2

Introduction to Java Applications

Last time

- administrative info
- course introduction
- introduction to Java programming language
- why study Java
- hello world program

Objectives

- practice with Eclipse IDE (lab this week)
- program output (System.out, println, printf)
- program input (System.in, Scanner, JOptionPane)
- Java's primitive types (int, double, boolean, etc.)
- arithmetic and relational operators
- control statements (if-else, for loops, etc.)

Outline

2.1 Introduction

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

First Program in Java: Printing a Line of Text

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

Outline

Welcome1.java

```
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!
```

First Program in Java (Cont.)

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

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

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

First Program in Java (Cont.)

```
3
```

- Blank line is ignored by compiler
- Blank lines, spaces, and tabs are white-space characters

```
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`

Java Identifiers

- **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**

First Program in Java (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)

First Program in Java (Cont.)

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

- Part of every executable Java application
- Applications begin executing at `main`
  - Parentheses indicate `main` is a method (i.e., function)
  - Exactly one method must be called `main`
- Methods can perform tasks and return information
  - `void` means `main` returns no information

```
8      {
```

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

First Program in Java (Cont.)

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

- Method to print a string of characters to system output
- `System.out`
  - Standard output object
  - Print to command window (console)
- Method `System.out.println`
  - Displays line of text
- This line known as a statement and must end with semicolon ;

First Program in Java (Cont.)

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

- Ends method declaration

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

- Ends class declaration
- Can add comments to keep track of ending braces

First Program in Java (Cont.)

- **Compiling a program**
  - **Go to directory where program is stored**
  - **Type `javac Welcome1.java`**
  - **If no syntax errors, `Welcome1.class` created**
    - **Has bytecodes that represent application**
    - **Bytecodes passed to JVM**

First Program in Java (Cont.)

- **Executing a program**
  - Type `java Welcome1`
    - Launches JVM
    - JVM loads `.class` file for class `Welcome1`
    - `.class` extension omitted from command
    - JVM calls method `main`

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**

```
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!
```

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

Outline

Welcome3.java

1. main

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

Program 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.

Displaying Text with `printf`

- `System.out.printf`
  - Displays formatted data

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

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

Outline

```
1  // Fig. 2.6: Welcome4.java
2  // Printing multiple lines in a dialog box.
3
4  public class Welcome4
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",
10          "Welcome to", "Java Programming!" );
11
12    } // end method main
13
14 } // end class Welcome4
```

System.out.printf displays formatted data.

Welcome4.java

main

printf

```
Welcome to
Java Programming!
```

Program output

Another Java Application: Adding Integers

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

Outline

```
1  // Fig. 2.7: Addition.java
2  // Addition program that displays the sum of two numbers.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class Addition
6  {
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10       // create Scanner to obtain input from command window
11       Scanner input = new Scanner( System.in );
12
13       int number1; // first number to add
14       int number2; // second number to add
15       int sum; // sum of number1 and number2
16
17       System.out.print( "Enter first integer: " ); // prompt
18       number1 = input.nextInt(); // read first number from user
19
```

`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`.

Addition.java

(1 of 2)

```
20       System.out.print( "Enter second integer: " ); // prompt
21       number2 = input.nextInt(); // read second number from user
22
23       sum = number1 + number2; // add numbers
24
25       System.out.printf( "Sum is %d\n", sum ); // d
26
27    } // end method main
28
29 } // end class Addition
```

Read an integer from the user and assign it to `number2`.

Calculate the sum of the variables `number1` and `number2`, assign result to `sum`.

Display the sum using formatted output.

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

Two integers entered by the user.

Outline

dition.java

of 2)

4. Addition

5. `printf`

Java API Packages

- Including the declaration

  ```
  import java.util.Scanner;
  ```

  allows the programmer to use `Scanner` instead of `java.util.Scanner`

- Java API documentation
  - http://docs.oracle.com/javase/8/docs/api/

Import declarations

```
3  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

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.

Error-Prevention Tip

Forgetting to include an `import` declaration for a class used in your program typically results in a compilation error containing a message such as "`cannot resolve symbol`." When this occurs, check that you provided the proper `import` declarations and that the names in the `import` declarations are spelled correctly, including proper use of uppercase and lowercase letters.

Software Engineering Observation

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.

Java Graphical User Interface (GUI)

- **Many libraries for GUI programming**
  - **java.awt**
  - **javax.swing**
  - **JavaFX (latest GUI library)**

Displaying Text in a Dialog Box

- **Windows and dialog boxes**
  - **Many Java applications use these to display output**
  - **`JOptionPane` provides prepackaged dialog boxes called message dialogs**

Outline

Dialog1.java

```
1  // Fig. 3.17: Dialog1.java
2  // Printing multiple lines in dialog box.
3  import javax.swing.JOptionPane; // import class JOptionPane
4
5  public class Dialog1
6  {
7     public static void main( String args[] )
8     {
9        // display a dialog with the message
10       JOptionPane.showMessageDialog( null, "Welcome\nto\nJava" );
11    } // end main
12 } // end class Dialog1
```

Import class `JOptionPane`

Show a message dialog with text

Displaying Text in a Dialog Box

- **Package `javax.swing`**
  - **Classes to help create graphical user interfaces (GUIs)**
  - **Contains class `JOptionPane`**
    - **Declares `static` method `showMessageDialog` for displaying a message dialog**

Entering Text in a Dialog Box

- **Input dialog**
  - **Allows user to input information**
  - **Created using method `showInputDialog` from class `JOptionPane`**

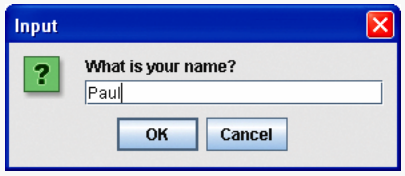
Outline

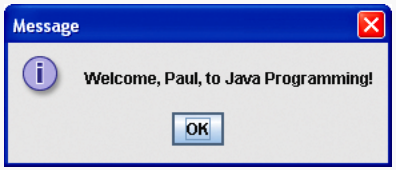
NameDialog.java

```
1  // Fig. 3.18: NameDialog.java
2  // Basic input with a dialog box.
3  import javax.swing.JOptionPane;
4
5  public class NameDialog
6  {
7     public static void main( String args[] )
8     {
9        // prompt user to enter name
10       String name =
11          JOptionPane.showInputDialog( "What is your name?" );
12
13       // create the message
14       String message =
15          String.format( "Welcome, %s, to Java Programming!", name );
16
17       // display the message to welcome the user by name
18       JOptionPane.showMessageDialog( null, message );
19    } // end main
20 } // end class NameDialog
```

Show input dialog

Format a `String` to output to user





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**

Primitive Types vs. Reference Types

- **Types in Java**
  - **Primitive**
    - `boolean`, `byte`, `char`, `short`, `int`, `long`, `float`, `double`
  - **Reference (sometimes called nonprimitive types)**
    - **Objects**
    - **Default value of `null`**

Java primitive data types

Table 2-1. Java primitive data types

Type	Contains	Default	Size	Range
`boolean`	`true` or `false`	`false`	1 bit	NA
`char`	Unicode character	`\u0000`	16 bits	`\u0000` to `\uFFFF`
`byte`	Signed integer	0	8 bits	–128 to 127
`short`	Signed integer	0	16 bits	–32768 to 32767
`int`	Signed integer	0	32 bits	–2147483648 to 2147483647
`long`	Signed integer	0	64 bits	–9223372036854775808 to 9223372036854775807
`float`	IEEE 754 floating point	0.0	32 bits	±1.4E-45 to ±3.4028235E+38
`double`	IEEE 754 floating point	0.0	64 bits	±4.9E-324 to ±1.7976931348623157E+308

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`

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	Evaluated first. If there are several operators of this type, they are evaluated from left to right.
/	Division	
%	Remainder	
+	Addition	Evaluated next. If there are several operators of this type, they are evaluated from left to right.
-	Subtraction	

Fig. 2.12 | Precedence of arithmetic operators.

Decision Making: 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

Fig. 2.14 | Equality and relational operators.

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

Fig. 2.16 | Precedence and associativity of operations discussed.