1

Introduction to Java Applications

Muhammed S. Anwar

OBJECTIVES

In this chapter you will learn:

- To write simple Java applications.
- To use input and output statements.
- Java’s primitive types.
- Basic memory concepts.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write decision-making statements.
- To use relational and equality operators.

Outline

2.1 Introduction

- **Java application programming**
  - **Display messages**
  - **Obtain information from the user**
  - **Arithmetic calculations**
  - **Decision-making fundamentals**

2.2 First Program in Java: Printing a Line of Text

- **Application**
  - **Executes when you use the `java` command to launch the Java Virtual Machine (JVM)**
- **Sample program**
  - **Displays a line of text**
  - **Illustrates several important Java language features**

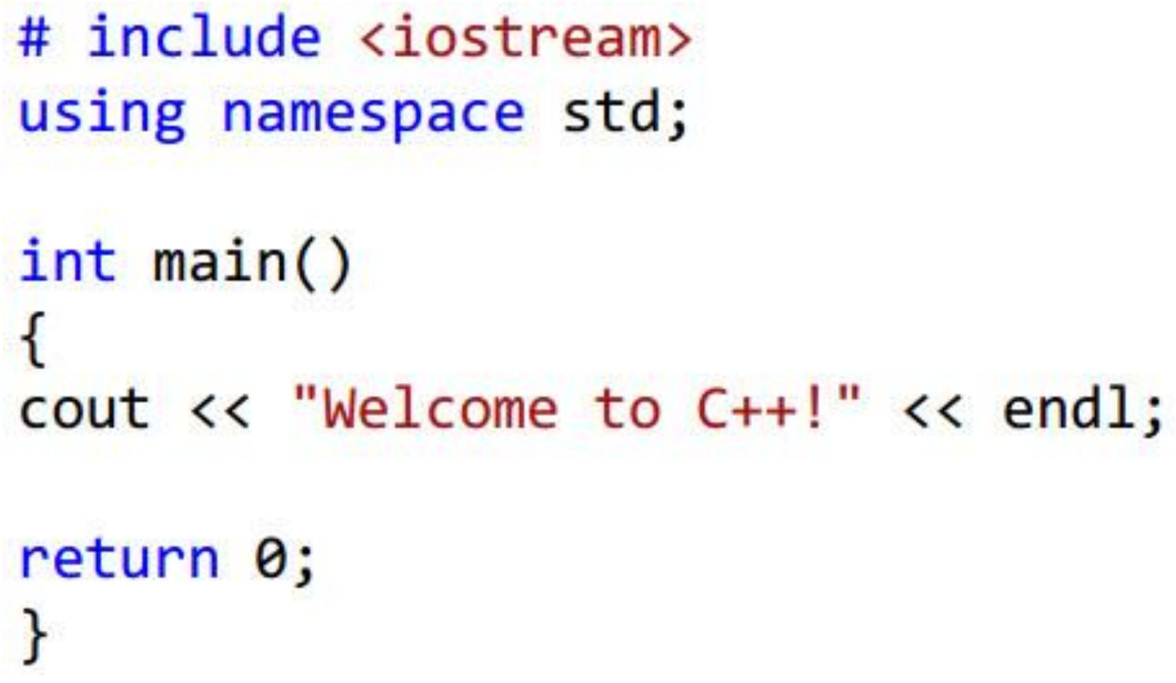
```java
// Fig. 2.1: Welcome1.java
// Text-printing program.

public class Welcome1
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.println( "Welcome to Java Programming!" );

   } // end method main

} // end clazss Welcome1
```

```
Welcome to Java Programming!
```



```cpp
# include <iostream>
using namespace std;

int main()
{
cout << "Welcome to C++!" << endl;

return 0;
}
```

2.2 First Program in Java: Printing a Line of Text (Cont.)

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

- Comments start with: `//`
  - Comments ignored during program execution
  - Document and describe code
  - Provides code readability
- Traditional comments: `/* ... */`

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

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

- Note: line numbers not part of program, added for reference

Good Programming Practice 2.1

Every program should begin with a comment that explains the purpose of the program.

2.2 First Program in Java: Printing a Line of Text (Cont.)

```
3
```

- Blank line
  - Makes program more readable
  - Blank lines, spaces, and tabs are white-space characters
    - Ignored by compiler

```
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`

Good Programming Practice 2.2

Use blank lines and space characters to enhance program readability.

2.2 First Program in Java: Printing a Line of Text (Cont.)

```
public class Welcome1
```

- Java 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

Good Programming Practice 2.3

By convention, always begin a class name's identifier with a capital letter and start each subsequent word in the identifier with a capital letter.

2.2 First Program in Java: Printing a Line of Text (Cont.)

```
4   public class Welcome1
```

- Saving files
  - File name must be class name with `.java` extension
  - `Welcome1.java`

```
5   {
```

- Left brace **{**
  - Begins body of every class
  - Right brace ends declarations (line 13)

Common Programming Error 2.3

It is an error for a `public` class to have a file name that is not identical to the class name (plus the `.java` extension) in terms of both spelling and capitalization.

Common Programming Error 2.4

It is a syntax error if braces do not occur in matching pairs.

It is an error not to end a file name with the `.java` extension for a file containing a class declaration.

2.2 First Program in Java: Printing a Line of Text (Cont.)

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

– Part of every Java application
  - Applications begin executing at `main`
    - Parentheses indicate `main` is a method
    - Java applications contain one or more methods
  - Exactly one method must be called `main`

– Methods can perform tasks and return information
  - `void` means `main` returns no information

2.2 First Program in Java: Printing a Line of Text (Cont.)

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

- Instructs computer to perform an action
  - Prints string of characters
    - String – series of characters inside double quotes
  - White-spaces in strings are not ignored by compiler
- `System.out`
  - Standard output object
- Method `System.out.println`
  - Displays line of text
- This line known as a statement
  - Statements must end with semicolon **;**

Common Programming Error 2.6

Omitting the semicolon ; at the end of a statement is a syntax error.

2.3 Modifying Our First Java Program (Cont.)

- **Modifying programs**
  - `Welcome2.java` **produces same output as** `Welcome1.java`
  - **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**

```java
// Fig. 2.3: Welcome2.java
// Printing a line of text with multiple statements.

public class Welcome2
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.print( "Welcome to " );
      System.out.println( "Java Programming!" );

   } // end method main

} // end class Welcome2
```

```
Welcome to Java Programming!
```

```cpp
#include <iostream>
using namespace std;

int main()
{
	//comments
	cout << "Welcome to “ ;
	cout << "Java Programming!“ << endl;

	return 0;
}
```

2.3 Modifying Our First Java Program (Cont.)

- **Escape characters**
  - **Backslash (\ )**
  - **Indicates special characters to be output**
- **Newline characters (`\n`)**
  - **Interpreted as "special characters" by methods `System.out.print` and `System.out.println`**
  - **Indicates cursor should be at the beginning of the next line**
  - `Welcome3.java`

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

  - **Line breaks at `\n`**

```java
// Fig. 2.4: Welcome3.java
// Printing multiple lines of text with a single statement.

public class Welcome3
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.println( "Welcome\nto\nJava\nProgramming!" );

   } // end method main

} // end class Welcome3
```

```
Welcome
to
Java
Programming!
```

A new line begins after each `\n` escape sequence is 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.

2.4 Displaying Text with `printf`

- `System.out.printf`
  - Feature added in Java SE 5.0
  - Displays formatted data

```
        System.out.printf( "%s\n%s\n",
           "Welcome to", "Java Programming!" );
```

  - Format string
    - Fixed text
    - Format specifier – placeholder for a value
  - Format specifier `%s` – placeholder for a string

```java
// Fig. 2.6: Welcome4.java
// Printing multiple lines in a dialog box.

public class Welcome4
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      System.out.printf( "%s\n%s\n",
         "Welcome to", "Java Programming!" );

   } // end method main

} // end class Welcome4
```

`System.out.printf` displays formatted data.

```
Welcome to
Java Programming!
```

2.5 Another Java Application: Adding Integers

- Upcoming program
  - Use `Scanner` to read two integers from user
  - Use `printf` to display sum of the two values
  - Use packages

```java
// Fig. 2.7: Addition.java
// Addition program that displays the sum of two numbers.
import java.util.Scanner; // program uses class Scanner

public class Addition
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number1 and number2

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

```

`import` declaration imports class `Scanner` from package `java.util`.

Declare and initialize variable `input`, which is a `Scanner`.

Declare variables `number1`, `number2` and `sum`.

Read an integer from the user and assign it to `number1`.

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

        sum = number1 + number2; // add numbers

        System.out.printf( "Sum is %d\n", sum ); // display sum

    } // end method main

} // end class Addition
```

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

2.5 Another Java Application: Adding Integers (Cont.)

```
import java.util.Scanner;  // program uses class Scanner
```

- `import` declarations
  - Used by compiler to identify and locate classes used in Java programs
  - Tells compiler to load class `Scanner` from `java.util` package

Common Programming Error 2.8

All `import` declarations must appear before the first class declaration in the file.

Placing an `import` declaration inside a class declaration's body or after a class declaration is a syntax error.

2.5 Another Java Application: Adding Integers (Cont.)

```
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );
```

- Variable Declaration Statement
- Variables
  - Location in memory that stores a value
    - Declare with name and type before use
  - `Input` is of type `Scanner`
    - Enables a program to read data for use
  - Variable name: any valid identifier
- Declarations end with semicolons `;`
- Initialize variable in its declaration
  - Equal sign
  - Standard input object
    - `System.in`

2.5 Another Java Application: Adding Integers (Cont.)

```
      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number 1 and number 2
```

- Declare variable `number1`, `number2` and `sum` of type `int`
  - `int` holds integer values (whole numbers): i.e., `0`, `-4`, `97`
  - Types `float` and `double` can hold decimal numbers
  - Type `char` can hold a single character: i.e., x, $, \n, 7

```
      int number1, // first number to add
          number2, // second number to add
          sum; // sum of number1 and number2
```

- Can declare multiple variables of the same type in one declaration
- Use comma-separated list

Good Programming Practice 2.11

Choosing meaningful variable names helps a program to be *self-documenting and readable.*

2.5 Another Java Application: Adding Integers (Cont.)

```
      System.out.print( "Enter first integer: " ); // prompt
```

- Message called a prompt - directs user to perform an action
- Package `java.lang`

```
      number1 = input.nextInt(); // read first number from user
```

- Result of call to `nextInt` given to `number1` using assignment operator =
  - Assignment statement
  - = binary operator - takes two operands
    - Expression on right evaluated and assigned to variable on left
  - Read as: `number1` gets the value of `input.nextInt()`

Software Engineering Observation 2.1

By default, package `java.lang` is imported in every Java program; thus, `java.lang` is the only package in the Java API that does not require an `import` declaration.

2.5 Another Java Application: Adding Integers (Cont.)

```
            sum = number1 + number2; // add numbers
```

– Assignment statement
  - Calculates sum of `number1` and `number2` (right hand side)
  - Uses assignment operator = to assign result to variable `sum`
  - Read as: `sum` gets the value of `number1 + number2`
  - `number1` and `number2` are operands

2.5 Another Java Application: Adding Integers (Cont.)

```
System.out.printf( "Sum is %d\n: " , sum ); // display sum
```

- Use `System.out.printf` to display results
- Format specifier `%d`
  - Placeholder for an `int` value

```
  System.out.printf( "Sum is %d\n: " , ( number1 + number2 ) );
```

- Calculations can also be performed inside `printf`
- Parentheses around the expression `number1` + `number2` are not required

2.6 Memory Concepts

- **Variables**
  - **Every variable has a name, a type, a size and a value**
    - **Name corresponds to location in memory**
  - **When new value is placed into a variable, replaces (and destroys) previous value**
  - **Reading variables from memory does not change them**

2.7 Arithmetic

- Arithmetic calculations used in most programs
  - Usage
    - `*` for multiplication
    - `/` for division
    - `%` for remainder
    - `+`, `-`
  - Integer division truncates remainder

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

    `7 % 5` evaluates to `2`

Java operation	Arithmetic operator	Algebraic expression	Java expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`–`	$p - c$	`p - c`
Multiplication	`*`	bm	`b * m`
Division	`/`	x / y or $\frac{x}{y}$ or $x \div y$	`x / y`

Arithmetic operators.

2.7 Arithmetic (Cont.)

- **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`

Operator(s)	Operation(s)	Order of evaluation (precedence)

`*`
`/`
`%`
	Multiplication
	Division
	Remainder
		Evaluated first. If there are several operators of this type, they are evaluated from left to right.

`+` `-`	Addition Subtraction	Evaluated next. If there are several operators of this type, they are evaluated from left to right.

Precedence of arithmetic operators.

Good Programming Practice 2.14

Using parentheses for complex arithmetic expressions, even when the parentheses are not necessary, can make the arithmetic expressions easier to read.

2.8 Decision Making: Equality and Relational Operators

- **Condition**
  - **Expression can be either `true` or `false`**
- **`if` statement**
  - **Simple version in this section, more detail later**
  - **If a condition is `true`, then the body of the `if` statement executed**
  - **Control always resumes after the `if` statement**
  - **Conditions in `if` statements can be formed using equality or relational operators (next slide)**

Standard algebraic equality or relational operator	Java equality or relational operator	Sample 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`

Equality and relational operators.

```java
// Fig. 2.15: Comparison.java
// Compare integers using if statements, relational operators
// and equality operators.
import java.util.Scanner; // program uses class Scanner

public class Comparison
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to compare
      int number2; // second number to compare

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

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

      if ( number1 == number2 )
         System.out.printf( "%d == %d\n", number1, number2 );

      if ( number1 != number2 )
         System.out.printf( "%d != %d\n", number1, number2 );

      if ( number1 < number2 )
         System.out.printf( "%d < %d\n", number1, number2 );
```

Test for equality, display result using `printf`.

Compares two numbers using relational operator <.

```java

      if ( number1 > number2 )
         System.out.printf( "%d > %d\n", number1, number2 );

      if ( number1 <= number2 )
         System.out.printf( "%d <= %d\n", number1, number2 );

      if ( number1 >= number2 )
         System.out.printf( "%d >= %d\n", number1, number2 );

   } // end method main

} // end class Comparison
```

Compares two numbers using relational operators >, <= and >=.

```
Enter first integer: 777
Enter second integer: 777
777 == 777
777 <= 777
777 >= 777
```

```
Enter first integer: 1000
Enter second integer: 2000
1000 != 2000
1000 < 2000
1000 <= 2000
```

```
Enter first integer: 2000
Enter second integer: 1000
2000 != 1000
2000 > 1000
2000 >= 1000
```

2.8 Decision Making: Equality and Relational Operators (Cont.)

```
23        if ( number1 == number2 )
24           System.out.printf( "%d == %d\n", number1, number2 );
```

- `if` statement to test for equality using (==)
  - If variables equal (condition true)
    - Line 24 executes
  - If variables not equal, statement skipped
  - No semicolon at the end of if line
  - Empty statement
    - No task is performed

Common Programming Error 2.11

Reversing the operators !=, >= and <=, as in =!, => and =<, is a syntax error.

It is a syntax error if the operators ==, !=, >= and <= contain spaces between their symbols, as in = =, ! =, > = and < =, respectively.

Common Programming Error 2.13

Placing a semicolon immediately after the right parenthesis of the condition in an `if` statement is normally a logic error.

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