Chapter 2
Introduction to Java Applications

Xuetao Wei

weixt@sustech.edu.cn

Our First Java Program

```java
// Text-printing program
public class Welcome1 {
    // main method begins the execution of Java application
    public static void main(String[] args) {
        System.out.println("Welcome to Java Programming!");
    }
}
```

`Welcome1` prints the following text in the command window (console):

```
Welcome to Java Programming!
```

Comments

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

- `//` indicates that the line is a comment.
- Comments help document programs to improve their readability.
- Compiler ignores comments.

Traditional comments begin with `/*` and end with `*/`. They can be spread over several lines

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

Traditional vs. End-of-Line Comments

- Traditional comments do not nest (嵌套), the first `*/` after the first `/*` will terminate the comment

```
/*
  /* comment 1 */
     ~~comment 2 */~~
```

Syntax Error (语法错误)

- End-of-line comments can contain anything

```
// /* this comment is okay */
```

Class Declaration

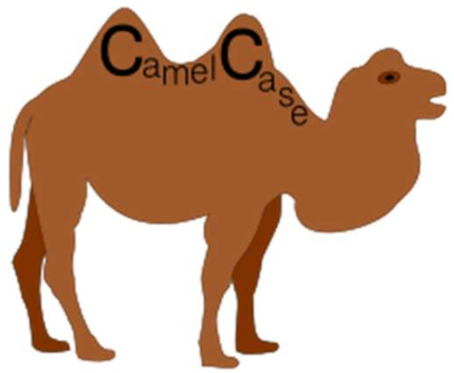
```
public class Welcome1
```

- Every Java program consists of at least one class (类) that you define
- The class keyword introduces a class declaration and is immediately followed by the class name
- Keywords are reserved for use by Java and are always spelled with all lowercase letters (we will see more later)

Identifiers

- A name in a Java program is called an **identifier**, which is used for identification purpose.
  - “`Welcome1`” is an identifier. It is the name for the class we just defined.
- The only allowed characters in Java identifiers are **a to z, A to Z, 0 to 9, $** and _ (underscore).
- Identifiers can't start with digits, e.g., **123name** is invalid.

Class Names



- By convention, class names begin with a capital letter and capitalize the first letter of each word they include (upper camel case, 大驼峰式命名法)
- Java is case sensitive—uppercase and lowercase letters are distinct (not in comments). “`Name`” and “`name`” are different identifiers.

The Braces

- A left brace { begins the declaration of every class and method
- A corresponding right brace } ends the declaration of each class and method
- Code between braces should be indented (good practice)

```
public class Welcome1 {
    public static void main(String[] args) {
        System.out.println("Welcome to Java Programming!");
    }
}
```

The main method

```
public static void main(String[] args) {
        System.out.println("Welcome to Java Programming!");
}
```

- Starting point of Java applications
- A method groups code that collectively achieves a functionality
- Parentheses after the identifier `main` enclose formal parameters (形式参数)
- Java class declarations normally contain one or more methods
- Keyword `void` indicates that this method will not return any data

The `main` method body

- Enclosed in left and right braces
- The statement in the method instructs the computer to print the string of characters contained between the double quotation marks

```
public static void main(String[] args) {
        System.out.println("Welcome to Java Programming!");
}
```

The System.out Object

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

- System.out is the standard output object that allows Java applications to display strings in the command window
- `System.out.println` method
  - Displays (or prints) a line of text in the command window
  - The string in the parentheses is the actual argument (实际参数) to the method
  - Positions the output cursor at the beginning of the next line in the command window

Compile `Welcome1.java`

- Type the following command in the command line interface to compile the program

```
javac Welcome1.java
```

- If the program contains no syntax errors, the command creates a `Welcome1.class` file (known as the class file) containing the platform-independent Java bytecodes that represent the application

Execute `Welcome1`

- To execute the program, type `java Welcome1`
- The command launches the JVM, which loads the `.class` file for class `Welcome1` and executes the program
- The `.class` file-name extension is omitted from the command; otherwise, JVM will not execute the program.

Modifying `Welcome1.java`

```
// Print a line of text with multiple statements
public class Welcome2 {
    public static void main(String[] args) {
        System.out.print("Welcome to ");
        System.out.print("Java Programming!");
    }
}
```

Class `Welcome2` uses two statements to produce the same output as class `Welcome1`

```
Welcome to Java Programming!
```

The `System.out.print()` method

- `System.out`'s method `print` displays a string
- Unlike the method `println`, `print` does not position the output cursor at the beginning of the next line in the command window (it simply prints the string)

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

Continue the modification

```java
// Print multiple lines of text using a single statement
public class Welcome3 {
    public static void main(String[] args) {
        System.out.println("Welcome\nto\nJava\nProgramming!");
    }
}
```

`Welcome3` prints the following text on the console:

```
Welcome
to
Java
Programming!
```

The newline character \n

- Newline characters instruct `System.out`’s `print` and `println` methods to position the output cursor at the beginning of the next line in the command window

- Newline characters are white-space characters, which represent horizontal or vertical space in typography and do not correspond to visible marks

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

Escape character

- The backslash (\) is an escape character (转义字符, a case of metacharacters), which invokes an alternative interpretation on subsequent characters

- Backslash \ is combined with the next character to form an escape sequence (转义序列)

- The escape sequence \n represents the newline character

Common Escape Sequences

Sequence	Description
`\n`	Newline. Position the cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the cursor to the next tab stop.
`\r`	Carriage return. Position the 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.
`\\`	Used to print a backslash character.
`\"`	Used to print a double-quote character. `System.out.println("\"in quotes\"");` displays `"in quotes"`

Displaying text with printf

```
// Print multiple lines with printf
public class Welcome4 {
    public static void main(String[] args) {
        System.out.printf("%s\n%s\n", "Welcome to",
                                "Java Programming!");
    }
}
```

`Welcome4` prints the following text on the console:

```
Welcome to
Java Programming!
```

The printf method

- printf displays “formatted” data

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

- It takes a format string (格式字符串) as an argument.
- The format specifiers (格式说明符) begin with a percent sign (%) and are followed by a character that represents the data type (e.g., %s is a placeholder for a string)

Tabulating output with `printf`

```
radius          perimeter       area

1               6.2832          3.1416

2               12.5664         12.5664

3               18.8496         28.2743

4               25.1327         50.2655
```

How to generate beautiful tables using `printf`?

Here is the magic code

```
double pi = Math.PI;
System.out.printf("%-20s%-20s%-20s\n", "radius", "perimeter", "area");
System.out.printf("%-20d%-20.4f%-20.4f\n", 1, 2*pi*1, pi*1*1);
System.out.printf("%-20d%-20.4f%-20.4f\n", 2, 2*pi*2, pi*2*2);
System.out.printf("%-20d%-20.4f%-20.4f\n", 3, 2*pi*3, pi*3*3);
System.out.printf("%-20d%-20.4f%-20.4f\n", 4, 2*pi*4, pi*4*4);
```

Please decode the format strings by yourself.

Check out the self-study materials on Blackboard.