Chapter 1: Java programming language

Introduction

In this lesson, you will learn

1. what makes Java such a hit with programmers,
2. what the components in the world of Java are,
3. how you build a Java application by going through the "Edit, Compile, Execute" process.

Why Java?

Java is a programming language built for the age of the Internet. It was built for a world in which everything that has some sort of electronic component: stereo systems, wireless phones, cars, even your refrigerator, are all on the Internet. This world is right around the corner.

What is so unique about Java that has propelled its rapid, wide acceptance?

- It's available on more devices world-wide than any other language. Notice that I say devices - not just computers. Java is currently being used not only on mainframe systems in the enterprise and personal computers in the office and at home - it's also running in cellphones.
- It was carefully designed to eliminate many of the most common causes of programming errors - bugs. Java programs that compile error-free tend to work! Strong data typing and complete memory management are two features that make this possible.
- It provides for secure programs that can be executed on the Internet without worry of them infecting your system with some virus or planting a trojan horse.

What is a Java Program and How do I Create One?

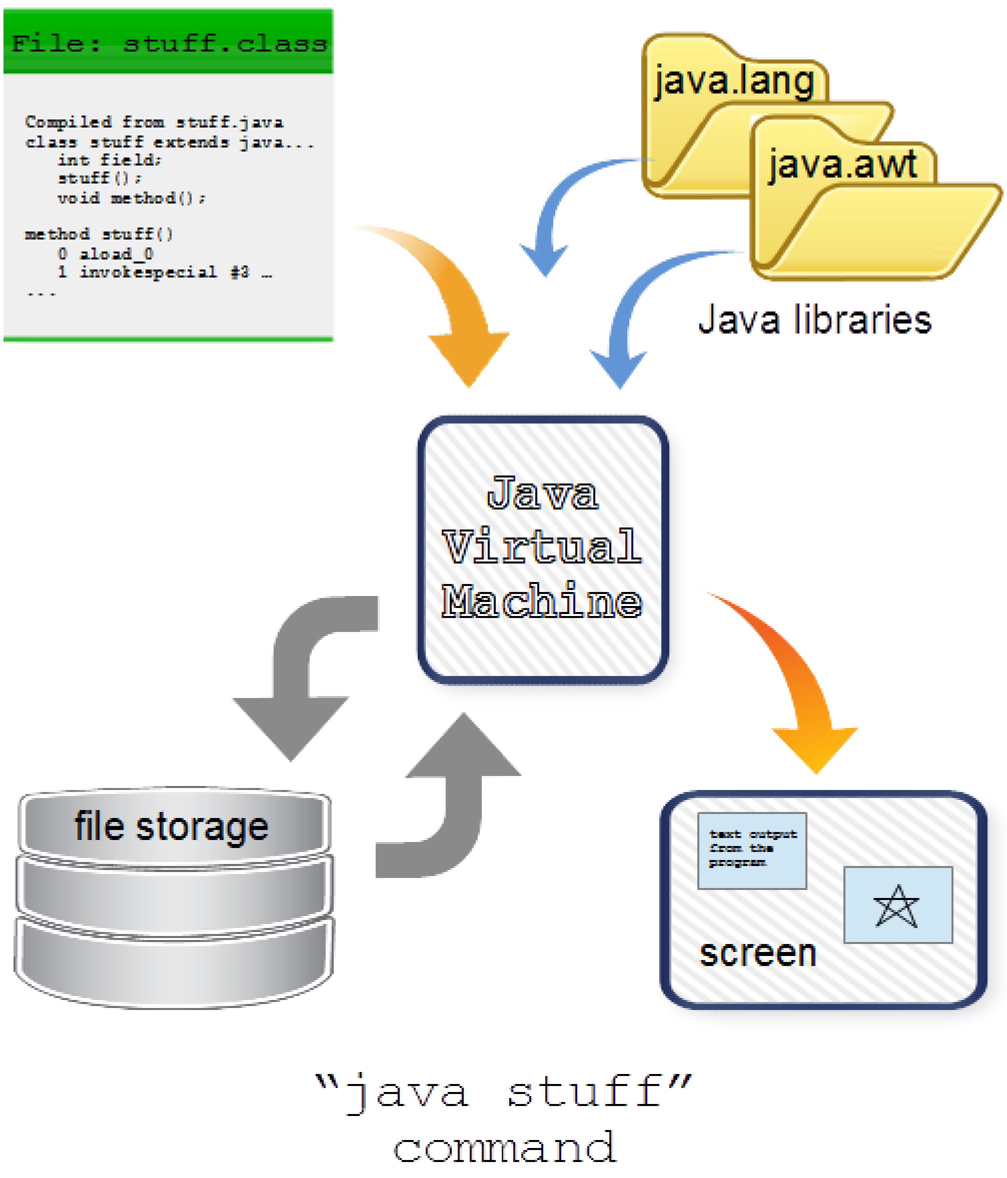
Let's look at what makes up a Java program. A Java program is built by writing (and referencing already available) things called *classes*. In the simplest sense, a Java program is a bunch of classes. You will construct at least one, typing its source code into a file.

The stuff you will enter (text) has a very specific structure (its syntax) that the Java compiler expects. You create your Java programming language files with an editor that is available on your computer. On a PC running Windows, Wordpad or Notepad will work just fine. On a Sun workstation, textedit is a nice editor.

Once you have some complete Java source code in a file, you compile it. The Java compiler turns your file full of characters into another file which contains instructions that a JVM (Java Virtual Machine) can interpret, a ".class" file.

Figure 20.1

The Java Virtual Machine takes over from here. JVMs exist for any computer and operating system, for example PCs running Windows, Sun Microsystems' computers running Solaris or Linux, cellphones, etc... The JVM takes your ".class" file, loads it into its virtual memory, links lots of stuff together, and then starts interpreting/executing the program. During linking, your class file will be combined with other classes that are part of the Java environment, e.g., java.awt.Canvas, java.lang.String, etc... Standard classes exist for helping you do things like displaying text on the screen, get characters typed on the keyboard, read/write files, display graphics stuff, communicate over the Internet, ... And, then, off it goes; your program comes to life.



```
Compiled from stuff.java
class stuff extends java...
    int field;
    stuff();
    void method();

method stuff()
    0 aload_0
    1 invokespecial #3 …
...
```

Figure 20.2

A Java Application, Our First!

There are two kinds of Java programs: applets and applications. You will start with a very simple application.

To get going you need to start by working your way through what's known as the "Edit, Compile, Execute" process or cycle. This is something you will tend to do over and over as you work on your Java programs. This exercise will give you a feel for how you will be working in the Java environment. To make this quick and simple, you'll start with what is the shortest Java program possible.

Type the following text into your computer, putting it in a file called "Hello.java"

```
class Hello
{
   public static void main(String[] args)
   {
     System.out.println("Hello World!");
   }
 } // end class Hello
```

Once you have it in a file in the computer, check it to see that it matches the example - *character for character*, upper-case only where the example shows upper-case characters, the same punctuation, etc... When you think you have it, make sure to save the text to the file: **Hello.java**.

Now it's time to run the Java compiler with the file "Hello.java" as input. This will get you your "Hello.class" file. Type in:

```
  javac Hello.java
```

Did it complain? Did it find a typo that you missed? If so, go back into the editor and compare your text with what I've provided. The compiler tells you the line number that it detected an error on. This should help. But, the error can be on an earlier line too. Repeat this "Edit, Compile" cycle until you get the compiler to quit complaining.

Now you can use the JVM to execute your program. Type in:

```
  java Hello
```

Notice that you don't type "java Hello.class" even though you had to provide the full filename to the Java compiler. The Java Virtual Machine (JVM) assumes and looks for the file Hello.class even though you only provided the class name Hello, not the filename. If you have everything right, the system should respond with:

```
  Hello World!
```

Cool... You've just entered and executed your first Java application.

You've used the **println** command in your jLogo programs too. I added this command in preparation for your move to Java. In standard Logo, the **print** command does what jLogo's **println** does. But, in Java, **print** does not add the *newline* character to the output - just like the way **print** works in jLogo. In Berkeley Logo, the **type** command leaves off the *newline*.

jLogo Command	**Logo Command**	**Java Method**	**Description**
print	type	print	Display the text provided as its input/argument.
println	print	println	Display the text provided as its input/argument, followed by a newline character.

Table 20.1

Play around, try both methods.

Tips

One of the most common initial mistakes is mixing character case. In Java, case matters. The word someThing is not the same as the word something.

A class' name, must match the name of the file that it is in. The Java source code will compile without any indication of problems. But, when you try to execute it, the JVM will complain that it can't find your class, the name of which is the file name.

Finally, notice that the java compiler command, **javac**, expects a full filename including the ".java" part; but, the **java** command will not accept the full filename - it expects to find a file with the name you specify, ending in ".class"