Introduction to Programming using Java

Part 1

C Deighan

Contents

Introduction to Java Programming

Java platform overview

Java technology is used to develop applications for a wide range of environments, from consumer devices to heterogeneous enterprise systems.

Like any programming language, the Java language has its own structure, syntax rules, and programming paradigm. The Java language's programming paradigm is based on the concept of object-oriented programming (OOP), which the language's features support.

The Java language is a C-language derivative, so its syntax rules look much like C's: for example, code blocks are modularized into methods and delimited by braces ({ and }), and variables are declared before they are used.

Structurally, the Java language starts with packages. A package is the Java language's namespace mechanism. Within packages are classes, and within classes are methods, variables, constants, and so on. You'll learn about the parts of the Java language in this booklet.

The Java compiler

When you program for the Java platform, you write source code in .java files (Blue J) and then compile them. The compiler checks your code against the language's syntax rules, then writes out bytecodes in .class files. Bytecodes are standard instructions targeted to run on a Java virtual machine (JVM). In adding this level of abstraction, the Java compiler differs from other language compilers, which write out instructions suitable for the CPU chipset the program will run on.

The JVM

At run time, the JVM reads and interprets .class files and executes the program's instructions on the native hardware platform for which the JVM was written. The JVM interprets the bytecodes just as a CPU would interpret assembly-language instructions. The difference is that the JVM is a piece of software written specifically for a particular platform. The JVM is the heart of the Java language's "write-once, run-anywhere" principle. Your code can run on any chipset for which a suitable JVM implementation is available. JVMs are available for major platforms like Linux and Windows, and subsets of the Java language have been implemented in JVMs for mobile phones and hobbyist chips.

The garbage collector

Rather than forcing you to keep up with memory allocation (or use a third-party library to do this), the Java platform provides memory management out of the box. When your Java application creates an object instance at run time, the JVM automatically allocates memory space for that object from the heap, which is a pool of memory set aside for your program to use. The Java garbage collector runs in the background, keeping track of which objects the application no longer needs and reclaiming memory from them. This approach to memory handling is called implicit memory management because it doesn't require you to write any memory-handling code. Garbage collection is one of the essential features of Java platform performance.

The Java Development Kit

When you download a Java Development Kit (JDK), you get — in addition to the compiler and other tools — a complete class library of prebuilt utilities that help you accomplish just about any task common to application development. The best way to get an idea of the scope of the JDK packages and libraries is to check out the JDK API documentation (see Resources).

A Java Class

A Java Class is a compiled program (.java file) which has been converted to a different program type (.class file) that the JDK reads and runs. The most basic Java class is composed of the following:

```
public class FirstClass
{
        public static void main(String[] args)
        {
          //Main method code goes here
        }
}
```

As you can see, in the basic program above we have a class statement and a main method. Furthermore, you may have noticed the curly brackets, when creating a class/method you must show where it starts and ends with the curly brackets ({ }) inside these brackets is where the code goes. As you can see, the main method is within the class brackets.

[REMEMBER] -- The first letter for a class name should always be capitalised.

Main Method (Routine)

The main method is used in every single program which you want to execute the instructions (code). This is where the compiler will start to read and perform the actions written in your program.

A Java Method (Sub-routine/Procedure/Function) see pages 5 and 6, is a collection of statements that are grouped together to perform an operation. When you call the print method (System.out.print), for example the system actually executes several statements in order to display message on the console.

Data Types and Variables

Programs manipulate data that are stored in memory. In machine language, data can only be referred to by giving the numerical address of the location in memory where the data is stored. In a high-level language such as Java, names are used instead of numbers to refer to data. It is the job of the computer to keep track of where in memory the data is actually stored; the programmer only has to remember the name. A name used in this way -- to refer to data stored in memory -- is called a variable.

Variables are actually rather subtle. Properly speaking, a variable is not a name for the data itself but for a location in memory that can hold data. You should think of a variable as a container or box where you can store data that you will need to use later. The variable refers directly to the box and only indirectly to the data in the box. Since the data in the box can change, a variable can refer to different data values at different times during the execution of the program, but it always refers to the same box.

For example:

Memory Location	Data Type	Identifier	Value
0X123	String	nameOfPerson	"John"
0X124	Int	ageOfPerson	67
0X125	Double	weightOfPerson	110.23

Whenever the program is executing the values can change.

Creating and assigning data to a variable.

It is worth noting here, depending on the variables data type it will dictate what can be done with it.

When creating variables you can create them any way of the following ways stated in 1 & 2, or 3.

1) int x;
2) x = 5;
3) int x = 5;

However it is always good practice to declare and initialise (give a starting value) a variable as it makes the program logically correct.

Further variable creation

```
int x                   // declares an integer variable x
x = 3;                  // assigns the value 3 to x
double y = 0.3;         // declares a floating point/decimal variable y and assign the value 0.3 to y
boolean b;              // declares a boolean variable b. Boolean will be true or false
b = (x < y);            // b will be assigned false, because the expression x < y evaluates to false
String s = "Hello"      //declares a String variable s and assign the value Hello to s
```

Creating Methods (Sub-routines, Procedures, Functions)

```
public static void myProcedure
{
     //the above procedure is made up of four parts
     //1 - public –visibility when creating instances - it can also be called private or protected
    // 2 – static – Means that it can be "seen" by the computer before compiling
    //3 -  void the return type, in the case procedures don't return anything, so its void
   //4 –  myProcedure the identifier - always call it something  meaningful
}
```

```
public static int myIntFunction
{
     //the above function is made up of four parts
     //1   public – it can also be called private or protected
    // 2   static – Means that it can be "seen" by the computer before compiling
    //3    int – in this case were returning an int. The return type can be any type of variable
   //4     myIntFunction – the identifier – always call it something meaningful

}
```

NOTE: The last line, before the closing curly bracket, you must use the keyword "return" to return a specific variable, the variable being returned must be the same type as declared in the function. For example above we have created an int function, we must return an int.

Creating Methods (Sub-routines, Procedures, Functions) with Parameters

A **parameter** is an additional item of data which is given to a procedure or function. It is given a variable type and name when procedure or function is defined. Parameters are optional when creating sub-routines. When a method is invoked (called), you pass a value to the parameter. This value is referred the **argument**.

```
public static String  myStringFunction(String myNumber)
{
     //the above function is made up of five parts
     //1   public – it can also be called private or protected
     // 2   static – Means that it can be "seen" by the computer before compiling
     //3    String – in this case were returning a String. The return type can be any type of variable
    //4    myStringFunction – the identifier – always call it something meaningful
    //5    String myNumber – The parameter, the additional piece of data the function needs to run
}
```

NOTE: Although we have created a String Function here, the parameter does not need to be a String too. We can change it to int or double. The parameter type will be depended upon the requirements of the function.

```
public static void  myStringProcedure(String myNumber)
{
     //the above procedure is made up of five parts
     //1 - public –visibility when creating instances - it can also be called private or protected
     // 2 – static – Means that it can be "seen" by the computer before compiling
     //3 -  void the return type, in the case procedures don't return anything, so its void
    //4 –  myStringProcedure the identifier - always call it something  meaningful
    //5    String myNumber – The parameter, the additional piece of data the function needs to run
}
```

NOTE: Although we have created a String Procedure here, the parameter does not need to be a String too. We can change it to int or double. The parameter type will be depended upon the requirements of the procedure.

Calling the sub-routine

We invoke (call) the sub-routine (function/procedure) by typing their name followed by **();** this will then execute the statement inside the sub-routine and then give control back to the main program whenever it has finished.

myProcedure(); //from page 5

If you want to invoke a sub-routine that needs a parameter, you type their name followed by **();** however this time, inside the brackets you need to pass your argument, For example from above, **myStringProcedure("12345");** //from page 6

Commenting your code

You have probably noticed that alongside some of the code above I have I have inserted "//" and then a description. This is called a comment. A comment is a line in the code for the programmer to read and remind him/her about what the line of code is what the programmer needs to do, etc. Comments are not read by the compiler thus, they can be anything.

How to create a comment: There are 2 types of comments in Java, the single line comment and the multi-line comment. To create a single line comment you start with // and then put anything you want. A multi-line comment is a bit more complex than the single line comment. A multiline comment looks like this:

/* start of comment

* all lines except for start and end

* should start like this

end of comment is like this */

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

         //Single line comment!
         /*This is the first line
         *second line
         *third line
         end*/

       }
}
```

The Hello World program is a generic program in all languages, it is a simple console-print out program that says "Hello World!".

```
public class FirstClass
{

	public static void main(String[] args)
	{
	System.out.print("**********\n");
	System.out.print("* *\n");
	System.out.print("* *\n");
	System.out.print("* Hello World! *\n");
	System.out.print("* *\n");
	System.out.print("* *\n");
	System.out.print("**********\n");
	System.out.println("**********");
	System.out.println("* *");
	System.out.println("* *");
	System.out.println("* Goodbye World *");
	System.out.println("* *");
	System.out.println("* *");
	System.out.println("**********");
	}

}
```

As you can see I used two different types of print out statements:

System.out.print -- This prints out whatever is in the quotations in a single line.
System.out.println -- This prints out whatever is in the quotations in a new line.

Also, in all the print lines you may see '\n' this is called a special key.

Java Special Keys

\n - new line
\b – backspace
\s – space
\t – tab
\" - Double quote
\' - single quote

Getting User Input

The Java Developer's Kit (JDK) comes with a huge amount of built-in Java code that can be used to build applications. However, for a Java class to use this code it must identify the package that contains that code. This is done at the top of the source file using the import keyword. Example:

import java.util.*; // gain access to all the code in the java.util package (* means all)

import java.util.Scanner; // gain access to the Scanner library

import java.util.Random; //gain access to the Random library

In Java, user input is done using 'Scanners' A scanner is an object that takes anything the user inputs and sets it as a string (unless an int is specifically defined):

```
import java.util.Scanner;

public class UserInput
{
        public static void main(String[] args)
        {
            Scanner kb = new Scanner(System.in);
            String text;
            System.out.print("Enter some text: ");
            text = scan.next();
           System.out.println(text);
       }
}
```

Let's have a look at this code piece by piece:

Scanner kb = new Scanner(System.in);

This creates a new Scanner object called 'scan' and uses the System's console (System) as the input device (.in).

text = scan.next();

This sets the previously created string named 'text' to the next string inputted by the scanner.

Along with .next() which means next thing inputted, the scanner also has:

.nextLine() -- looks at the line of input after the current line

.nextInt() -- reads the next integer inputted

.nextDouble() -- reads the next double inputted

User Input continued….

Scanner kb = new Scanner(System.in); //kb now becomes a new scanner class

As we have created a new Scanner class called "kb", we have access to all the Scanner Class methods by using its new name. Therefore to read an integer from the user we would use the "nextInt()" method as below.

User Input – Integers

```
System.out.println("Please enter a number");     //Asking the user to input a number

int number = kb.nextInt();          //this line is assigning the number entered to the
                                    //variable  number
```

To print the value that number refs to.

System.out.println("The number entered was " + number); // + means to concatenate two items.

User Input – Strings

To read a String from the user, we use the "next()" method as below

```
System.out.println("Please enter your name");
String name = kb.next();
System.out.println("Hello " + name);
```

Selection Statements

An if statement is a conditional statement that says:

```
if<conditional>
{
codeblock  will happen
}
```

There are three 'types' of if statements:

1)
```
if <conditional>
{
        code block will  happen
}
```

2)
```
If <conditional>
{
        code block will happen
}
else if<condition>
{
        code block will happen if does not apply
}
```

3)
```
If <conditional>
{
        code block will happen
}
else if<condition>
{
        code block will happen if above does not apply
}
else
{
        code block will happen if above does not apply
}
```

Example of Selection

A good example of if statements would be:

```
public class Selection
{
        public static void main(String[] args)
        {
                int x = 10;
                If (x > 10)
                {
                        System.out.print("X is greater than 10.");
                }

                else if (x < 10)
                {
                        System.out.print("X is less than 10.");
                }
                else
                {
                        System.out.print("X is equal to 10.");
                }
        }
}
```

If you take the above code and put it into basic sentences, it will look like:

The integer 'x' is equal to 10.

If x is greater than 10, print out "X is greater than 10." Otherwise, if x is less than 10, print out "X is less than 10." If neither of the above statements applies to x, print out "X is equal to 10."

Selection continued….

If statements can be used with user input. For example:

```
import java.util.Scanner;
public class UserInput
{
        public static void main(String[] args)
        {
            Scanner kb = new Scanner(System.in);
            String password = “xyxy”;
            String text;
            System.out.print("Enter A Password: ");
            text = kb.next();
            if(text != password)
            {
                System.out.print(“Access denied.”);
            }
            else
            {
                System.out.print(“Access granted.”);
            }
    }
}
```

The above program asks the user for a password, if the user enters “xyxy”, access is granted. However, if the user does not enter “xyxy”, access is denied.

Comparison in Selection statements

Comparing integers the following applies:

!= - means not equal to
== - means exactly equal to

Comparing String the following applies:

!= - means not equal to
.equals – means exactly equals to.

Example with Strings

```
if(text.equals(“xyxy”)
{
    code block
}
else
{
   code block
}
```

CASE/SELECT – Switch Statements

A Switch statement in java is like an 'if statement' but more organised, efficient and can be added to more easily.

Example:

```
public class switch
{
        public static void main(String[] args)
        {
            int day = 5;
            switch (day)
            {
                case 1:
                System.out.println("Monday");
                break;

                case 2:
                System.out.println("Tuesday");
                break;

                case 3:
                System.out.println("Wednesday");
                break;

                case 4:
                System.out.println("Thursday");
                break;

                case 5:
                System.out.println("Friday");
                break;

                case 6:
                System.out.println("Saturday");
                break;

                case 7:
                System.out.println("Sunday");
                break;

                default:
                System.out.println("Invalid Day.");
                break;

    }
}
```

The above code will print out Friday. CASE/SELECT is similar to "IF Statements", but seen as more efficient, easier to maintain and easier to add additional options for menus system.

Iteration

In java there are loops that allow the programmer to have a block of code repeated while a conditional is true/false. There are three main types of loops, these are:

While loop:

The while loop says

```
while <conditional>
{
    code block will happen
}
```

The while loops is useful for long and fast loops. One example of this type of loop is:

```
public class loops
{
        public static void main(String[] args)
        {
                int x = 10;
                while(x != 0)
                {
                        System.out.println("X = " + x);
                         x--;
                }

         }
}
```

For loop:

The for is like a while loop, but it is used when the number of iterations are known before the program executes. To write this type of loop we use:

How to structure loops in Java Code:

```
for (conditional;    conditional >/</<=/>=/;    conditional++/conditional--).
```

One example of this type of loops is:

```
public class loops
{
        public static void main(String[] args)
        {
           for (int i = 10; i > 0; i--)
        {
           System.out.println("i = " + i);
     }
  }

}
```

FOR LOOP

The above FOR LOOP below is going to execute 10 times.
int i = 10; is your index/counter
i > 0; is how many times the loop will run and when the loop will terminate
i--; is decrementing your index/counter

Do-while loop:

The do while loop is used when we don't know the number of iterations required AND when we want the loop to execute at least once.
To write this type of loop we use:

```
do
{
code block
}
while<conditional>.
```

One example of this loop is:

```
public class loops
{
        public static void main(String[] args)
        {
                int x = 10;
                do
                {
                        System.out.println("X =" + x);
                        x--;
                }
                while(x != 0);
        }
}
```

As you can see, each of these loops is created differently but they are all able to do the same thing. It is up to the programmer which one he/she wants to use.