

Java Applets

Interactive Programming

by Elizabeth Sugar Boese

Copyright Year: 2006
Copyright Notice: by Elizabeth Sugar Boese. All rights reserved.



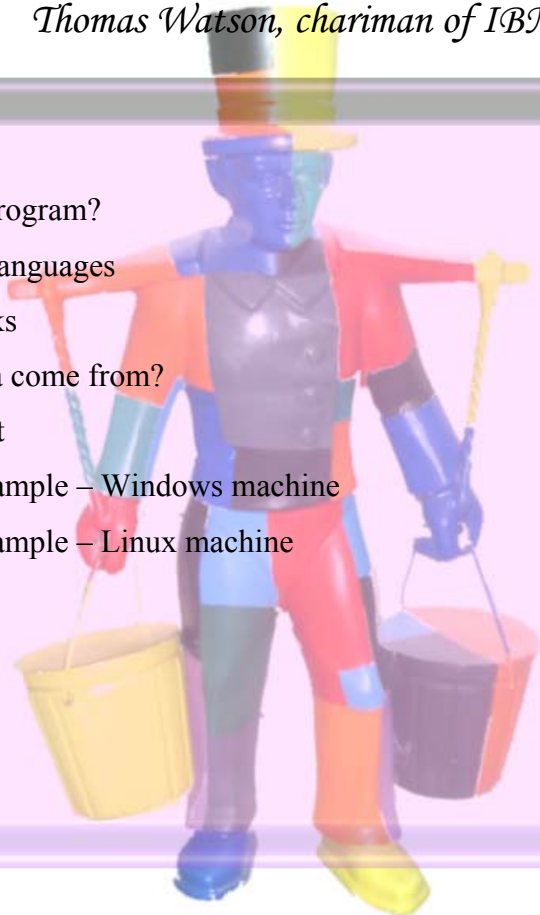
Chapter 1

Introduction to Programming

*"I think there is a world market for maybe five computers."
- Thomas Watson, chairman of IBM, 1943*

Objectives:

- ✦ Why learn to program?
- ✦ Programming languages
- ✦ How Java works
- ✦ Where did Java come from?
- ✦ The Java applet
- ✦ Running an example – Windows machine
- ✦ Running an example – Linux machine



Why learn to program?

We have all learned the basics of computation, such as: $2 + 2 = 4$. We learned how to compute more complex mathematical expressions, such as: $5 * (3 + 7) / 2$. We also learned how to describe a calculation, such as: *the area is the width multiplied by the height*.

Learning how to program is learning how to solve problems. First you have to figure out what the problem is, then design a solution, implement, test and fix errors. This takes a lot of practice and patience. Sometimes we'll get error messages that tell us explicitly what the problem is, and sometimes we won't get any error messages at all – and it doesn't work! This process can be somewhat frustrating, but also can be very rewarding when you figure out the problem.

Programming helps you develop critical thinking skills and problem solving. Problem solving is important for all disciplines and especially useful when employed. By learning how to program, you learn how to formulate problems, think creatively about solutions and express a solution clearly and accurately. These skills are important for all fields; for example

- ✦ Mathematics - use of formal languages, calculation of formulae
- ✦ Engineering - designing, assembling components, evaluating between solutions
- ✦ Natural Science - observing behavior of humans/animals/plants/weather
 - forming hypotheses and testing
 - analyzing results from experiments
- ✦ Art - digital image/video/sound/movie manipulation
- ✦ History - recording, storing, indexing, searching, analyzing and retrieving
 - historical documents and information
- ✦ All fields - process of discovering something new
 - exploration of data and analysis

Programming Languages

When working with computers, computers cannot understand the complexity of the English language. Instead, we need to write a computer program. A **program** is a set of instructions for the computer to execute. Programs are written in a programming language. A **programming language** is a grammar to designate information and instructions that a computer understands. Programming languages are a lot less descriptive than English, but they do use English words (aren't you glad you speak English?). The grammar is also referred to as the **syntax**.

Types of Programming Languages

High-level language: `a = b + 9;`

Assembly language:

```
lw $2, b
lw $3, 9
add $2, $2, $3
sw $2, a
```

Machine language:

```
00010110 01000010 10000010
00010110 01000011 00001001
00010101 01000010 01000010 01000011
00011000 01000010 10000001
```

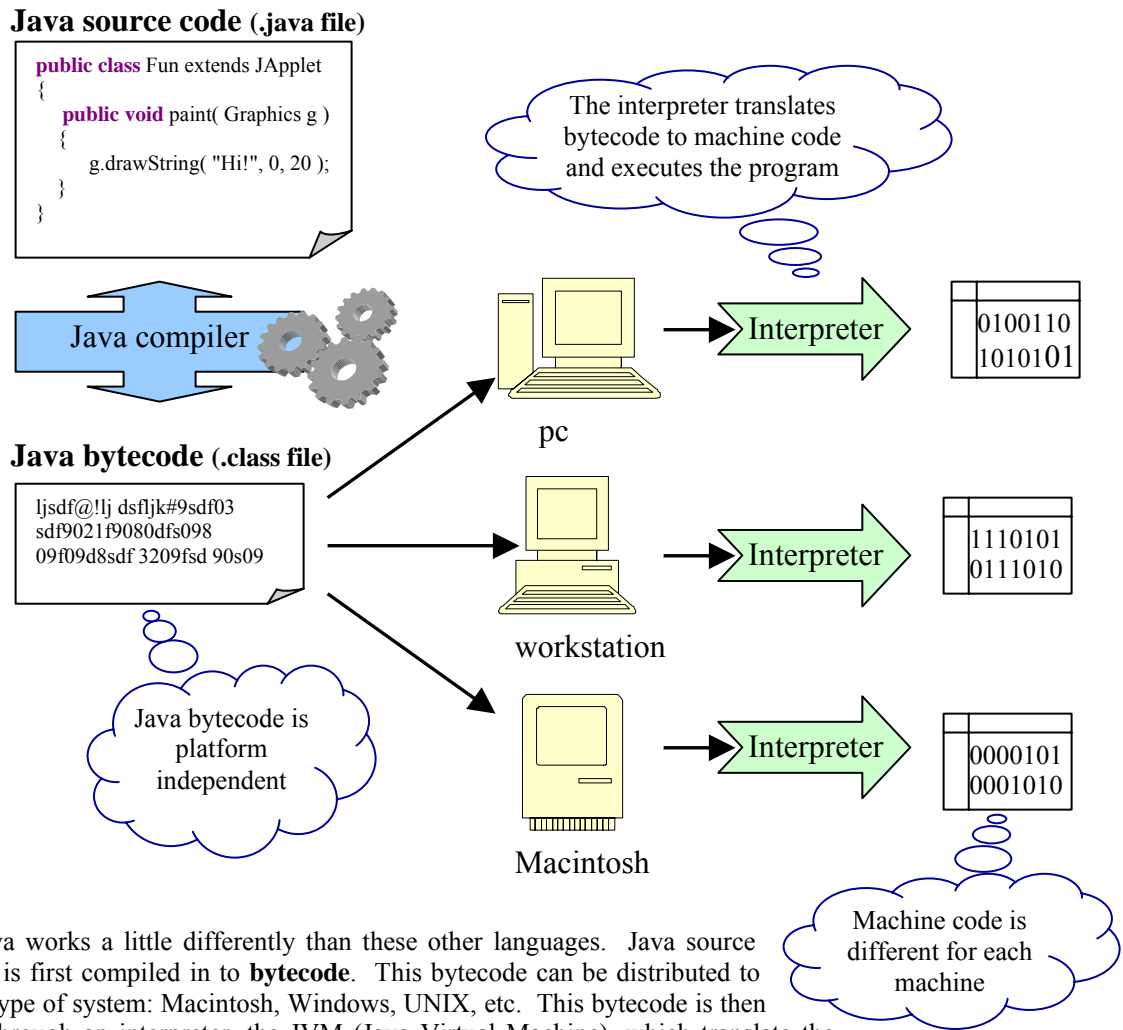
lw \$2, b means load variable **b** in to register 2
add \$2, \$2, \$3 adds values in registers 2 and 3 stores in register 2
sw \$2, a takes the result from register 2 and stores in to variable **a** in memory

Direct 1-to-1 mapping from assembly to machine code.

There are three main types of languages that we're going to look at: machine, assembly, and high-level languages. **Machine code** is the lowest level of programming languages; it is the only encoding that the computer understands – 0's and 1's. Computers today store all information in a binary representation – as groups of 0's and 1's. Machine code differs for different machine types: the machine code to add two values together will be different groups of 0's and 1's on a Macintosh system vs. a Windows system vs. a UNIX system. **Assembly language** is a mapping of the machine code to something more readable. "lw \$2, b" will load the value in a variable named b in to register 2. **High-level language** allows us to simply the program code with a higher level of abstraction than assembly code. An example of high-level code would be: "x = y + 2;". Java is an example of a high-level language.

When we write in a high-level language, we still need to decompose it in to machine code before the computer can execute its instructions. To do this, we take the **source code** which is the program that we wrote and either run it through a compiler or an interpreter. A **compiler** translates source code into a target language. Sometimes compilers translate source code into machine code, and sometimes it translates to another type of code. Programming languages like C and C++ are run through a compiler which translate the source code in to machine code. This is important because it means the compiled C program can only be executed on machine types that it was compiled on. Other languages like HTML and JavaScript are not compiled but interpreted. The source code for these programs are run through an interpreter. An **interpreter** translates the code into machine code and then executes the machine code.

How Java Works



Java works a little differently than these other languages. Java source code is first compiled in to **bytecode**. This bytecode can be distributed to any type of system: Macintosh, Windows, UNIX, etc. This bytecode is then run through an interpreter, the JVM (Java Virtual Machine), which translate the

bytecode in to machine code for the particular machine for it to run. This is where the buzzwords "platform independent" and "write once, run anywhere" concepts are derived.

There are two important catch-phrases associated with Java:

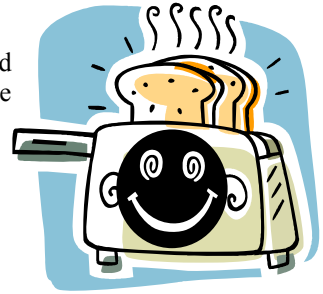
“platform independent” and **“write once, run anywhere”**.

It is important to understand these phrases when talking about Java. Java is platform independent because the source code is compiled to bytecode, and it's the bytecode that can be used on any platform. This works because each platform has an interpreter that can translate the bytecode to the machine code for that particular platform. So platform independence is assuming the use of an interpreter, but the source code doesn't need to be re-compiled on these machines. This leads into the phrase, "write once, run anywhere". Java bytecode works on any platform. Other languages require the program to be modified so it can be compiled for each platform.

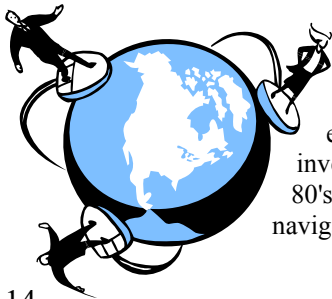
The advantages to the use of interpreters is that you don't need a specific compiler for each machine, the code is platform-independent allowing the code to be run on different machines. However, there are some disadvantages to using interpreters. Code is slower to execute, between 10 to 100 times slower than code compiled straight to machine code. It still requires an interpreter to be on the machine in order to execute the code. And it also limits the abilities that could be programmed, since the code should be able to execute on all types of machines. Therefore, a special machine such as SGIs which are known for extra graphics abilities would not be exploited when programming in an interpreted language.

Where did Java come from?

Java was originally intended to make smart appliances, such as toasters and TVs. The idea was to create a small language that is easy to learn (unlike machine code) that could run on any kind of computer chip. This would enable a manufacturer of a smart appliance to upgrade the computer chip without having to re-do the software to run it. However, Java never took off for electronic appliances, but then along came the World-Wide-Web.



The Internet and World-Wide-Web



A common misconception is that the World-Wide-Web (WWW) is the Internet, but in fact the Internet was around long before the WWW came in to existence. In 1969, the US Department of Defense connect four universities together to form the first Internet. They wanted to make it easier to share information between researchers. By 1970, e-mail was invented to communicate between people via computers on the Internet. In the 80's and early 90's, programs such as gopher and WAIS were developed to navigate text files on the Internet.

There are two different types of Java programs that we can create: applets and applications. Applets are Java programs embedded in a web page. Applications are stand-alone programs that can be run by themselves. In this course, we'll be writing applets so that we can display our applets on the Internet. In order to create an applet, there are three different files that we will work with: the Java program (also called source code) which is what we will write, the bytecode which is what the output from the compiler, and the HTML file to display the applet. The source code is in a file with a **.java** file extension. The bytecode is in a file with a **.class** file extension. HTML files end with either **.htm** or **.html** file extensions.

The Java Applet

Java programs come in two forms: applications and applets. Applications run on their own, similar to MS Word or the Netscape browser. Applets run inside a web page. Our focus is on applets, so let's look at a simple example of an applet.

```
import javax.swing.*;
import java.awt.*;

/** first program
 * @author E.S.Boese
 * @version fall 2005
 */
public class MyFirstProgram extends JApplet
{
    public void paint( Graphics g )
    {
        super.paint( g );
        g.drawString( "Hello World", 30, 20 );
    }
}
```

Let's break this program down into each part for discussion. There are four main parts in this program.

- ✦ Import statements
- ✦ Header comments
- ✦ class declaration
- ✦ paint method

Import Statements

The two import statements

```
import javax.swing.*;
import java.awt.*;
```

Import statements allow us to re-use code that has already been written. There are classes that we can use so that we don't have to re-invent the wheel again. There are many standard classes that come with every Java distribution. For example, we want to create an applet and want the text to appear on the screen. Someone else has already written the code to make the screen draw things for us. We can use that class by referencing it: the `JApplet` class in our example. But when the compiler looks for this `JApplet.java` file, it needs to know where it is. We can tell the compiler where to look by specifying that it is in the `javax.swing` package. The star (`*`) designates to the compiler to look at all the classes in that package, as the package may contain many different classes. We could also be more specific and state exactly which class, by the following import statement:

```
import javax.swing.JApplet;
```

Both ways work, and neither is more efficient than the other.

The second import statement tells the compiler to look for classes in the `java.awt` package. This is where it will find the `Graphics` class. The `Graphics` class works with the browser to draw things on the screen. All we need to do is add instructions on exactly what we want drawn and where we want it.

Import statements need to go at the very top of our program.

Header Comments

```
/** first program
 * @author E.S.Boese
 * @version fall 2005
 */
```

The top of your program should always begin with a header comment. This comment area should include a description of the program, your name as the author, the date you created it. There is more information you could provide here, but for now, we will keep it to these basics.

There are two types of comments in Java programs: multi-line comments and in-line comments.

The example above used a multi-line comment, designated with a slash-star (`/*`) to begin the comment and a star-slash (`*/`) to end the comment. Everything between the `/*` and `*/` is ignored when the program is executed. This format allows for the comment to extend over multiple lines, as we see in this example.

Another type of comment is an in-line comment, designated with two forward slashes (//). We will see this in our future examples. The // designates that the rest of the line (everything after the //) is to be ignored as a comment. The // can only apply to a single-line.

Class Declaration

```
public class MyFirstProgram extends JApplet
{
}

```

Each class we create needs a class declaration. Here we define the name of the class. In our example, the name of the class is `MyFirstProgram`. This also means that the class is located in a file called `MyFirstProgram.java`. It is important to note that the class name and the filename must match exactly - including upper/lower case letters.

Because we are developing applets, we want to specify that our class is an applet. To do so, we say that our class **extends** the `JApplet` class. This way we *inherit* the functionality of an applet without re-writing the code to make an applet an applet!

We start out our class declaration with the word `public`, to ensure that we can run the program directly. We're not going to go into much more depth about this at the moment.

Paint Method

```
public void paint( Graphics g )
{
    super.paint( g );
    g.drawString ( "Hello World", 30, 20 );
}

```

The `paint` method is where the meat of the program occurs. In our example, we want to draw the text for "Hello World" inside our applet. The `Graphics` class helps us accomplish this. We can call a *method* on the `Graphics` object. The method we call is the `drawString` method, which prints text.

To use the `Graphics` class, we have a *variable* reference: in our example, we used the name `g`. Now we can call *methods* on this object, by calling `g.<method>` where `<method>` is the name of a method such as `drawString`.

When we call this method, we need to send it some *parameters*. Each parameter is separated with commas. In our example, there are three parameters: "Hello World", 30, and 20. The first parameter, "Hello World", is the text that we want to print. The second parameter 30 designates the x-coordinate of where to draw the text, and the third parameter 20 designates the y-coordinate of where to draw the text.

We'll be talking more about methods later.

HTML file

The HTML file is essential for displaying applets – applets are intended to be embedded within web pages. This needs to be in a separate file than our `.java` file with our source code. HTML files should also have an extension of `.html`. This book is not about writing HTML, so we will only present the essential HTML code to get an applet to display.

```
<HTML>
<BODY>
  <APPLET CODE=JavaClassName.class
    WIDTH = 400
    HEIGHT = 500 >
  </APPLET>
</BODY>
</HTML>
```

JavaClassName should match your class name inside your Java source code

Width and height you want the applet

You MUST have the end tag `</APPLET>` otherwise it will not show up!

Note that some IDE's such as Eclipse create the HTML file for you to run your applets. Therefore, while testing your code within the IDE, you won't need to write your own HTML. However, once you want to put your applet up on the Internet, you will have to create an HTML file.



Running an example – Windows machine

To set up our first applet, follow the steps below on a computer running Windows.

1. Open a "Command Prompt".
(Select start → All Programs → Accessories → Command Prompt)
2. We're going to use the basic editor, Notepad to write our first program. If we want a program named **FirstProgram**, we need to add the .java extension, so we would type:

```
notepad FirstProgram.java
```

3. Type in the following code below, exactly how it appears:

```
/** First program with Java
 * @author: your name
 */
import javax.swing.*;
import java.awt.*;

public class FirstProgram extends JApplet
{
    String text = "Cookie Monster";
    public void paint ( Graphics g )
    {
        g.drawString ( text, 15, 20 );
    }
}
```

4. To save the file, make sure Notepad doesn't add a .txt extension. Change the box that says "Save as Type:" to "All Files", and make sure your filename includes the .java extension.
5. Close Notepad, and return to the Command Prompt window.
6. Now we want to compile the program. The java compiler is called **javac** Type:

```
javac FirstProgram.java
```

If there are any syntax errors, they will be listed in the terminal window. If you have errors, you need to go back to step 2 and correct the code until you get no errors when you compile the program.

- When we run the source code through the compiler without errors, we get a bytecode file. If we get a listing of the directory, we should see the .class file listed:

```
dir
```

You should see the two files listed: FirstProgram.java and FirstProgram.class

- Now we need to create the HTML file to embed the applet. Using Notepad, open a new file named FirstProgram.html

```
notepad FirstProgram.html
```

- Enter the following code exactly in this file:

```
<HTML>
<BODY>
  <APPLET CODE="FirstProgram.class" WIDTH=500 HEIGHT=400>
</APPLET>
</BODY></HTML>
```

Java is case sensitive, so ensure that FirstProgram has a capital F and capital P, just as it is inside the java file.

- Save this file as you did before, this time with the .html extension. Close Notepad. To view the applet, we use the program named: **appletviewer**

```
appletviewer FirstProgram.html
```

A window should appear with Cookie Monster typed in it.



Running an example – Linux machine

To set up our first applet, follow the steps below on a Linux machine.

1. Open a term window.
2. We're going to use the basic editor, pico to write our first program. If we want a program named **FirstProgram**, we need to add the .java extension, so we would type:

```
pico FirstProgram.java
```

3. This editor is similar to Notepad on Windows. Type in the following code below, exactly how it appears:

```
/** First program with Java
 * @author: your name
 */
import javax.swing.*;
import java.awt.*;

public class FirstProgram extends JApplet
{
    String text = "Cookie Monster";
    public void paint ( Graphics g )
    {
        g.drawString ( text, 15, 20 );
    }
}
```

4. To save the file, hold down the cntrl key and press the letter O key.
5. To exit the pico environment, press cntrl-x
6. Now we want to compile the program. The java compiler is called **javac** Type:

```
javac FirstProgram.java
```

If there are any syntax errors, they will be listed in the terminal window. If you have errors, you need to go back to step 2 and correct the code until you get no errors when you compile the program.

- When we run the source code through the compiler without errors, we get a bytecode file. If we get a listing of the directory, we should see the .class file listed:

```
ls
```

(the lower-case letter L and the lower-case letter s)

You should see the two files listed: FirstProgram.java and FirstProgram.class

- Now we need to create the HTML file to embed the applet. Using pico, open a new file named FirstProgram.html

```
pico FirstProgram.html
```

- Enter the following code exactly in this file:

```
<HTML>
<BODY>
  <APPLET CODE="FirstProgram.class" WIDTH=500 HEIGHT=400>
  </APPLET>
</BODY></HTML>
```

Java is case sensitive, so ensure that FirstProgram has a capital F and capital P, just as it is inside the java file.

- Save this file as you did before.
- To view the applet, we use the program named: **appletviewer**

```
appletviewer FirstProgram.html
```

A window should appear with Cookie Monster typed in it.



Summary

- ✦ A **program** is a set of instructions for the computer to execute.
- ✦ The **syntax** is the grammar used by a programming language.
- ✦ A **programming language** is a grammar to designate information and instructions that a computer understands.
- ✦ **Machine code** is made up of 0's and 1's and is the only language a computer understands. Programs in other languages need to be converted to machine language before a computer can execute it.
- ✦ **Assembly language** is a mapping of the machine code to something more readable, based on English words such as "load" and "add".
- ✦ **High-level language** allows us to simplify the program code with a higher level of abstraction than assembly code, such as "x = x + 2".
- ✦ **Source code** is the program that you write, usually in a high-level language such as Java.
- ✦ A **compiler** translates source code into a target language.
- ✦ An **interpreter** translates the code into machine code and then executes the machine code.
- ✦ When Java is compiled, the compiler creates an intermediate coding called **bytecode**.
- ✦ A programming language is considered to be **platform-independent** if it can be interpreted/executed on different machine types (e.g., SunOS, MacOS, Linux, Windows) without changes.
- ✦ Applets are embedded within web pages. The .java file is the source code, the .class file is the bytecode, and the .html file (web page) specifies the .class file.
- ✦ Import statements are placed at the top of a program to designate the packages where other classes used by the program can be found.
- ✦ Header comments are listed at the beginning of a program to specify the author and date.
- ✦ Comments are not executed as part of the program, and are used to explain things about the code. They can either be two forward slashes // to designate the rest of the line is a comment, or across multiple lines by beginning with /* and ending with */
- ✦ The class header starts all programs. This needs to include
public class nameOfProgram extends JApplet
- ✦ The paint method is where we can draw things on the applet.

Intro Exercises

1. True or False. A Java compiler translates source code to machine code.
2. True or False. Java is an object-oriented language.
3. Java is considered to be **platform-independent** because
 - a. the source code can be compiled to machine code on any machine
 - b. the bytecode can be interpreted on any machine
 - c. the source code is the same independent of the machine it is developed on
 - d. the bytecode gets translated to source code on any machine
4. Approximately when did the Internet begin? the WWW? computers?
5. Why do we use the import statement?
6. True or False: Java source code gets compiled into a .class file which is machine code.
7. True or False: Computers store all information using decimal representation.
8. True or False. A byte is a group of 8 binary digits, called bits.
9. What is the difference between information and data?
10. What is bytecode? How does it differ from machine code?
11. What is the definition of syntax?
12. How is the use of an IDE such as Eclipse useful for programming?
13. What's the difference between running Java programs vs. C programs on different computer types? What more do you need to do with a C program?
14. What is a program?
15. What is an algorithm?
16. How do programs differ from algorithms?
17. There are three major programming language types: machine languages, assembly languages, and high-level languages. Give an example of each. How are they inter-related?
18. List the numbers between zero and ten in binary.
19. True/False: Import statements can be listed anywhere in the program.
20. True/False: Comments are used by the interpreter to determine the instruction set.

21. Match the following terms to their best fitting definitions
- | | |
|----------------------|--|
| ___ program | a. grammar |
| ___ package | b. data in context |
| ___ algorithm | c. made up of 0's and 1's |
| ___ syntax | d. steps to solve a problem |
| ___ machine language | e. what Java compiles to |
| ___ compiler | f. a mathematical computation |
| ___ byte code | g. type of programming language |
| ___ assembly | h. set of instructions for a computer |
| | i. referenced using the import statement |
| | j. process of viewing the source code |
| | k. translates the source code in to another language |
22. True/False: Computers store all information using analog representation.
23. How can you add two binary numbers? (e.g., 110101011 + 11010111)
24. What does it mean for a language to be portable? Define "platform-independent" and what the Java motto "write once, run anywhere" means.
25. Explain the process of a Java program: source code, interpreter, compiler, bytecode, machine code. Draw a diagram.
26. If the name of your class is called **CurrencyConverter**, what is the name of the source code file? What is the name of the bytecode file?
27. What is the difference between in-line comments and block comments?
28. What class do applets need to inherit from? What package is it in?
29. Put the following in order of when they came into existence.
- ___ Java
 - ___ WWW
 - ___ computers
 - ___ Internet