Chapter 2: Programming Concepts

Objectives

Students should

- Know the steps required to create programs using a programming language and related terminology.
- Be familiar with the basic structure of a Java program.
- Be able to modify simple Java program to obtain desired results.

Programming Languages

A *program* is a set of instructions that tell a computer what to do.

We *execute* a program to carry out the instruction listed by that program.

Instructions are described using *programming languages*.

High-level programming languages are programming languages that are rather natural for people to write. Examples of high-level programming languages include Java, C, C++, C#, Visual Basic, Pascal, Delphi, FORTRAN, and COBOL.

The most primitive type of programming language is a *machine language* or *object code*. Object code is a set of binary code that is unique to the type of CPU. Each instruction of the object code corresponds to a fundamental operation of the CPU. That means it usually requires many object code instructions to do what can be done in one line of high-level language code. Writing object code directly is tedious and error-prone.

High-level language code does not get executed on a computer directly. It needs to be translated from the high-level language to suitable machine language first. Such a translator program is normally referred to as a *compiler*.

The input of a compiler is referred to as *source code*. Source code is *compiled* to by a compiler to obtain *target code*.

Running a Java Program

Unlike other programming languages, compiling Java source code does not result in a machine language program. Instead, when Java source code is compiled, we get what is called Java *bytecode*. Java bytecode is a form of machine language instructions. However, it is not primitive to the CPU. Java bytecode runs on a program that mimics itself as a real machine. This program is called the *Java Virtual Machine* (JVM).

This architecture makes Java bytecode runs on any machines that have JVM, independent of the OSs and CPUs. This means the effort in writing Java source code for a certain program is spent once and the target program can run on any platforms. (E.g. Windows, MacOS, Unix, etc.)

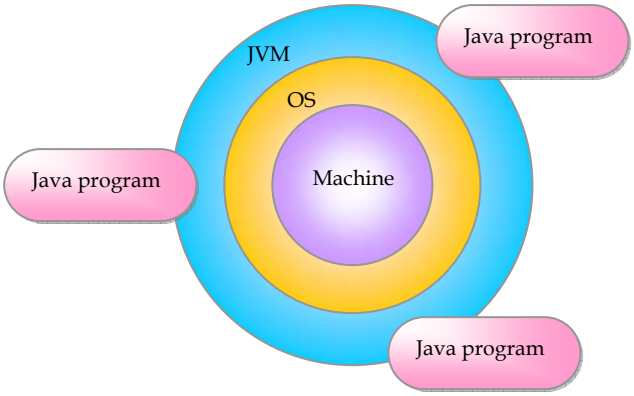


Figure 1: Platform-independent architecture of Java language

Typical Programming Cycle

During the implementation of a program, programmers always run into a typical cycle, as shown in the figure below.

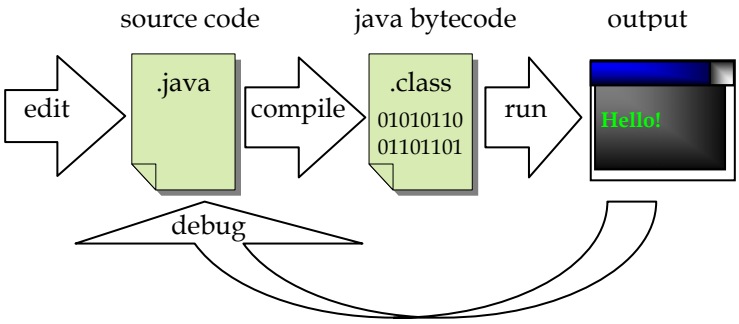


Figure 2: Typical cycle in Java programming

It normally starts with the coding step, in which the source code is written in any text editors or integrated programming environment. In Java, source code files are usually saved with *.java* extension. Once the Java source code is saved, it is compiled using a java compiler, such as *javac*, to obtain the resulting Java bytecode, which is also called a Java *class*. The actual Java class file is created with *.class* extension. Then, the program is executed by running the command '*java*' on the class file. If the result appears as expected, the cycle terminates. Otherwise, the source code has to be edited, then compiled and executed again. The process of fixing the source code in order to obtain the right result is called 'debugging'.

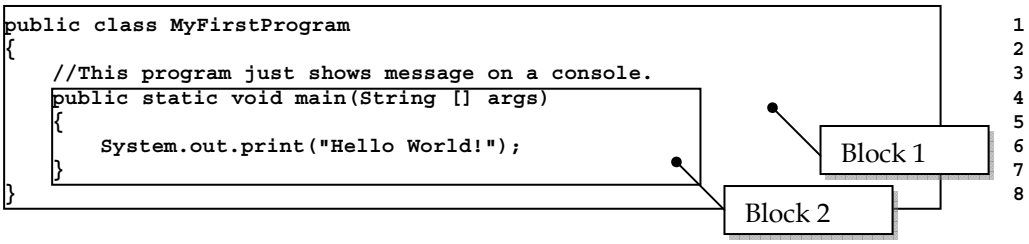
Java Basic Program Structure

Let's look at a source code named "MyFirstProgram.java" below. Note that the number at the end of each line indicates line number. They are not part of the source code.

```
public class MyFirstProgram                                    1
{                                                              2
    //This program just shows message on a console.            3
    public static void main(String [] args)                    4
    {                                                          5
        System.out.print("Hello World!");                      6
    }                                                          7
}                                                              8
```

When compiled, this program shows the message "Hello World!" on screen. Now we will investigate the structure of this Java source code in order to get familiar with the basic structure.

First of all, we can observe that there are two pairs of curly brackets {}. The opening curly bracket on line 2 and the closing curly bracket on the last line make a pair. The ones on line 5 and line 7 make another pair locating inside the first one. Here, we will refer to the content between a pair of curly bracket together with its associated header, i.e. `public class MyFirstProgram` for the outer pair or `public static void main(String [] args)` for the inner pair, as "block". The idea of looking at the program structure as blocks can be illustrated below.



Block 1 is the definition of the program. Block 2 is a part of the definition which is called the *main*() method. The symbol // is used for indicating that any texts after that until the end of the line are comments, which are ignored by the compiler. They are just there for programmers to make notes for themselves or other people reading their source code. Thus, we can say that the definition of this program consists of a comment on line 3 and the main() method from line 4 to 7.

The main() method is the program's entry point. Program is executed following the commands listed in the main method, starting from the top. From this example, when the program is executed, it is the statement on line 6 that prints the message "Hello World!" (without double quotes) on screen.

In Java, what we do all the time is creating what are called *classes*. In this MyFirstProgram .java source code, we create a class named MyFirstProgram. Each class that is created in Java can have one or more *methods*, which are units of code that perform some action. They are generally similar to *functions*, *procedures*, or *subroutines* in other programming languages. A class whose definition contains the main() method can be executed, or, in other words, is a program. Classes that do not have the main() method cannot be executed. They are there for other program to use.

Now that you have heard about the term *class*, Block 1 in the above example is called the class definition of the class names MyFirstProgram, which is a Java program since it has the main() method. The statement `public class MyFirstProgram` indicates the starting of the class definition, while the statement `public static void main(String [] args)` explains how the method named main() should be used. We will go into details about these later.

Note that Java requires the source code to be saved using the same name as the class contained in the file (with the extension .java). Only one class can be in each .java file.

Below is an example of a Java program that is perfectly correct but seems to do nothing at all.

```
public class MyLazyProgram                                              1
{                                                                       2
    public static void main(String [] args)                            3
    {                                                                   4
    }                                                                   5
}                                                                       6
```

Syntax, Keywords, and Identifiers

Writing program in Java, or other computer programming languages, is like writing in any human spoken language but a lot more formal. When writing in a sentence in a human language, we need to follow its grammar. Programming languages also have syntax or rules of the language and they have to be followed strictly.

Keywords are words that are reserved for particular purpose. They have special meanings and cannot be changed or used for other purposes. For example, from the class definition header in the last example, `public class MyFirstProgram` , the word "public" and "class" are keywords. Thus, both words cannot be used in Java programs for naming classes, methods, or variables.

Identifiers are names that programmers give to classes, methods, and variables. There are certain rules and styles for Java naming, which we will discussed in details later in this course. For the previous example, `public class MyFirstProgram`, the word "MyFirstProgram" is a Java identifier selected as the name of the class.

Note that Java is a case-sensitive language. Uppercase and lowercase letters are not considered the same. Java keywords are only consisted of lowercase letters.

Comments

It is a good practice to always add meaningful comments into the source code. There are two ways of commenting in Java source code. The first way is called single line commenting. It is done by using double slashes // as shown in the previous example. Any texts between and including // and the end of the line are ignored by the compiler.

The second way is called block commenting. In contrary to the first way, this can make comment span more than one line. Any texts that are enclosed by a pair of the symbol /* and */ are ignored by the compiler, no matter how many lines there are.

The following program, MyCommentedProgram.java, yields the same result as MyFirstProgram.java. The green (lighter) texts are the parts commented out.

```
// An example of how to add comments                                    1
// This is a part of chapter 2 of 2140101 class notes                   2
// Written by Atiwong Suchato                                           3
                                                                        4
public class MyCommentedProgram                                         5
{                                                                       6
    //Program starting point                                            7
    public static void main(String [] args)                            8
    {                                                                   9
        System.out.print("Hello World!");                              10
        //This part of code is commented out.                           11
        /*                                                              12
        System.out.print("Hello again.");                              13
        System.out.println();                                           14
        */                                                              15
    }//end of main()                                                    16
}// end of class                                                       17
```

Be Neat and Organized

Whitespace characters between various elements of the program are ignored during the compilation. Together with comments, good programmers use space, new line, and indentation to make their program more readable.

Observe the different between the following two source code listings.

```
public class ExampleClass{
public static void main(String [] args){
        double x=5;double z,y=2;z=x+y;
System.out.println(z);
}}
```

```
public class ExampleClass
{
    public static void main(String [] args)
    {
        double x=5;
        double z,y=2;
        z=x+y;
        System.out.println(z);
    }
}
```

Both perform exactly the same task. Still, it is obvious that the bottom one is easier to read and follow. Indentation is used in the bottom one, so that the lines in the same block have the same indentation. And, the indentation increases in deeper blocks. Curly brackets are aligned so that the pairing is clear. Also, the programmer tries to avoid multiple operations on the same line.

First Look at Methods

Just a reminder, we mentioned that methods are units of code that perform some actions. Here, without going too deep into the details, we will take a look at two frequently-used methods. They are the method *print()* and *println()*.

You might notice that when we talk about methods, there is always a pair of brackets () involved. It is not wrong to say that when you see identifiers with brackets, you can tell right away that they are method names. When invoking a method, we put arguments that are required for the action of that method inside the associated bracket. What to put in there and in which order is defined when the method is made. Right now, we will not pay attention to the making of methods just yet.

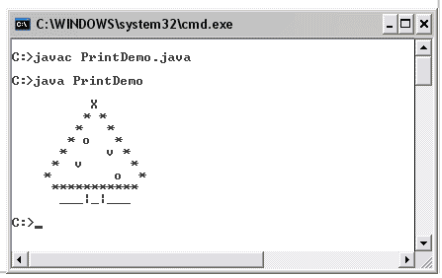
The method print() and println() are methods that print some messages onto the screen. They are defined in a standard Java class called *System*. When we say standard, it means that we can just use them and the compiler will know where to look for definitions of those standard methods. The print() and println() method are invoked by using:

```
System.out.print(<message you want to print>);
System.out.println(<message you want to print>);
```

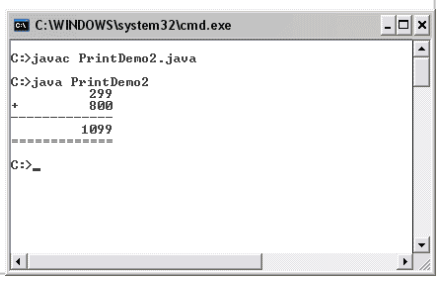
If you want to show a string of character on screen, replace `<message you want to print>` with that string of character enclosed in double quotes. If you replace `<message you want to print>` with a numeric value without double quotes, both methods will show the number associated with that numerical value. The different between the two methods is that method println() makes the screen cursor enter the next line before executing other instructions.

More Examples

```
public class PrintDemo
{
    public static void main(String [] args)
    {
        System.out.println("");
        System.out.println("          X");
        System.out.println("        * *");
        System.out.println("      *   *");
        System.out.println("    * o   *");
        System.out.println("  *     v *");
        System.out.println("*  v      *");
        System.out.println("*        o  *");
        System.out.println(" **********");
        System.out.println("    ___|_|___");
    }
}
```




```
public class PrintDemo2
{
    public static void main(String [] args)
    {
        System.out.print("          ");
        System.out.println(299);
        System.out.println("+         800");
        System.out.println("-------------");
        System.out.print("          ");
        System.out.println(299+800);
        System.out.println("=============");
    }
}
```



In the class PrintDemo2, notice that different types of value are used as inputs for the print() and println() methods. A sequence of characters enclosed by double quotes is called a "*string*". Strings are used as inputs on lines 5, 7, 8, 9, and 11. On lines 6 and 10, numeric values are used as inputs to the println() method.

Escape Sequences

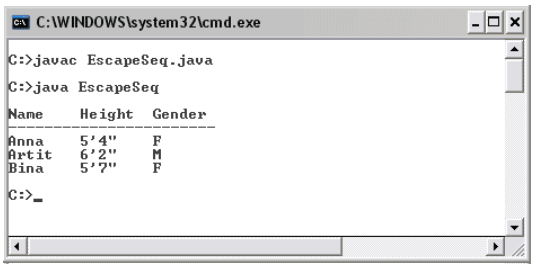
An escape sequence is a special character sequence that represents another character. Each of these special character sequences starts with a backslash, which is followed by another character. Escape sequences are used to represent characters that cannot be used directly in a string (double quoted message). Some escape sequences are listed in the table below.

Escape sequence	Represented character	Escape sequence	Represented character
\b	Backspace	\\	Backslash
\n	Newline	\"	Double quote
\t	Tab	\'	Single quote
\r	Carriage return		

Table 1: Escape sequences

The following Java code shows an example of how escape sequences work.

```
public class EscapeSeq
{
    public static void main(String [] args)
    {
        System.out.print("\n");
        System.out.println("Name\tHeight\tGender");
        System.out.println("----------------------");
        System.out.println("Anna\t5\'4\"\tF");
        System.out.println("Artit\t6\'2\"\tM");
        System.out.println("Bina\t5\'7\"\tF");
    }
}
```



In this example, \n is used for entering a new line and \t is used for inserting tab to format the output.

Variable at a Glance

Variables are symbolic names of memory locations. They are used for storing values used in programs. We will discuss Java types of values and variables later in this course. In many programming languages including Java, before a variable can be used, it has to be declared so that its name is known and proper space in memory is allocated. Syntax used for declaring variables can be observed from the example below.

```
int x;
double y;
String myText;
```

On the first line, a variable `x` is created for storing an *int* (integer) value. On the second line, a variable `y` is created for storing a *double* (double-precision floating point) value. On the last line, a variable `myText` is created for storing a reference to *String* (a Java standard class representing double quoted text) object. The name `x`, `y`, and `myText` are Java identifiers. *int* and *double* are Java keywords. *String* is the name of a Java class.

Variables are normally used with the *assignment operator* (=), which assign the value on the right to the variable on the left. For example:

```
x = 3;
y = 6.5;
myText = "Java Programming";

int z;
z = x;
```

On the first three lines, values are assigned to the three variables according to their types. On the last two lines, a variable `z` is declared and then assigned the value of the variable `x`.

Declaration and value assignment (initialization) could be done in the same expression. For example:

```
int i = 1;
double f = 0.0;
String mySecondText = "Java is fun.";
```

Naming Rules and Styles

There are certain rules for the naming of Java identifiers. Valid Java identifier must be consistent with the following rules.

An identifier cannot be a Java reserve word.

An identifier must begin with an alphabetic letter, underscore (_), or a dollar sign ($).

If there are any characters subsequent to the first one, those characters must be alphabetic letters, digits, underscores (_), or dollar signs ($).

Whitespace cannot be used in a valid identifier.

An identifier must not be longer than 65,535 characters.

Also, there are certain styles that programmers widely used in naming variables, classes and methods in Java. Here are some of them.

Use lowercase letter for the first character of variables' and methods' names.

Use uppercase letter for the first character of class names.

Use meaningful names.

Compound words or short phrases are fine, but use uppercase letter for the first character of the words subsequent to the first. Do not use underscore to separate words.

Use uppercase letter for all characters in a constant. Use underscore to separate words.

Apart from the mentioned cases, always use lowercase letter.

Use verbs for methods' names.

Here are some examples for good Java identifiers.

Variables: `height, speed, filename, tempInCelcius, incomingMsg, textToShow.`

Constant: `SOUND_SPEED, KM_PER_MILE, BLOCK_SIZE.`

Class names: `Account, DictionaryItem, FileUtility, Article.`

Method names: `locate, sortItem, findMinValue, checkForError.`

Not following these styles does not mean breaking the rules, but it is always good to be in style!

Statements and Expressions

An *expression* is a value, a variable, a method, or one of their combinations that can be evaluated to a value. For example, each of the following is an expression.

```
3.857
a + b - 10
8 >= x
p || q
"go"
System.out.print("go")
Math.sqrt(2)
squareRootTwo = Math.sqrt(2)
```

A *statement* is any complete sentence that causes some action to occur. A valid Java statement must end with a semicolon. For example, each of the following is a statement.

```
int k;
int j = 10;
double d1, d2, d3;
k = a + b - 10;
boolean p = ( a >= b );
System.out.print("go");
squareRootTwo = Math.sqrt(2);
```

Each of the following is not a valid Java statement since it does not cause any action to occur or syntactically incorrect.

```
i;
2 <= 3;
1 + 2 + 3 + 4;
"go";
Math.sqrt(2);
```

Simple Calculation

Numeric values can be used in calculation using arithmetic operators, such as add (+), subtract (-), multiply (*), divide (/), and modulo (%). Assignment operator (=) is used to assign the result obtained from the calculation to a variables. Parentheses are used to define the order of the calculation.

The following program computes and prints out the average of the integers from 1 to 10.

```
public class  AverageDemo                                                    1
{                                                                            2
    public static void main(String[] args)                                  3
    {                                                                        4
        double avg, sum;                                                     5
        sum = 1.0+2.0+3.0+4.0+5.0+6.0+7.0+8.0+9.0+10.0;                      6
        avg = sum/10;                                                        7
        System.out.println(avg);                                             8
    }                                                                        9
}                                                                           10
```

More operators will be presented later in the next chapter.

Exercise

1. What are the differences between machine languages and high-level programming languages?
2. Explain why Java is said to be a platform-independent language.
3. Describe the benefit(s) of using proper indentation.
4. Write a complete Java program that shows your first name and last name on screen.
5. Write a complete Java program that shows the pattern similar to the following.

```
*******      *******
      *      *     *
      *******      *******
```

6. Change this source code as little as possible, so that it gets compiled successfully?

```
public class Ex2_6
{
    public static void main(String [] args)

        System.out.println("What's wrong with me?)
    }
}
```

7. Show how to use method print() once in order to print the following message on screen.

```
He called Clancy at headquarters and said:
"There's a family of ducks walkin' down the street!"
```

8. Which of the following identifiers are not valid?

google	$money	company_motto	java.org
12inches	money$	X-Ray	CC
Item3	$$money	!alert_sign	entry point
public	main	Class	_piece
JavaApplet	fillForms	"Gorilla"	_1234

9. List every keywords and identifiers appeared in the class definition of `AverageDemo`.

10. Modify the source code of the class `AverageDemo` so that the program shows both the summation and the average of the integer 1 to 5 as the following.

```
The sum of 1 to 5 is 15.
The average of 1 to 5 is 3.
```