2.2 A First Simple Program

IT1406 - Introduction to Programming

Level I - Semester 1

2.2 A First Simple Program

- Let's look at some actual Java programs.
- Let's start by compiling and running the short sample program shown here.
- As one can see, this involves a little more work than one might imagine.
- NOTE
- The descriptions that follow use the standard Java SE 8 Development Kit (JDK 8), which is available from Oracle. If you are using a different Java development environment, then you may need to follow a different procedure for compiling and executing Java programs. In this case, consult your compiler's documentation for details.

2.2.1.
Writing and saving the Java source file

```
/* This is a simple Java program. Call this file
 "Example.java". */
 class Example {
// Your program begins with a call to main().
public static void main(String args[]) {
  System.out.println("This is a simple Java program.");
    }
}
```

Entering the Program

- For most computer languages, the name of the file that holds the source code to a program is immaterial.
- However, this is not the case with Java.
- The first thing that you must learn about Java is that the name you give to a source file is very important.
- For this example, the name of the source file should be Example.java. Let's see why.
- In Java, a source file is officially called a *compilation unit*.
- It is a text file that contains (among other things) one or more class definitions. (For now, we will be using source files that contain only one class.)
- The Java compiler requires that a source file use the .java filename extension.

Entering the Program

- As you can see by looking at the program, the name of the class defined by the program is also Example.
- This is not a coincidence. In Java, all code must reside inside a class.
- By convention, the name of the main class should match the name of the file that holds the program.
- One should also make sure that the capitalization of the filename matches the class name.
- The reason for this is that Java is case-sensitive.
- At this point, the convention that filenames correspond to class names may seem arbitrary.
- However, this convention makes it easier to maintain and organize your programs.

2.2.2 Compiling the Program

- To compile the Example program, execute the compiler, javac, specifying the name of the source file on the command line, as shown here:

C:\>javac Example.java

- The javac compiler creates a file called Example.class that contains the bytecode version of the program.
- As discussed earlier, the Java bytecode is the intermediate representation of your program that contains instructions the Java Virtual Machine will execute.
- Thus, the output of javac is not code that can be directly executed.

2.2.2 Executing the Program

- To actually run the program, one must use the Java application launcher called java.
- To do so, pass the class name Example as a command-line argument, as shown here:

  C:\>java Example
- When the program is run, the following output is displayed:

  This is a simple Java program.
- When Java source code is compiled, each individual class is put into its own output file named after the class and using the .class extension.

2.2.2 Executing the Program

- This is why it is a good idea to give your Java source files the same name as the class they contain—the name of the source file will match the name of the .class file.
- When execute java as just shown, actually specifying the name of the class that one want to execute. It will automatically search for a file by that name that has the .class extension.
- If it finds the file, it will execute the code contained in the specified class.

2.2.2 Executing the Program

- To actually run the program, one must use the Java application launcher called java.
- To do so, pass the class name Example as a command-line argument, as shown here:

  C:\>java Example

- When the program is run, the following output is displayed:

  This is a simple Java program.

2.2.2 Executing the Program

- When Java source code is compiled, each individual class is put into its own output file named after the class and using the .class extension.
- This is why it is a good idea to give your Java source files the same name as the class they contain—the name of the source file will match the name of the .class file.
- When execute java as just shown, actually specifying the name of the class that one want to execute. It will automatically search for a file by that name that has the .class extension.
- If it finds the file, it will execute the code contained in the specified class.

2.2.3 Structure of a Java program

- Although Example.java is quite short, it includes several key features that are common to all Java programs.
- Let's closely examine each part of the program.

2.2.3 Structure of a Java program

- The program begins with the following lines:

/* This is a simple Java program. Call this file "Example.java". */

- This is a *comment*. Like most other programming languages, Java lets you enter a remark into a program's source file.
- The contents of a comment are ignored by the compiler. Instead, a comment describes or explains the operation of the program to anyone who is reading its source code.
- In this case, the comment describes the program and reminds you that the source file should be called Example.java. Of course, in real applications, comments generally explain how some part of the program works or what a specific feature does.
- Java supports three styles of comments. The one shown at the top of the program is called a *multiline comment*. This type of comment must begin with /* and end with */. Anything between these two comment symbols is ignored by the compiler. As the name suggests, a multiline comment may be several lines long.

2.2.3 Structure of a Java program

- The next line of code in the program is shown here:

  class Example {

- This line uses the keyword class to declare that a new class is being defined.
- Example is an *identifier* that is the name of the class. The entire class definition, including all of its members, will be between the opening curly brace ({) and the closing curly brace (}).
- For the moment, don't worry too much about the details of a class except to note that in Java, all program activity occurs within one.
- This is one reason why all Java programs are (at least a little bit) object-oriented.

2.2.3 Structure of a Java program

- The next line in the program is the *single-line comment*, shown here:

// Your program begins with a call to main().

- This is the second type of comment supported by Java.
- A *single-line comment* begins with a // and ends at the end of the line. A
- s a general rule, programmers use multiline comments for longer remarks and single-line comments for brief, line-by-line descriptions.
- The third type of comment, a *documentation comment*, has the following form

*/** we can write documentation comments to document*

code related writings ***/***

2.2.4.
Using blocks of code and issues of lexical in Java at a glance

Using Blocks of Code

- Java allows two or more statements to be grouped into *blocks of code*, also called *code blocks (compound statements)*.
- This is done by enclosing the statements between opening and closing curly braces.
- Once a block of code has been created, it becomes a logical unit that can be used any place that a single statement can.
- For example, a block can be a target for Java's if and for statements.

2.2.4.
Using blocks of code and issues of lexical in Java at a glance

Using Blocks of Code

- Consider this if statement:

```
if(x < y) { // begin a block
        x = y; y = 0;
        } // end of block
```

- Here, if x is less than y, then both statements inside the block will be executed.
- Thus, the two statements inside the block form a logical unit, and one statement cannot execute without the other also executing.
- The key point here is that whenever you need to logically link two or more statements, you do so by creating a block.

2.2.4. Using blocks of code and issues of lexical in Java at a glance

Using Blocks of Code

- Let's look at another example. The following program uses a block of code as the target of a for loop.

```
/* Demonstrate a block of code. Call this file "BlockTest.java" */
class BlockTest {
 public static void main(String args[]) {
  int x, y; y = 20; // the target of this loop is a block
for(  x = 0; x<10; x++) {
    System.out.println("This is x: " + x);
     System.out.println("This is y: " + y);
    y = y - 2; } } }
```

2.2.4.
Using blocks of code and issues of lexical in Java at a glance

Using Blocks of Code

- In this case, the target of the for loop is a block of code and not just a single statement.
- Thus, each time the loop iterates, the three statements inside the block will be executed.
- This fact is, of course, evidenced by the output generated by the program.
- blocks of code have additional properties and uses. However, the main reason for their existence is to create logically inseparable units of code.

2.2.4.
Using blocks of code and issues of lexical in Java at a glance

Using Blocks of Code

- In this case, the target of the for loop is a block of code and not just a single statement.
- Thus, each time the loop iterates, the three statements inside the block will be executed.
- This fact is, of course, evidenced by the output generated by the program.
- blocks of code have additional properties and uses. However, the main reason for their existence is to create logically inseparable units of code.

2.2.4.
Using blocks of code and issues of lexical in Java at a glance

Using Blocks of Code

- In this case, the target of the for loop is a block of code and not just a single statement.
- Thus, each time the loop iterates, the three statements inside the block will be executed.
- This fact is, of course, evidenced by the output generated by the program.
- blocks of code have additional properties and uses. However, the main reason for their existence is to create logically inseparable units of code.

2.2.4.
Using blocks of code and issues of lexical in Java at a glance

Lexical Issues

- Now that you have seen several short Java programs, it is time to more formally describe the atomic elements of Java.
- Java programs are a collection of whitespace, identifiers, literals, comments, operators, separators, and keywords.
- The operators are described in the next chapter. The others are described next.

2.2.4.
Using blocks of code and issues of lexical in Java at a glance

Whitespace

- Java is a free-form language.
- This means that you do not need to follow any special indentation rules.
- For instance, the Example program could have been written all on one line or in any other strange way you felt like typing it, as long as there was at least one whitespace character between each token that was not already delineated by an operator or separator.
- In Java, whitespace is a space, tab, or newline.

2.2.4.
Using blocks of code and issues of lexical in Java at a glance

Identifiers

- Identifiers are used to name things, such as classes, variables, and methods.
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers, or the underscore and dollar-sign characters. (The dollar-sign character is not intended for general use.)
- They must not begin with a number, lest they be confused with a numeric literal.
- Again, Java is case-sensitive, so VALUE is a different identifier than Value.

2.2.4.
Using blocks of code and issues of lexical in Java at a glance

Identifiers

- Some examples of valid identifiers are:

  num1, average2, $Mark, secondName, value_One
- Invalid identifier names include these:

  2ndNumbers, *Mark, values-given, marks/value

2.2.4 Comments in Java

- **There are three types of comments in Java**
  - **Single line comments(C++ Style)**
    - **Started with //**

Example

```
// This is a Single line Comment
```

2.2.4
Comments Contd...

- **Multi Line Comments**
  - **Started with /* and Ended with */ (C Style)**

Example

```
/* This is a
        Multi line Comment */
```

2.2.4 Comments Contd...

- Comments that can be used in Automatic Class Documentation by javadoc tool
  - Started with /** and Ended with */
  - Can Span Multiple lines as well

Example

```
/** The Integer class  Encapsulates
primitive Data Type int .. */
```

2.2.4
Comments Contd...

- **Multi Line Comments**
  - **Started with /* and Ended with */ (C Style)**

Example

```
/* This is a
        Multi line Comment */
```

2.2.4 Comments Contd...

- Comments that can be used in Automatic Class Documentation by javadoc tool
  - Started with /** and Ended with */
  - Can Span Multiple lines as well

Example

```
/** The Integer class  Encapsulates
primitive Data Type int .. */
```

2.2.4 Separators

- In Java, there are a few characters that are used as separators.
- The most commonly used separator in Java is the semicolon.
- As one can see, it is used to terminate statements.

2.2.4 Separators

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

2.2.5 Java Key words

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while