Introduction to Java^{TM}

March 31, 2000

Overview

Java is an object oriented programming language developed and owned by Sun Microsystem. Originally it was meant to be a common language for integrated devices, e.g, in VCRs, cappuccino machines, ...

After its introduction in 1995 it soon became very popular for programming web applets and later for general applications due to its "Write once, run everywhere" promise.

Contrary to C/C++ and FORTRAN programs Java is not compiled into machine code for a particular processor, but runs within the Java Virtual Machine (JVM). This makes it independent of the specific architecture and therefore possible to run the same program on different machines.

The payoff is that optimization has to be done by the JVM during runtime.

1 Program Layout

Just like C/C++ Java is a free form language, i.e. there is no specific layout that is enforced by the compiler, however there are conventions for indentions that should be followed. In general, everytime you have an opening curly bracket the next line of code is indented further than the previous one. Closing curly brackets are usually placed at the same level as the line that opened the block. Here is an example:

```
/* HelloWorld
 *
 * v1.0
 *
 * Jan H. Meinke
 */


/** HelloWorld prints out either 'Hello World!' or if called with an argument
 *  'Hello <i>Name<\i>'
 */
public class HelloWorld{
    public static void main(String[] argv){
        if (argv.length>0){
            System.out.println("Hello "+argv[0]);
        }
        else{
            System.out.println("Hello World!");
        }
    }
}
```

Also notice that every statement ends with a semicolon.

1.1 Comments

Java provides the same comments as c++, i.e., \\ for a single line comment and *... *\ for multiline comments. In addition a variation of the multiline comment is used to provide a mechanism for automatic documentation creation: **... *\.

This last kind of comment is read by the program javadoc that comes with the Java Development Kit (JDK) and is used to automatically generate the documentation of the Application Programming Interface (API) of your program.

2 Compiling and Running a Java Program

To write and run Java programs a few tools are needed:

1. A text editor to write the source code
2. The Java Development Kit (JDK) that provides the tools and libraries needed to compile and run the program.

2.1 Writing the source code

The source code for a Java program that contains a top level public class must have the name of that class - case sensitive - and the extension *.java*.

2.2 Compiling and Running the Program

If the above condition is fulfilled the program can be compiled using: javac <ClassName.java> where class name is replaced by the actual name of your class. To compile the above example you would use: *javac HelloWorld.java*

This will produce a file called HelloWorld.class. This is equivalent to the executable of a c program. To run it type: *java HelloWorld*[1] Notice that there is no extension given here!

The program will write *Hello World!* to the screen. Now run the program by typing: *java HelloWorld <your name>*.

If you took me a little bit too literal the output will now be: *Hello <your.* Replace <your name> with your first name.

3 Syntax

3.1 Primitive Data Types

Java provides the following primitive data types: boolean, char, byte, short, int, long, float and double. Everything else is implemented as classes. There are also class wrappers available for each of the primitive data types. Contrary to C/C++ the sizes are well defined. (See Table)

Type	Contains	Default	Size	Min Value	Max Value
boolean	true or false	false	1 bit	N.A	N.A
char	Unicode character	\u0000	16 bits	\u0000	\uFFFF
byte	signed integer	0	8 bits	-128	127
short	signed integer	0	16 bits	-32768	32767
int	signed integer	0	32 bits	-2147483648	2147483647
long	signed integer	0	64 bits	-9223372036854775808	9223372036854775807
float	IEEE 754 floating-point	0.0	32 bits	$\pm 3.40282347E+38$	$\pm 1.40239846E-45$
double	IEEE 754 floating point	0.0	64 bits	$\pm 1.797693148623157 0E+308$	$\pm 4.94065645841246544E-324$

[1] There are other programs that will compile or run a Java program but I will stick with the standard tools provided by Sun.

3.2 Classes and Objects

A the heart of Object Oriented Programming are object, but if you look at the example program there is not a single command object only a class.

Classes are the blueprint for objects. They contain information about the variables and the behavior of an object, but just like you cannot live in the blueprint of a house you cannot perform a programming task using a class.[2]

3.3 Declarations

A public class should be declared in a file with the same name plus the ending `.java`. Alternatively a public class can be declared static if it is contained within the source code of another class. This naming scheme is used by the run time class loader to find and load the appropriate class.

A typical class declaration looks like this:

```
public class ANewClass extends AnExistingClass implements AnInterface{
⋮
}
```

Here the bold words are Java key words. All but the class key word are optional.

3.4 Variables

3.4.1 Variable Declaration

Before a variable can be used it needs to be declared and initialized. Declaring a variable defines its accessibility, type, name and possibly its value, e.g.,

```
public static final double PI=3.1415927;
```

The first three parts are modifiers, the fourth one defines the variable type followed by the name and finally the initial (and in this particular case final) value of the variable. The full syntax for a one variable declaration is the following:

[{public|protected|private}] [static] [final] <*Variable type*> <*Variable name*> [*Variable initialization*]

The following table summarizes the meaning of each part of the declaration:

	Meaning
public	Accessible anywhere its class is.
protected	Accessible in its package, class and subclasses
private	Accessible only in its class.
none	Accessible within the package
static	Static variables can be accessed without first creating an object of there class. This is useful for constants (see final modifier) and other variables all instances of a class need to keep synchronized.
final	Like const in C/C++. It disallows changes after initialization. Variables that are declared final must be initialized at declaration.
<Variable type>	Can be any primitive data type or class!
<Variable name>	Can be any combination of letters including $ and _ (underscore), although these should be avoided as first symbol. (See 3.4.2 for more details.
<Variable initialization>	Any command that leads to a value of the correct type, e.g, `private Currency myCurrency=Currency.getLocalCurrency();` is allowed if Currency.getLocalCurrency() returns a Currency. (If it does not, the name for the method was chosen very badly.)

[2]This is not quite correct. Static methods and variables of a class can be used without creating an object from this class and there are situations when this comes in very handy.

3.4.2 Variable Names

Variable names should be short and descriptive. They should start with a lower case letter avoiding '$' or '_' and every internal word starts with a capital letter. Avoid one letter variable names except for "throw away" variable, e.g., in loops. Here are some examples for variable name:

```
int i;
int myAge;
JFrame mainApplicationFrame;
```

Constants should be all uppercase with words separated by underscores.

```
static final double MAX_TOLERANCE=0.1;
```

3.5 Control Statements

Every programming language needs to control the flow of execution. Java provides the same control mechanisms as C++.

3.5.1 do-while Loop

Repeat the enclosed statement until expression is true, but at least once.

do ... while(expression) statement

3.5.2 for Loop

The for loop is used for the repeated execution of a program block. After performing initialization repeat the loop until expression is true and perform statement2 before statement1.

for(initialization , expression, statement1) statement2

3.5.3 while Loop

Repeat the enclosed statement while expression is true.

while(expression) statement

3.5.4 switch Statements

The switch statement allows to execute different statements depending on the value of expression. It also can include a default statement that gets executed if expression doesn't match any of the given cases.

switch(int expression){
case V1: statement1
break;
case V2: statement2
break
default: statementD;
}

3.5.5 if Statements

The if-then-else constructs executes different statement depending on boolean expressions.

if(expression) then {
statement
}
else{
statement
}

3.5.6 Jump Statements

Sometimes it is useful to skip the rest remainder of a statement in a loop and either to break out of it or continue it at the beginning. Java provides the break and continue statement for this.

break will jump to the first line after the current iteration.

continue will skip the rest of the current iteration and continue with the next iteration.