Simple Java Program

Here is an example Java program. It is about as small as a Java program can be. When it runs, it writes `Hello World!` on the computer monitor. The details will be explained later.

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

This program can be created using a text editor such as the Notepad editor that comes with Windows. (Details later.) This source program is a text file saved on a hard disk. The file is named `Hello.java`.

A **source program** is a text file that contains a program (such as above) written in a programming language. Since it contains ordinary text (stored as bytes) it cannot be directly executed (run) by the computer system. As a text file, you can print it, display it on the monitor, or alter it with a text editor.

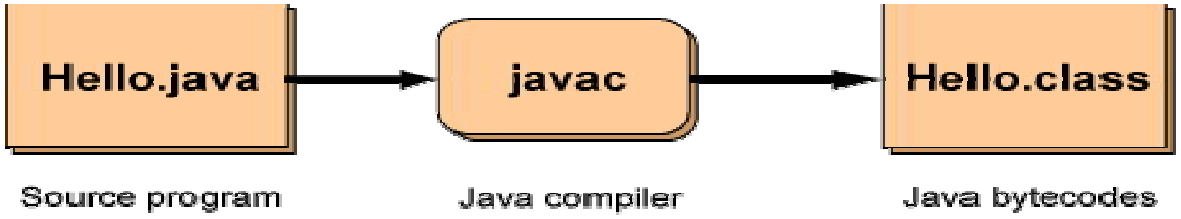
QUESTION 2:

(Q1:) What are the two ways that a source program can be run on a computer system?

Answer:

1. Translation (into machine instructions, which are then directly executed by the processor)
2. Interpretation (by an interpreter program)

Bytecodes



Java Program Translation

Java *combines* these ideas in a way that will take some explaining. To run a Java program the source file is first translated into a file of **bytecodes**.

A Java **bytecode** is a machine instruction for a Java processor. A file of bytecodes is a machine language program for a Java processor.

The picture shows the Java compiler translating the sample Java program `Hello.java` into bytecodes. The file of bytecodes (machine language for the Java processor) is called `Hello.class.`

In this picture, the source program `Hello.java` is examined by a program called `javac` running on your computer. The javac program is a compiler (a translator) that translates the source program into a bytecode file called`Hello.class.`

Important Idea: The bytecode file will contain exactly the same bytecodes no matter what computer system is used.

The architecture of the processor that executes Java bytecodes is well-documented and is available to anyone. The Java compiler on a Macintosh will produce the exact same bytecodes as the Java compiler on an Intel system.

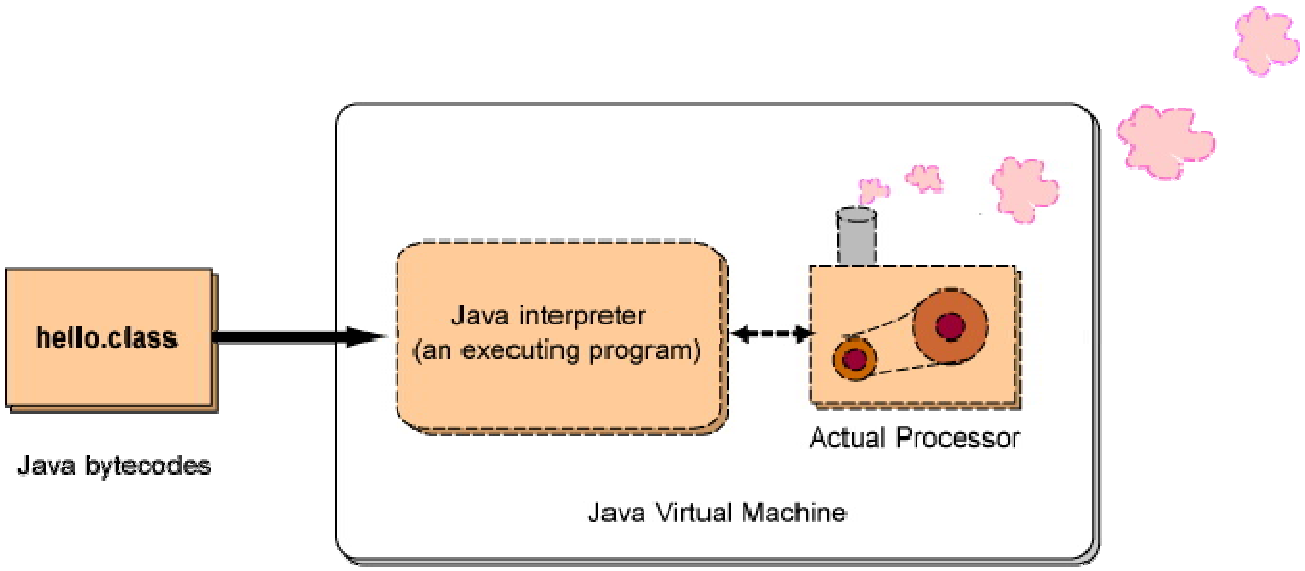
QUESTION 3:

Could a processor chip be built that executes Java bytecodes directly, just as a Pentium executes its machine language directly?

Answer:

Yes. Hardware Java processor chips have been created out of silicon and can execute bytecode files just as a Pentium executes files of its machine language.

Java Virtual Machine



Interpreting Java Bytecode on a Virtual Machine

Usually, however, people do not have hardware Java processor chips. They have ordinary PCs and Macintoshes.

Now for the clever part: the Java processor can be implemented as software! It is implemented as a program that reads the bytecodes and performs the operations they specify. (This type of program is called an interpreter.) The Java bytecode interpreter is an executable program that runs on whatever computer system you have.

The "Java interpreter" in the picture is an executable program that is running on an ordinary computer system, such as a desktop Pentium system. Each type of computer system has its own Java interpreter that can run on that system. The "Actual Processor" is the actual, hardware processor chip of that computer system.

(Another) **Important Idea:** When the Java interpreter is running on a computer system, that system acts just like a hardware Java bytecode processor. It is a *Java Virtual Machine.*

Any computer system can execute a Java bytecode program by using a Java interpreter. The Java interpreter has to be specifically written for that particular computer system, but once that is done, the computer system can become a Java virtual machine. That is, it looks like a computer with a hardware Java processor chip and can run Java bytecodes.

When a Java program is translated into bytecodes, the bytecodes are exactly the same no matter what computer system is used. This means the bytecodes on a Sun computer can be sent to an Intel based computer and they will run without a problem.

QUESTION 4:

Say that Apple has just come out with a new computer and wants this computer to run Java programs. What must Apple do?

Answer:

Apple must write a Java interpreter for their new system.

Portability

Once Apple writes the interpreter and includes it on their new system, the system can run any Java program. Nothing new needs to be done with those programs.

Java programs are **portable**, which means that the same bytecode program can run on any computer system that has a Java interpreter. Also, a source program can be compiled into bytecodes on any computer that has a Java compiler.

The source program does not have to be changed to meet the particular needs of various computer systems. No matter what computer you have, you can write the same Java programs.

This is unlike most other programming languages, where a different version of a program must be made for each variety of computer, and an executable program that runs on one type of computer will not run on another.

QUESTION 5:

Can bytecodes be sent from computer to computer over the Internet?

Answer:

Yes.

Applets

An **applet** is a Java bytecode program that runs on a Web browser. Most up-to-date Web browsers include a Java interpreter. A Web page may contain an applet, which means that part of what the page displays is controlled by Java bytecodes. The computer that hosts the Web page sends Java bytecodes to a client computer (like yours) that has asked for the page. The Web browser on the client runs the Java applet using its built-in interpreter.

Applets are used for user interaction, graphics, and animation. Applets will be discussed in later chapters of these notes. For now, let us concentrate on Java programs that get input from the keyboard and write output to the command prompt window of the monitor. These are called Java application programs.

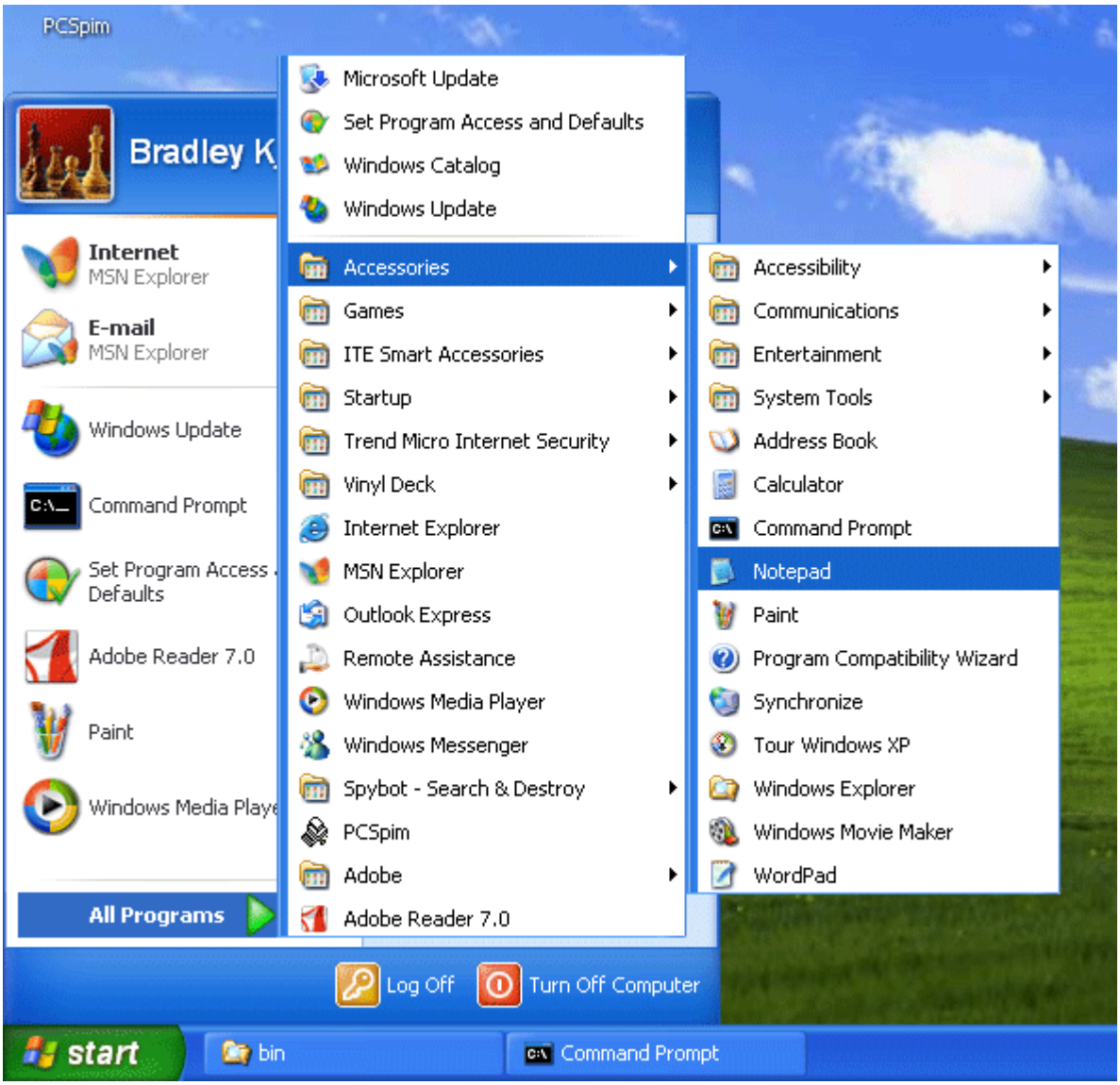
QUESTIOj;N 6:

Is the Java used to write applets the same Java as that used to write applications?

Answer:

Yes.

Creating a Java Source Program



To start out, use the Notepad editor that comes with Microsoft Windows operating systems. Notepad is simple to use and good for getting started. After you see how things work using Notepad, move on to a better text editor (such as Crimson or Notepad++) or to a Java integrated development environment (IDE) (such as BlueJ or Dr. Java). For now, our goal is to create a text file called `Hello.java` containing the text at right.

On a Windows computer, first start up the command prompt. Look for the "Command Prompt" icon. Click on the "Start" button in the lower left of your screen. Then click "All Programs" and the "Accessories". You should now see a menu of choices similar to the picture.

Recent Windows operating systems will add the Command Prompt icon to the first start menu once you have used it. The more recent the Windows system, the harder it is to find the command prompt. When you find it, click on the "Command Prompt" icon to start a command prompt window.

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

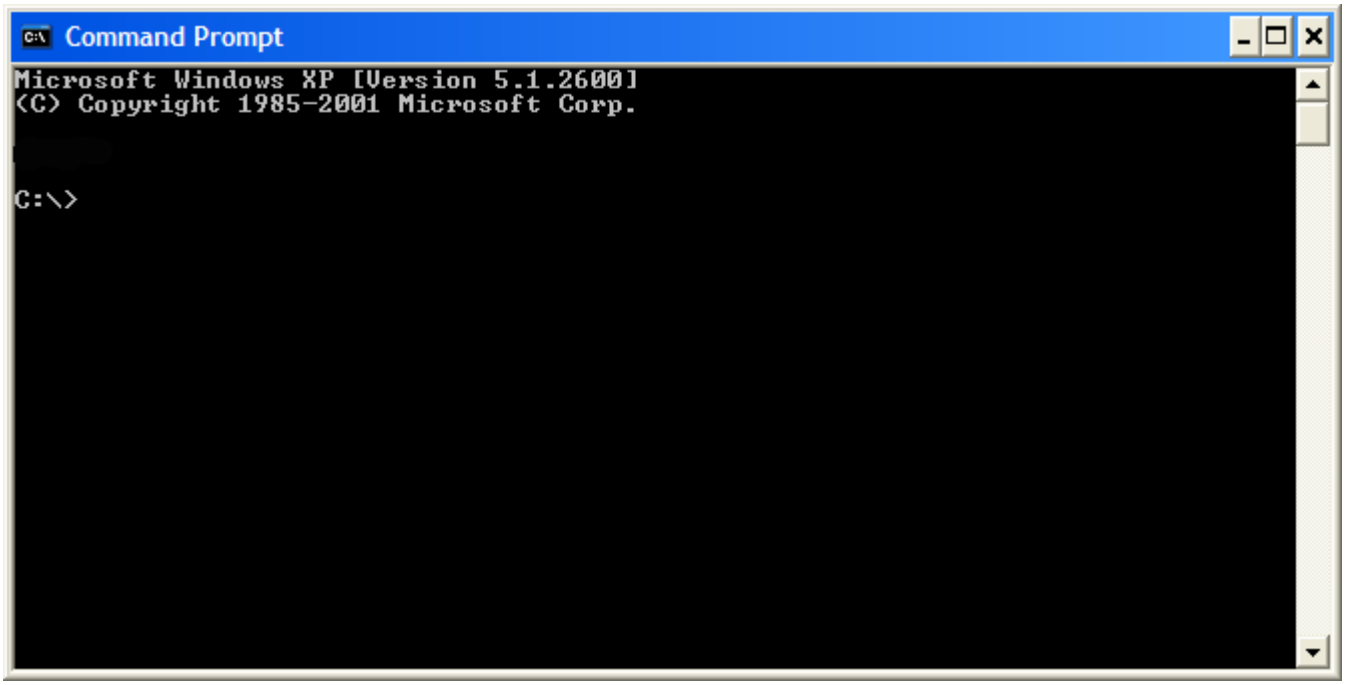
QUESTION 7:

Will you see exactly the same things as above on your computer system?

Answer:

No. You will see different choices, depending on what programs you have installed, and what programs you have recently run.

Command Interpreter Window



After you start the command prompt window you can enter commands as if you were running an old-time DOS computer. You should see something like the picture. The line of text

```
C:\>
```

is a **prompt**. You are expected to enter commands to the system after it. Depending on how your computer has been set up, you may see a prompt something like this:

```
C:\WINNT\System32>
```

This prompt means that the command interpreter is automatically expecting to use files in the directory`C:\WINNT\System32.` This is called the **default directory**. (A directory is a section of the disk that has a name. A directory can hold many files.) You should not use the `C:\WINNT\System32` directory.

It doesn't matter where you start out because you can always move to where you want to be using the `CD` command. For now, let us create a Java program in the `C:\Temp`directory. To get to this directory type the following commands:

```
C:
CD \Temp
```

Type these commands after whatever command prompt you see. The prompt changes to show the current default directory. The first command `C:` switches to the `C:` disk (which is the hard disk of the system if you have only one). The second command `CD \Temp` makes `C:\Temp` the default directory.

QUESTION 8:

Can you create a directory?

Answer:

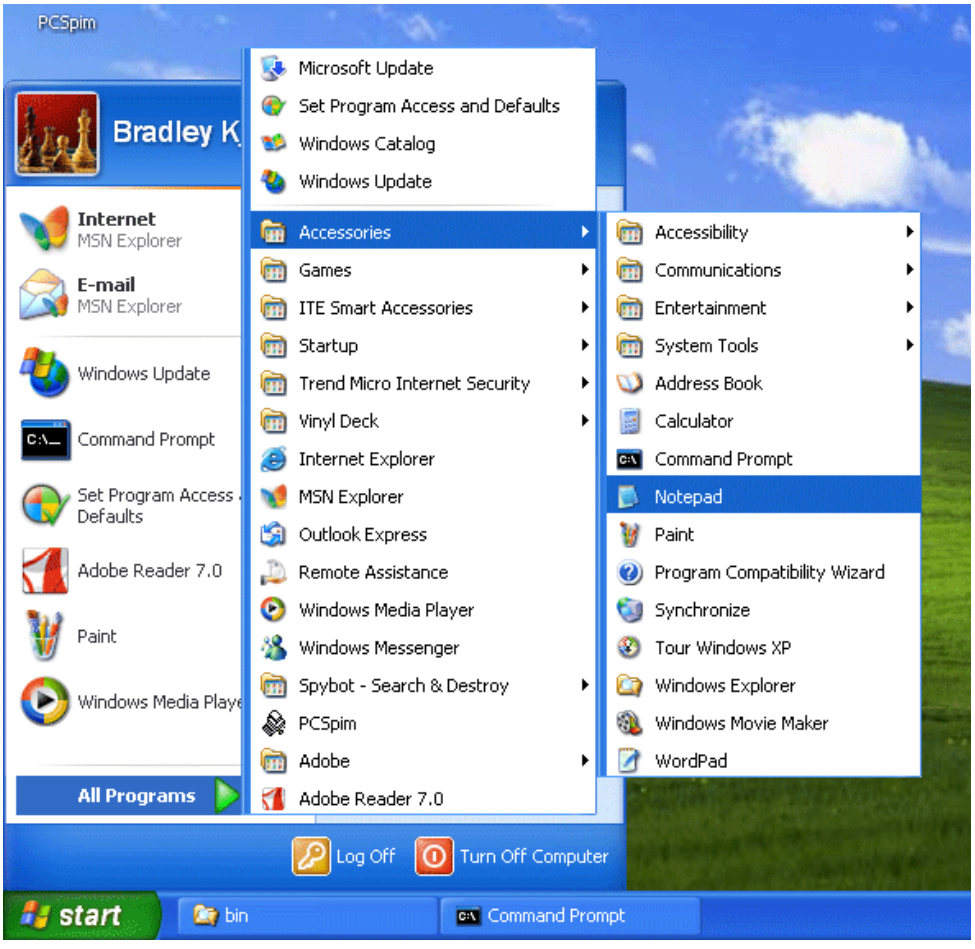
Yes. If you know how, create a new subdirectory (perhaps "C:\JavaPrograms") and use the CD command to move to it.

With the command prompt window, the command to create a new directory is:

```
MDdirectoryName
```

If you want, you can create a new directory using Windows Explorer and then use `CD` to move to it in the command prompt window.

Starting Notepad



We will use the *Notepad* text editor which comes with Microsoft operating systems. Click on the "Start" button in the lower left, then on "All Programs" then on "Accessories", and then on

"Notepad". As you might expect, different versions of Windows put Notepad in different places. You may have to hunt.

Roughly similar things can be done on Mac and Linux systems.

After clicking on the icon, Notepad starts. Another way is to start it from the command prompt:

```
C:\TEMP> notepad
C:\TEMP>
```

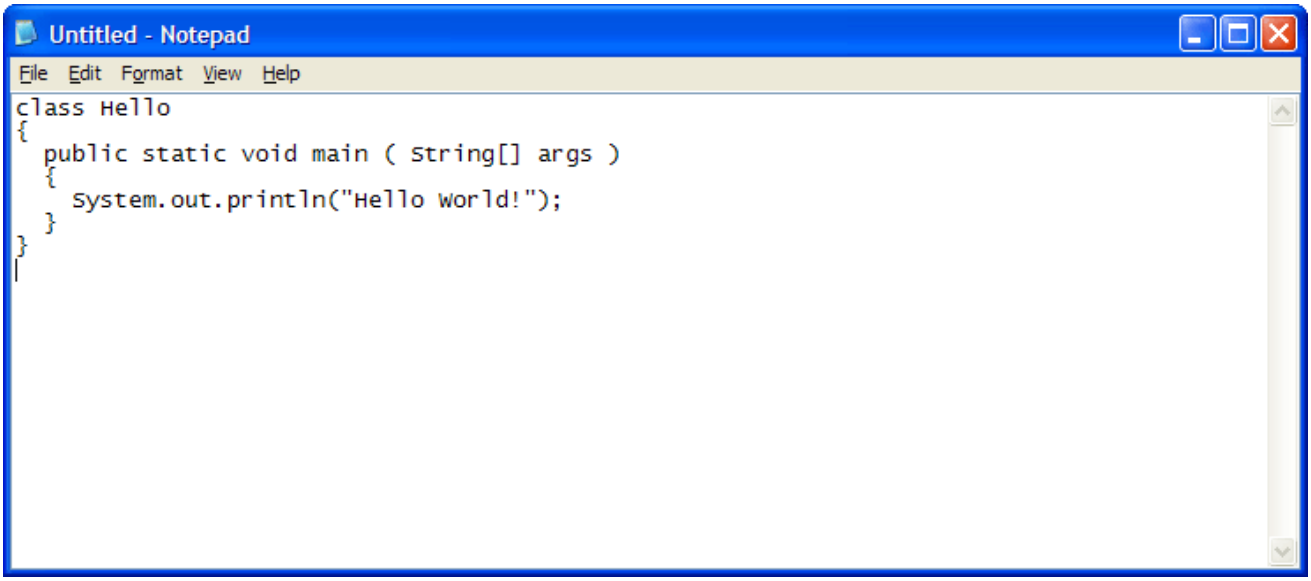
QUESTION 9:

Is all this getting to be just too much?

Answer:

- **Yes** — Find a Web site that explains basic computer use and DOS commands, or find a suitable book in your library. You will only need to look at the first one or two chapters. Better yet, find a friend that knows all this and is happy to show off.
- **No** — Good.

Notepad



Once Notepad is running, just type in the program, as in the picture.

To enter characters, just type them. You can move around the text using the mouse or arrow keys. To delete mistakes, use the "backspace" key or the "delete" key. To start a new line, just hit "Enter".

In typing in the program, make sure upper and lower case letters and all punctuation are **exactly** correct. You do not have to get the spaces exactly correct.

Now you need to save the file to the hard disk. One of the selections in the menu bar at the top of the Notepad window is used to do this.

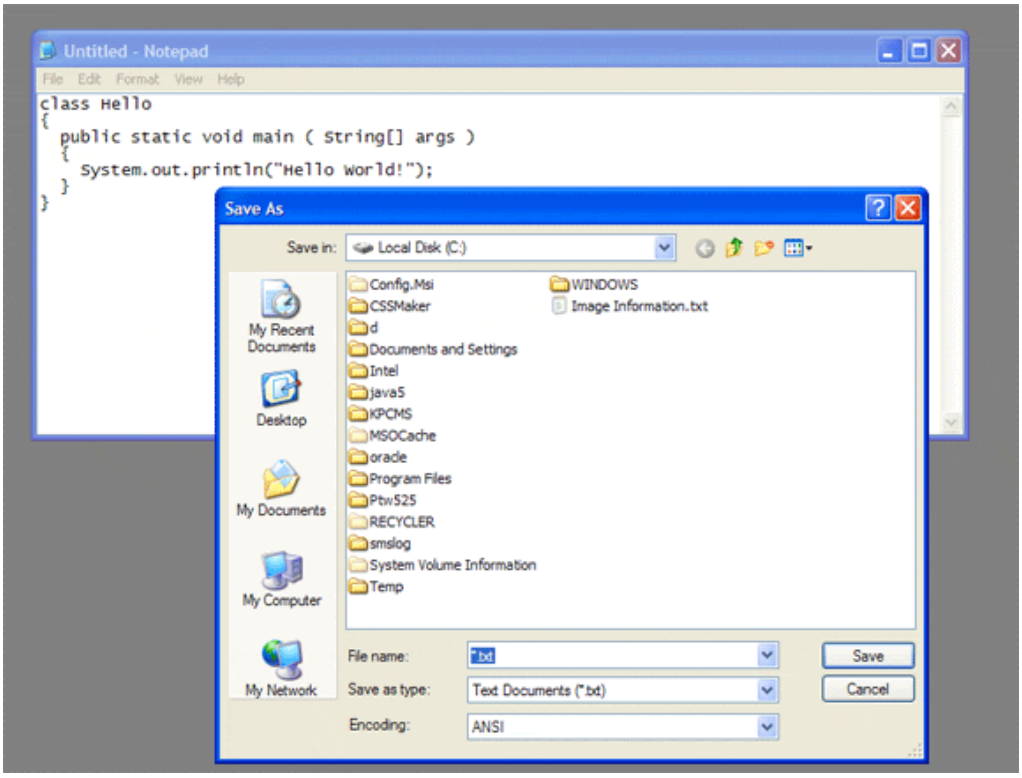
QUESTION 10:

Which of the menu selections (in the menu bar at the top of the Notepad window) do you suppose is used to save the file?

Answer:

The "File" menu.

Saving the Source File



Move the mouse pointer to "File" and click on it. You will get a sub-menu. Now click on "Save As". As the picture shows, you may not start out in the C:\Temp directory. If not, you will have to use the various controls at the top of the "Save As" dialog to get to the directory you want.

When you see it, click on the "Temp" directory to say that it is where you wish to save your program. The "Temp" directory is intended for temporary files and will usually be full of many different files.

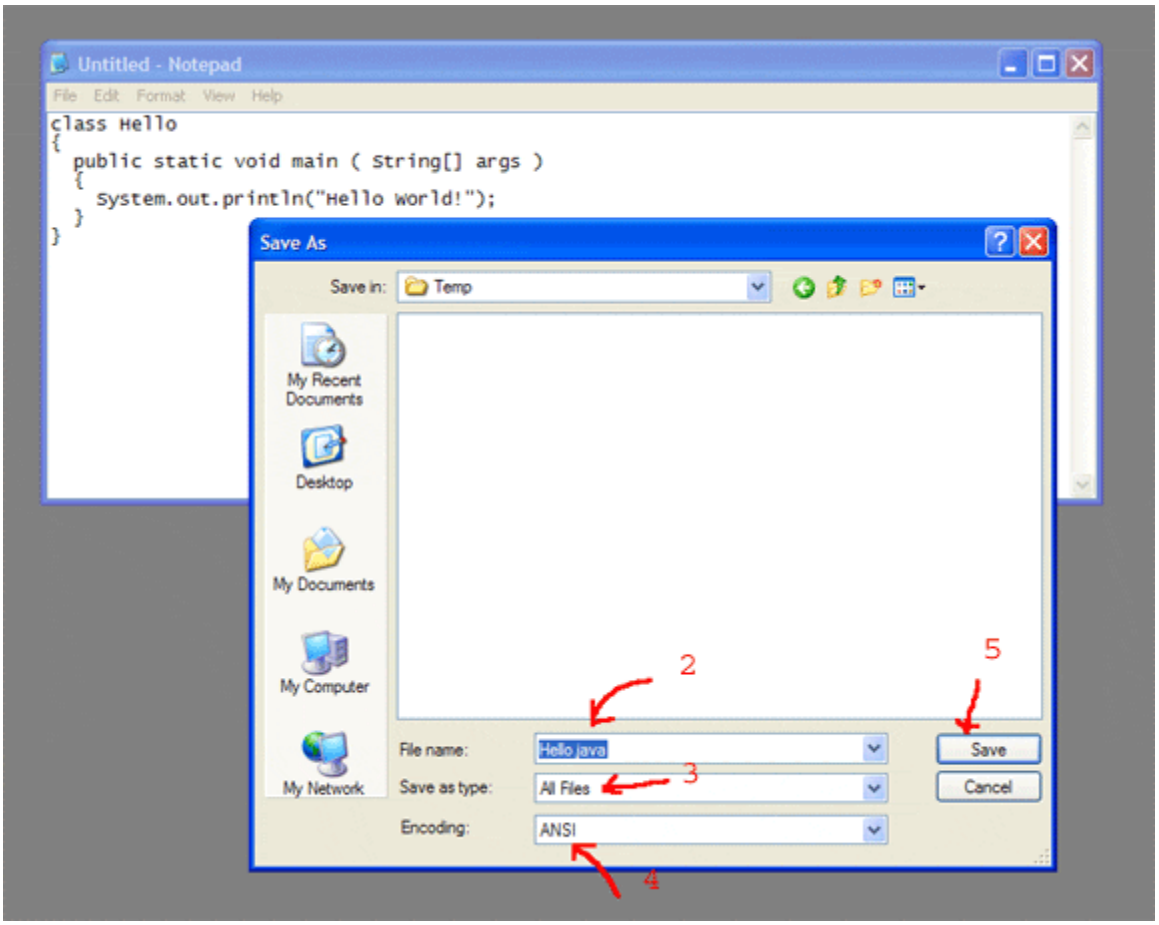
QUESTION 11:

If you save the file in a different directory than the one being used for the DOS window, will there be a problem?

Answer:

Yes. You want the source file `Hello.java` to be saved in the disk directory that is the default directory for the command prompt window.

Naming the File



The name of the source file should be the same as the name that follows the word `class` in the program (match upper and lower case), and should be given the extension `.java`(lower case). For example, our program contains the line:

```
class Hello
```

so the file name must be `Hello.java`. When you get to the `Temp` directory (or whatever default directory you are using):

1. Type the file name into the box.

2. The file name should be `Hello.java`
    - For some versions of Notepad you should put quote marks around the file name like this: "Hello.java" even though the quote marks will not be part of the file name.
    - Sometimes Notepad automatically uses the extension ".txt" and you may need to struggle to name the file what you want.

3. Select "Save as all files" by clicking on the little down arrow it the box below the file name box.
4. Select "Encoding: ANSI" in the bottom box.
5. Click on "Save"

Your system is likely to be slightly different. Experiment with different choices until you are able to create a text file that contains the sample Java program.

Fussy, bothersome, irksome, irritating details. Yes, I know... Some people actually like this stuff. But if you can tie your shoes or drive a car you can do this. It might take some practice.

QUESTION 12:

So, finally your Java program source file is saved. What must you do to run it?

Answer:

Compile it into Java bytecodes, then run the bytecode file with the Java interpreter.

Running the Program

To do all that, find the DOS command prompt window you started up a while back. It should still be positioned in the `C:\Temp`directory (or other directory of your choice). To check that you put the source file `Hello.java` in this directory, enter the command `dir *.java`. You should see the source file, as below.

```
C:\TEMP>dir *.java
 Volume in drive C has no label.
 Volume Serial Number is 7C68-1E55

 Directory of C:\TEMP

08/23/98  01:07a                 115 Hello.java
               1 File(s)            115 bytes
                          2,448,368,640 bytes free
```

To compile the source file (thereby producing a file of bytecodes) enter the command `javac Hello.java`.

```
C:\TEMP>javac Hello.java
compiling: Hello.java
```

Notice that this command is **javac**, java-with-a-c which invokes the Java compiler. If you see the following

```
C:\TEMP>javac Hello.java
The name specified is not recognized as an
internal or external command, operable program or batch file.
```

then the PATH environment variable has not been set correctly. Look at Appendix C for information about this. A temporary alternative is to tell the command promt exactly where to find `javac`:

```
C:\Temp> C:\Program Files\Java\jdk1.6.0_07\bin\javac Hello.java
```

Adjust the above command to match whatever directory your version of Java has been installed in. Finally, to run the program, enter the command `java Hello`.

```
C:\TEMP>java Hello

Hello World!

C:\TEMP>
```

Notice that this command is **java**, java-without-a-c which invokes the Java interpreter. Again, if PATH is incorrect, you can run`java` by using the full path name to its location.

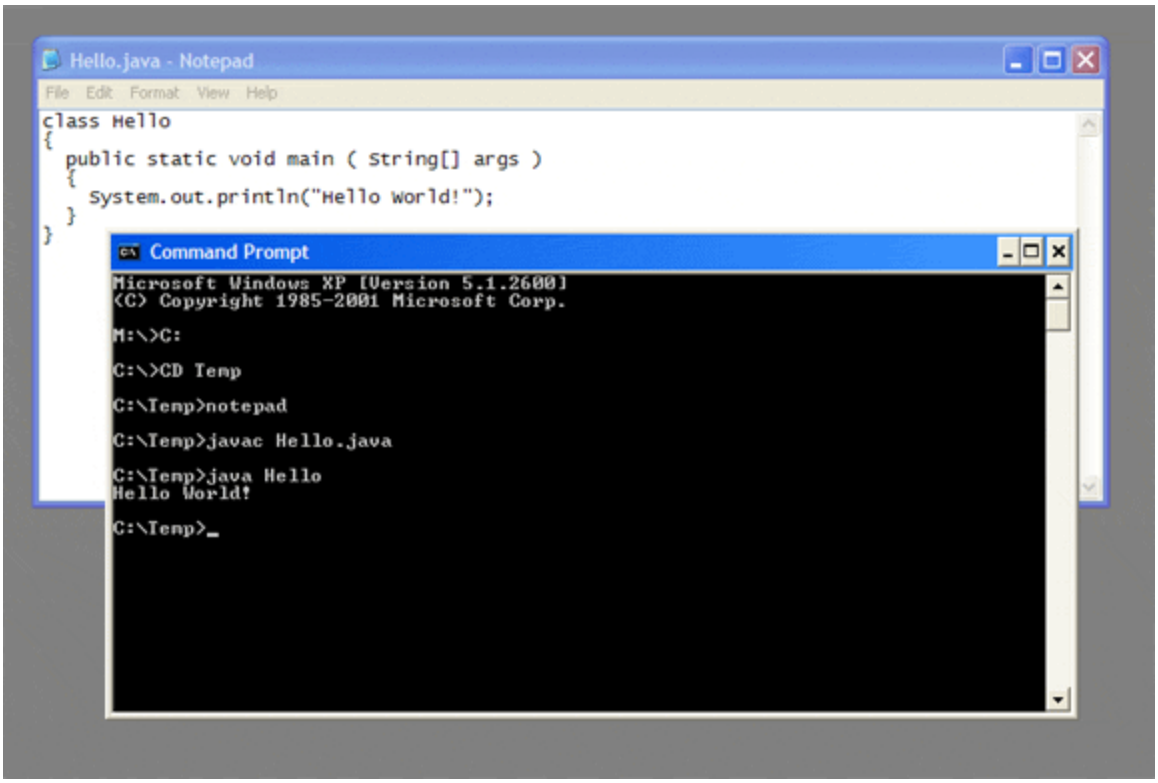
QUESTION 13:

After all of this, what did the Java program actually do?

Answer:

The example Java program wrote the words "Hello World!" in the command interpreter window.

Result of Running the Example Java Program



The picture shows my system when everything has worked correctly (which happens every time, of course).

Your program may not have run correctly. It is possible that you did not type in exactly the right characters. Spaces don't particularly matter. But check especially characters like [and { and (and ". You have to use the correct ones. Check that upper case and lower case characters match the sample program.

Fuss around for a while to get the program to work. If you just can't get it to work, give up and move on. Probably there is some trival detail that you have overlooked and will see clearly later on. This happens all the time in programming.

If everything worked out perfectly, then go back and do something wrong. This will help you understand what happened later on when you unintentionally do something wrong.

QUESTION 14:

Had enough, for now?

Answer:

Yes.

End of the Chapter!

Data Types in Java

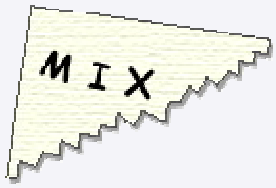
Computer memory stores arbitrary bit patterns. Making sense of these patterns involves the use of *data types*.

Topics:

- Data Types
- Primitive Data vs. Objects
- The Eight Primitive Data Types of Java
- Numeric Data Types
- Character and Boolean Data Types

QUESTION 1:

Say that you came across the following torn slip of paper. What can it mean?



?

Answer:
Nothing

Data Types

You would like to see the rest of the paper, or know where it came from. Without knowing the context, it is hard to say what MIX means. It *could* be 1009 in Roman numerals, or it could be the English word "mix" (which itself has several meanings), or it could be the last name of the old-time radio hero Tom Mix. It could be part of a label on a music CD, "MTV Dance MIX", or part of a label on a bottle, "BLOODY MARY MIX". Or maybe you are looking at it upside down and it should be "XIW". Of course, it might not be English at all. Without knowing the context, a string of letters has little meaning.

Computer memory stores arbitrary bit patterns. As with a string of letters, the meaning of a string of bits depends on how it is used. The scheme being used for a particular string of bits is its **data type.**

A **data type**

- Is a scheme for using bits to represent values.
- Values are not just numbers, but any kind of data that a computer can process.
- All values in a computer are represented using one data type or another.

For example

```
0000000001100111
```

is a pattern of 16 bits that might be found somewhere in computer memory. What does it represent?

Without knowing more about how the above pattern is being used, it is impossible to say what it represents. The type`short` is one of Java's data types. If the pattern is of data type `short`, then it represents the value 103 (one hundred and three).

QUESTION 2:

What does the following 16 bit pattern represent?

```
0000000000000000
```

Answer:

Without knowing more about how the pattern is being used, it is impossible to say.

There are Many Data Types

You might be tempted say that the pattern 0000000000000000 represents "zero". But it doesn't necessarily. Even such an obvious pattern has no automatic meaning.

If you were told that the above pattern were of type `short`, then you would know that it represents the integer zero. Here is another pattern:

```
1111111110011001
```

As a `short`, this pattern represents -103 (negative one hundred three).

There are uncountably many types of data that can be represented in the memory of a computer. If specific patterns always had specific meanings, then only a few types of data could be represented. This would be much too restrictive. Many data types are *invented* by programmers as they write programs. Without this flexibility computers would be much less useful.

The main memory of a general purpose computer (such as a desktop or a laptop computer) contains very many bits. Programs can use this memory with whatever data types they need. For example, one program may use data type `int`with four bytes of memory, and data type `double` with another eight bytes of memory. A different program may use the same memory with different data types.

Not all machines use memory this way. A simple electronic calculator, for example, uses memory for one purpose only: to store floating point numbers. It uses only one data type, and can do only those few things with that data type that it has been wired to do. The engineers who designed the calculator decided how to represent numbers with bit strings, and then designed the electronics to work with just those strings. This is too restrictive for a general purpose computer.

QUESTION 3:

Do you imagine that the computers of the 1960's were built to handle audio data?

Answer:

No — certainly this was not a common use for computers then.

Primitive Data Types

byte	short	int	long	float	double	char	boolean

But some 1960's computers *did* deal with audio data; it was merely a case of inventing a way to represent audio with bit patterns and then writing the programs for it.

It would be awkward if every time you used data you had to invent your own scheme to represent it with bits. There are types of data that are so fundamental that ways to represent them are built into Java. These are the *primitive data types*. The eight primitive data types are: byte, short, int, long, float, doubhle, char, and boolean.

Upper and lower case characters are important in these names. So "byte" is the name of a primitive data type, but "BYTE" is not. Computer languages where case is important are called **case sensitive**. Some languages are not case sensitive, especially old languages that were designed when data entry equipment did not have lower case characters.

In the phrase *primitive data type* the word **primitive** means "a fundamental component that is used to create other, larger parts." This word is used frequently in computer science. To solve a large problem, you look for the primitive operations that are needed, then use them to build the solution.

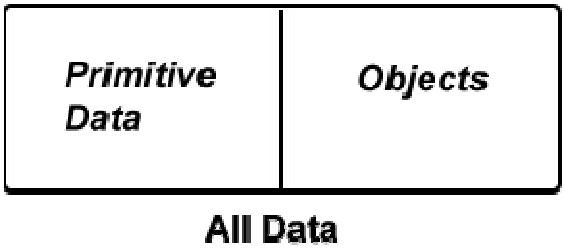
QUESTION 4:

(Trick Question:) Is `Int` a primitive data type?

Answer:

No — it is not on the list of primitive data types (the word that starts with a small "i" is on the list: `int`). Remember that Java is case sensitive.

Objects



All data in Java falls into one of two categories: **primitive data** and **objects**. There are only eight primitive data types. However, Java has *many* types of objects, and you can invent as many others as you need. Any data type you invent will be a type of object.

Much more will be said about objects in future chapters (since Java is a *object oriented* programming language). The following is all you need to know, for now:

- A primitive data value uses a small, fixed number of bytes.
- There are only eight primitive data types.
- A programmer can not create new primitive data types.

- An object is a big block of data. An object may use many bytes of memory.
- An object usually consists of many internal pieces.
- The data type of an object is called its **class**.
- Many classes are already defined in Java.
- A programmer can invent new classes to meet the particular needs of a program.

QUESTION 5:

Are numbers of fundamental importance to computer programs?

Answer:

Yes.

Numeric Primitive Data Types

Numbers are so important in Java that 6 of the 8 primitive data types are numeric types.

There are both *integer* and *floating point* primitive types. Integer types have no fractional part; floating point types have a fractional part. On paper, integers have no decimal point, and floating point types do. But in main memory, there are no decimal points: even floating point values are represented with bit patterns. There is a fundamental difference between the method used to represent integers and the method used to represent floating point numbers.

Each primitive type uses a *fixed* number of bits. This means that if you are using a particular data type then the same number of bits will be used no matter what value is represented.

For example, all values represented using the `short` data type use 16 bits. The value zero (as a `short`) uses 16 bits and the value thirty thousand uses 16 bits.

All values represented using the `long` data type use 64 bits. The value zero (as a `long`) uses 64 bits, the value thirty thousand uses 64 bits, and the value eight trillion uses 64 bits.

Values that are large in magnitude (negative or positive) need more bits to be represented. This is similar to writing out numbers on paper: large numbers need more digits. If a value needs more bits than a particular data type uses, then it cannot be represented using that data type.

In the tables, E means "ten to the power of". So `3.5E38` means 3.5×10^{38}

Integer Primitive Data Types		
Type	**Size**	**Range**
byte	8 bits	-128 to +127
short	16 bits	-32,768 to +32,767

int	32 bits	-2 billion to +2 billion (approximately)
long	64 bits	-9E18 to +9E18 (approximately)

Floating Point Primitive Data Types		
Type	**Size**	**Range**
float	32 bits	-3.4E38 to +3.4E38
double	64 bits	-1.7E308 to 1.7E308

QUESTION 6:

Say that you want to deal with the number 1,023,004 in your computer program. Would data type `short` be an appropriate choice?

Answer:

No. Data of type `short` can be only in the range -32,768 to +32,767.

More Bits for More Range

Larger ranges of numeric values require more bits. The different sizes for integer data enable you to pick an appropriate size for the data you are working with. Usually you should pick a data type that has a range **much greater**than the range of numbers you expect to deal with. If a program uses only a few dozen variables it will run just as fast and take up about as much main memory no matter what size is used for its variables.

Why do the small sized data types exist, then? Well, many real-world programs deal with massive amounts of data (billions of data items) and then using the smaller sizes may save significant amounts of space and time. But we will not use that much data in these notes. Usually you should use`int` or `double` for your numeric data. When you write a program you do not have to know how to represent a number in bits. You can type the number just as you would on a typewriter. This is called a **literal**. The word "literal" means that a value is explicitly shown in the program.

For example, 125 literally repesents the value one hundred twenty five. Integer *literals* in a program are written as in a book, except there are no commas:

```
125   -32   16   0   -123987
```

All of the above examples are 32 bit `int` literals. A 64 bit`long` literal has a upper case 'L' or lower case 'l' at the end. However, **NEVER** use the lower case 'l' because it is easily confused with a digit '1'.

```
125L   -32L   16L   0l   -123987l
```

The last two examples use lower case 'l' and are very confusing.

QUESTION 7:

Is the following an integer literal?

```
197.0
```

Answer:

No — it has a decimal point.

Floating Point Types

If you use the literal 197.0 in a program, the decimal point tells the compiler to represent the value using a**floating point** primitive data type. The bit pattern used for floating point 197.0 is very much different than that used for the integer 197. There are two floating point primitive types.

Data type `float` is sometimes called "single-precision floating point". Data type `double` has twice as many bits and is sometimes called "double-precision floating point". These phrases come from the language FORTRAN, at one time the dominant programming language.

In programs, floating point literals have a decimal point in them, and **no commas** (no thousand's separators):

```
123.0   -123.5

-198234.234   0.00000381
```

Note: Literals written like the above will automatically be of type `double`. Almost always, if you are dealing with floating point numbers you should use variables of type `double`. Then the data type of literals like the above will match the data type of your variables. Data type `float` should be used only for special circumstances (such as when you need to process a file of data containing 32 bit floats).

Floating Point Primitive Data Types		
Type	**Size**	**Range**

`float`	`32 bits`	`-3.4E+38 to +3.4E+38`
`double`	`64 bits`	`-1.7E+308 to 1.7E+308`

QUESTION 8:

(Thought question:) Do you think that using `float` instead of `double` saves a significant amount of computer memory?

nswer:

No. For most programs using variables of type `double` will cost only a few extra bytes in a program thousands of bytes long.

Floating Point Literals

Sometimes you need to explicily ask for a single-precision`float` literal. Do this by putting a lower case 'f' or upper case 'F' at the end, like this:

```
123.0f          -123.5F

-198234.234f    0.00000381F
```

Sometimes you need to explicily ask for a double-precision`double` literal. Do this by putting a lower case 'd' or upper case 'D' at the end, like this:

```
123.0d          -123.5D

-198234.234d    0.00000381D
```

Remember, that without any letter at the end, a floating point literal will automatically be of type `double`.

QUESTION 9:

Do you think that the following is legal?

```
8912D
```

Answer:

Yes. The 'D' will make the literal a `double` (even though it lacks a decimal point). However, to avoid confusion, always include a decimal point in a floating point literal, even where it is not required.

Scientific Notation

You will sometimes see **scientific notation**. The following are all double-precision literals:

```
1.23E+02   -1.235E+02

-1.98234234E+05    3.81E-06
```

The big "E" means "times 10 to the power of" . The integer that follows it says what power of ten to multiply the rest of the number by.

Another way to say this is that the integer that follows "E" says in which direction and for how many places to shift the decimal point. Positive integers mean right shifts; negative integers mean left shifts.

QUESTION 10:

What is the usual way to write this number: 1.9345E+03

Answer:

1934.5

The +03 says to move the decimal point three places right.

Precision of Floating Point Numbers

Consider writing the value 1/3 in decimal notation:

```
0.333333333333333333
```

There is no limit to the number of 3's required for complete accuracy. With a limited amount of paper, you can not be completely accurate. With a data type, there is a limited number of bits. Those bits cannot accurately represent a value that requires more than that number of bits.

The data type `float` has 23 bits of precision. This is equivalent to only about 7 decimal places. (The rest of the 32 bits are used for the sign and size of the number.)

The number of places of precision for `float` is the same no matter what the size of the number. Data type `float` can represent numbers as big as about 3.4E+38. But the precision of these large numbers will also be about 7 decimal digits.

Remember: data type `float` has about the range and precision of a cheap electronic calculator. This is usually not sufficient.

QUESTION 11:

What is wrong with the following constant, expected to be of type `float`?

```
1230.00089F
```

Answer:

There are *nine* decimal places of precision. Data type `float` can't handle that. (The compiler will round the number into a value that can fit in a `float`).

Precision of Double

You might wish to argue that there are only five places used in the above number: the places used by the digits 1, 2, 3, 8, and 9. However, the four 0's in the middle do count. It takes bits to represent them, even if they are zeros.

Primitive data type `double` uses 64 bits, and has a much greater range, -1.7E+308 to +1.7E+308. It also has a much greater precision: about 15 significant decimal digits.

Because of this, if you write a literal like 2.345 in a Java program, it will automatically be regarded as a double, even though a float might be good enough. The other numbers in the program might need to be double, so we might as well make them all double.

QUESTION 12:

Do you suspect that *characters* are important enough to be one of the eight primitive data types?

Answer:

Yes.

The `char` Primitive Data Type

Computer programs frequently work with character data. The primitive data type for characters in Java is named `char`. The `char` type represents a character using 16 bits. In many programming languages, only 8 bits are used for this purpose. Java uses 16 bits so that a very large number of characters can be represented, nearly all of the characters in all of the World's languages. The method used is called Unicode.

For example, here is a 16 bit pattern:

```
0000000001100111
```

If you know that these 16 bits are of data type `char`, then you could look in a table and discover that they represent the character 'g'. If you have a really good memory, you might recall that the same 16 bits represent the integer 103 if they are regarded as data type `short`. Knowing the data type of a pattern is necessary to make sense of it.

Upper and lower case characters are represented by different patterns. Punctuation and special characters are also `char` data. There are also special characters, like the space character that separates words.

Control characters are bit patterns that show the end of a line or where to start pages. Other control characters represent the mechanical activities of old communications equipment (such as teletypes) that are rarely used these days. Many of these control characters are no longer used for their original purpose.

Primitive type `char` represents *a SINGLE* character. It does not include any font information. When you want to deal with more than one character at a time (almost always), you need to use objects that have been built out of `char` data.

QUESTION 13:

Is each of the following a different character?

```
0       O       o
```

(Depending on the fonts your browser is using, you may have to look carefully.)

nswer:

Yes. Each of the above (zero, Capital 'O', and lower case 'o') is a different character and has its own 16 bit code.

Character Literals

In a program, a character literal is surrounded with an apostrophe on both sides:

```
'm'       'y'         'A'
```

In a program, control characters are represented with several characters inside the apostrophes:

```
'\n'       '\t'
```

Each of these is what you do in a program to get a *single*char. The first one represents the 16 bit newline character and the second one represents the tabulation character. You will rarely use any control characters other than these two. Several others are listed in the Java documentation.

Warning: The following is **not** a character literal:

```
"Hello"
```

This is a `String`, which is not primitive data. It is, in fact, an object. `Strings` are surrounded by double quote marks `"`, not by apostrophes.

QUESTION 14:

What is wrong with the following `char` literal:

```
"W"
```

Answer:

The character is not surrounded by apostrophes. It should be: `'W'` .

With double quotes, `"W"`, you get a `String` that contains a single character. This is not the same as a primitive single character. A `String` is represented as an object and may consist of several hundred bytes. A primitive character data item is always only two bytes.

Primitive Data Type `boolean`

Another of the primitive data types is the type `boolean`. It is used to represent a single true/false value. A `boolean`value can have only one of two values:

```
true              false
```

In a Java program, the words `true` and `false` always mean these `boolean` values. The data type `boolean` is named after George Boole, a nineteenth century mathematician, who discovered that a great many things can be done with true/false values (otherwise known as bits).

QUESTION 15:

Would you like a data type named after you someday?

```
true     false
```

Answer:

Maybe.

End of Chapter