Our First Java Program

- Modifying programs
  - Welcome2.java (Fig. 2.3) produces same output as Welcome1.java (Fig. 2.1)
  - Using different code

```
9         System.out.print( "Welcome to " );
10        System.out.println( "Java Programming!" );
```

  - Line 9 displays “Welcome to ” with cursor remaining on printed line
  - Line 10 displays “Java Programming! ” on same line with cursor on next line



Outline

```
1  // Fig. 2.3: Welcome2.java
2   // Printing a line of text with multiple
  statements.
3
4   public class Welcome2 {
5
6      // main method begins execution of Java
  application
7      public static void main( String args[] )
8      {
9         System.out.print( "Welcome to " );
10        System.out.println( "Java Programming!" );
11
12     } // end method main
13
14  } // end class Welcome2
```

`System.out.print` keeps the cursor on the same line, so `System.out.println` continues on the same line.

```
Welcome to Java Programming!
```

`Welcome2.java`

1. Comments

2. Blank line

3. Begin class `Welcome2`

3.1 Method `main`

4. Method `System.out.print`

thod `m.out.println`

5. end `main`, `Welcome2`

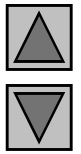
Program Output

2.3 Modifying Our First Java Program

- Newline characters (`\n`)
  - Interpreted as "special characters" by methods `System.out.print` and `System.out.println`
  - Indicates cursor should be on next line
  - `Welcome3.java` (Fig. 2.4)

```
9         System.out.println( "Welcome\nto\nJava\nProgramming!" );
```

  - Line breaks at \n
- Usage
  - Can use in `System.out.println` or `System.out.print` to create new lines
    - `System.out.println( "Welcome\nto\nJava\nProgramming!" );`



Outline

```
1  // Fig. 2.4: Welcome3.java
2  // Printing multiple lines of text with a single statement.
3
4  public class Welcome3 {
5
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.println( "Welcome\nto\nJava\nProgramming!"
);
10
11    } // end method main
12
13 } // end class Welcome3
```

```
Welcome
to
Java
Programming!
```

Notice how a new line is output for each \n escape sequence.

Welcome3.java

1. main

2. System.out.println (uses \n for new line)

Program Output

Displaying Text with printf

```
1   // Fig. 2.4: Welcome3.java
2   // Printing multiple lines of text with a single
    statement.
3
4   public class Welcome3 {
5
6      // main method begins execution of Java application
7      public static void main( String args[] )
8      {
9         System.out.printf("%s\n%s\n", "Welcome to","Java
    programming");
10
11     } // end method main
12
13  } // end class Welcome3
```

- ✓ System.out.printf method for displaying formatted data—the f in the name printf stands for "formatted".
- ✓ This method call specifies three arguments.
- ✓ When a method requires multiple arguments, the arguments are separated with commas(,)—this is known as a ***comma-separated list***.

2.3 Modifying Our First Java Program

Escape characters

- Backslash (\ )
- Indicates special characters be output

Escape sequence	Description
\n	Newline. Position the screen cursor at the beginning of the next line.
\t	Horizontal tab. Move the screen cursor to the next tab stop.
\r	Carriage return. Position the screen cursor at the beginning of the current line; do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
\\	Backslash. Used to print a backslash character.

\"	Double quote. Used to print a double -quote character. For example,
	`System.out.println( "\"in quotes \"" );`
	displays
	`"in quotes"`

Fig. 2.5 Some common escape sequences.

2.7 Arithmetic

- Arithmetic calculations used in most programs
- Arithmetic operators are binary operator because they each operate on two operands.
  - Usage
    - `*` for multiplication
    - `/` for division
    - `+`, `-`
    - No operator for exponentiation (more in Chapter 5)
  - Integer division truncates remainder

    `7 / 5` evaluates to `1`
  - Remainder operator `%` returns the remainder

    `7 % 5` evaluates to `2`

2.7 Arithmetic

- Operator precedence
  - Some arithmetic operators act before others (i.e., multiplication before addition)
    - Use parenthesis when needed
  - Example: Find the average of three variables `a`, `b` and `c`
    - Do not use: `a + b + c / 3`
    - Use: `( a + b + c ) / 3`
  - Follows **PEMDAS**
    - Parentheses, Exponents, Multiplication, Division, Addition, Subtraction

2.7 Arithmetic

Operator(s)	Operation(s)	Order of evaluation (precedence)
* /	Multiplication Division Remainder	Evaluated first. If there are several of this type of operator, they are evaluated from left to right.
+ -	Addition Subtraction	Evaluated next. If there are several of this type of operator, they are evaluated from left to right.

Fig. 2.17 Precedence of arithmetic operators.

2.8 Decision Making: Equality and Relational Operators

- `if` control statement
  - If a condition is true, then the body of the `if` statement executed
    - `0` interpreted as false, non-zero is true
  - Control always resumes after the `if` structure
  - Conditions for `if` statements can be formed using equality or relational operators (next slide)

```
if ( condition )
    statement executed if condition true
```

    - No semicolon needed after condition
      - Else conditional task not performed

2.8 Decision Making: Equality and Relational Operators

Standard algebraic equality or relational operator	Java equality or relational operator	Example of Java condition	Meaning of Java condition
Equality operators			
=	==	x == y	x is equal to y
	!=	x != y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 2.19 Equality and relational operators.

- Upcoming program uses `if` statements
  - Discussion afterwards

```
1       // Fig. 2.20: Comparison.java
2       // Compare integers using if statements, relational operators
3       // and equality operators.
4
5       // Java packages
6       import java.util.Scanner;
7
8       public class Comparison {
9
10        // main method begins execution of Java application
11        public static void main( String args[] )
12        {
13            // create Scanner to obtain input from command window
12.           Scanner input = new Scanner(System.in);

17            int number1;          // first number to compare
18            int number2;          // second number to compare
19
20            // read first number from user as a string
21            System.out.print("Enter first number: "); // prompt
22            number1=input.nextInt(); // read first number from user

             System.out.print("Enter second number: "); // prompt
22           number2=input.nextInt(); // read second number from user
```

```
34          if ( number1 == number2 )
35             System.out.printf("%d == %d \n" , number1 , number2);
36
37          if ( number1 != number2 )
38             System.out.printf(" %d ! = %d\n" , number1 ,number2);
39
40          if ( number1 < number2 )
41             System.out.printf (" %d < %d \n",number1, number2);
42
43          if ( number1 > number2 )
44             System.out.printf(" %d > %d\n", number1 ,number2);
45
46          if ( number1 <= number2 )
47             System.out.printf(" %d <= %d\n",number1 , number2);
48
49          if ( number1 >= number2 )
50             System.out.printf("%d >= %d", number1 , number2);
51
52
58       } // end method main
59
60    } // end class Comparison
```

2.8 Decision Making: Equality and Relational Operators

- Precedence of operators
  - All operators except for = (assignment) associates from left to right
    - For example: `x = y = z` is evaluated `x = (y = z)`

Operators	Associativity	Type
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

Fig. 2.21 Precedence and associativity of the operators discussed so far.

- Line 6

  import java.util.Scanner; ---- is an import declaration that helps the compiler locate a class that is used in this program. Programmer use import declaration to identify the predefined classes used in a java program.

- The import declaration in this line indicates that this examples uses java's predefined Scanner class from package ***java.util.***
- A Scanner enables a program to read data for use in a program. The data can come from many sources such as a file on a disk or the user at the keyboard.

  Scanner input = new Scanner (System.in);

- This line is a variable declaration statement (also called declaration) that specifies the name (input) and type (Scanner) of a variable that is used in this program.

- In line 12 the equal (=) sign indicates that Scanner variable input should b initialized in its declaration with the result of the expression new Scanner(System.in) to the right of the equal sign.
- This expression creates a Scanner object that reads data typed by the user at the keyboard.

Code Description

- The standard output object,System.out,allows java application to display characters in the command window.
- The standard input object, System.in,enables java application to read information typed by the user.
- Forgetting to include an import declaration for a class used in your program typically results a compilation error containing a message such as “cannot resolve symbol”

- Number1 = input.nextInt();

- This line uses Scanner object input’s nextInt method to obtain an integer from the user at the keyboard.At this point the program waits for the user to type the number and press the Enter key to submit the number to the program.
- This statement is read as “number1 gets the value of input.nextInt()”.
- Operator = is called a binary operator because it has two operands “number1 and number2”

System.out.print(“Enter first number”); // prompt

This message is called a prompt because it directs the user to take a specific action.