***Get JAVA***

To compile programs you need the JDK (Java Development Kit). To RUN programs you need the JRE (Java Runtime Environment). This download will get BOTH of them, so that you will be able to both compile and run the codes you compile.

I will just tell you what I did (on August 28, 2012).

I went to: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

and saw this:



I clicked on the leftmost option ( to download Java Platform (JDK) 7u6). Then I saw: 

I accepted the License Agreement and picked the one next to the bottom: …-i586.exe, but your computer may need a different one. This window popped up:



I clicked “Run”. After it downloaded (about 30 seconds), it gave me this:



I clicked “Run”. There were a couple of windows I didn’t capture (“verifying” or something), and then it said:



I picked “Next” for this and the next window. It ran for about a minute and then gave me:



I picked “Next” again. Notice that this step updates your JRE, the program that runs Java programs (and is used by lots of programs). If you do not do this, then there is a chance that your existing JRE will not run the Java programs you write!

Then, after some gyrations, I got:



I clicked on “Close”. (Don’t know if I will regret that later.) It then gave me:



which I also closed (that is, I closed the window itself).

Assuming you followed my example, let’s see what you have. Open an MSDOS window (“Command Prompt”) and enter “java”. This is the command to RUN Java programs (although you did not tell it which one to run!). But we are just checking to see if it knows what you are talking about.

If it says “’java’ is not recognized as an internal or external command”, then you are in trouble that I do not know how to get you out of. It SHOULD respond by simply chiding you for not giving it a program to run (by starting with “Usage: java [-options] class …” followed by multiple pages of “options” that begin with a minus sign. If you see this, you are doing fine.

But, now enter “javac” which is the command we will use to COMPILE Java programs.

If it comes back with the same type of multiple pages syntax chiding, then you can skip the next section.

**If “javac” gives you “’javac’ is not recognized…” message**

It cannot see the compiler, so you have some work to do to change your PATH. If you know how to do this, go ahead and do it. If not, then here is what I did:

1. Open a MSDOS window and enter “cd \prog\*”. The “/Program files” should become the MSDOS directory.
2. Enter “cd java”, which will make “/Program files/java” the MSDOS directory.
3. Enter “dir” to see what existing subdirectories are. There should be several with today’s date. Note the most recent one that starts with “jdk”. For me this is “jdk1.7.0\_06”. Write this down.
4. Go to ‘System Properties’(may be under “Start”/”Control panel”/”System”).



1. On the left had side, click on ‘Advanced system settings’.



1. Click on ‘Environment Variables’.



1. Under ‘System variables’, scroll down and select Path. Then click edit.



1. Don’t delete anything in either of the boxes. At the end of the ‘Variable value’ box, type **;c:\Program Files\Java\jdk1.7.0\_07\bin.**
2. The semi-colon is required to separate the paths. The jdk1.7.0\_07 portion may be different for your Java edition.
3. Click ‘OK’ three times.



Now, to make sure it works, do this:

1. Open a new MSDOS window

2. Enter “javac” and make sure you get the pages of syntax chiding (rather than the “not recognized” message).

If you get the “not recognized” message, then I do not know how to help you. Seek professional help.

***Set up a Favorites link to Java documentation***

Create a link in your browser of choice to the Java documentation. (It is possible to download the documentation—useful if you code where the internet is not available.) Returning to our first step, when you get too:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

and see this:



Scroll down until you see this:



and pick that third download button, for “Java SE 7Documentation”. (I always download the resulting ZIP file to the c:/Program files/Java/jdk…. directory. This will put it in a subdirectory named “docs”. I then add a link to …/docs/api/index.html to my desktop for convenience.)

***Compile your first program***

Now that you have the Java compiler, let’s make sure you can compile things with it.

Do the usual startup:

1. Open a new MSDOS window

2. Maneuver to your subdirectory of choice using “cd” (or “mkdir” to make a directory).

Using your preferred text editor (making sure your files are saved in the equivalent of “text mode” so that extra formatting mess is not present), create a file called “HelloWorld.java” with the following lines:

class HelloWorld

{

 public static void main(String[] args)

 {

 System.out.println(" Hello, world!");

 }

}

Back at the MSDOS prompt, enter “javac HelloWorld.java”. It should return immediately with no comments.

Now enter “java HelloWorld” and see if it prints out “Hello, world!” like it should.

Here is what this simple code is doing, line by line:

class HelloWorld

This line just says you are defining a class (more about that later) named HelloWorld. Due to a trick it HAS to be in a file of the same name with “.java” appended.

{

This is the first of the many times that you will see the syntax that you SAY you are defining something (in this case the class HelloWorld) and you follow it with the actual definition between open and close braces. This is the open brace and the close brace is the last line in the file. I recommend that you put both of these on lines by themselves, indented to the same column, AND that you then indent two more spaces for what lies between them.

 public static void main(String[] args)

This states that you are defining the “main” routine, which tells Java that this is what you want it to run when (from the MSDOS prompt) you type “java HelloWorld”. The “public” says that you want to be able access this routine from outside the class itself. The “static” means that this routine is independent of a particular instance of the class, but a member of the class itself. (Clear as mud, I know, but just type it in.) The “void” is necessary because EVERY Java routine has to say what type of variable it returns when it ends, so you are telling it that nothing is returned.

 {

Following the previous pattern of “open brace”/”close brace” this brace indicates that you are beginning to define what the main return is supposed to do. The next-to-last line of the code is the close brace. (Again, not the recommended indention.)

 System.out.println(" Hello, world!");

This is the one and only line of coding. It tells the computer to print this particular message to the screen.

 }

}

These two are the previously mentioned close braces.

 ***Java commands***

Now that you know how to get a sample problem working, let me show you how to do simple things with it.

The first thing to remember (if you have not already learned it the hard way) is that Java is very particular about what files are named. It will NOT let you put the HelloWorld example program in a file named anything BUT HelloWorld.java. (That is because the file actually constructs a class named HelloWorld and the name of the file has to be the name of the class with “.java” appended.)

We are not going to dig deeply into classes for awhile, so you can get by for now by just making sure that the file name (with .java removed) corresponds exactly (caps and all) with the name in the line that begins with the word “class” in the text file.

***Write a comment***

1. If you enter double forward slash ANYWHERE, it makes the rest of the line a comment:

Example:

x=4; // I will need x later

Example:

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//

// Block of comments \*

// \*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[NOTE that the 3rd character position is BLANK. This is because I like the asterisks, but the character sequence //\* has a special meaning that I want to avoid.]

2. The entry /\* makes everything a comment until its mirror image (\*/) appears

 Example:

/\* This is a comment

 still commenting

...

 I have nothing more to say

\*/

I seldom use this (preferring the “comment block”, but use it sometimes to quickly “comment out” a block of code that I am temporarily replacing (but am not quite ready to delete).

***Continue a "line" of code on the next line***

Just do it. Until the compiler gets to a semicolon (;), it will keep adding to the line.

Example:

 double a=4.5;

is the same as:

 double

 a

 =

 4.5;

NOTE: It looks like I am putting blank characters at the beginning of the line, like old FORTRAN requires. This is not necessary; I just did it for readability. Your coding can start in the first column of the line.

***Most common data types***

In my coding, the most common variables that I use are:

int which makes a integer (There is a ‘long’ which can have bigger integers, but I seldom us it.)

double which makes a double precision real number (There is a single precision’float’ but I seldom use it)

String which holds names (alphanumeric constants)

***Create a single variable: int, double, String, boolean***

Variables must be declared before (or as) being used for 1st time

Examples:

 int a= 5;

 double b;

 b=5.456;

 String name="Bob";

 boolean isItSafe=true;

They only exist within their "scope"--a "braced" level, a method.

Example:

 double a=1.3;

 if(isItSafe)

 {

 double b=a+1.23;

 }

Outside the IF bracket, b does not exist (but a does)

If you want them to be used anywhere, declare them right up front (before main), then they belong to the whole thing.

Example:

 class Temp

 {

 double a;

 public static void main(String[] args)

 {

 a=34.;

***Print something to the screen***

Probably the cludgiest printing of any language I have ever seen! Printing is NOT built into the language.

Luckily you always have access to the System.out object, which has print and println commands (i.e., with and without a line feed)

 System.out.println(“...argument is a SINGLE string”);

Example:

 System.out.println(“Charley”);

 String name="Charley");

 System.out.println(name);

Luckier still, strings can be concatenated with a + sign to make longer strings.

 Example:

 System.out.println(name+" Sheen");

 System.out.print(name); //Notice lack of a line feed (ln)

 System.out.println(" Sheen");

 And luckier still still, you can concatenate another variable in, too.

Example:

 int countFilms=4;

 System.out.println(name+" Sheen has been in "+countFilms+" movies.");

Unfortunately, there has to be SOME String in there

Example:

 double a=4.;

 System.out.println(a); \\ Will NOT work since a is not a String

 System.out.println(" "+a); \\ Will work just fine, as will

 System.out.println(a+" ");

Special print "characters" --> They have to be inside quotation marks

 \n New line

 \t Tab

 \b Makes a beep noise

Example:

 System.out.println(name+"\n Sheen has been in \n\t"

 +countFilms+"\n movies!\b");

There are lots of other special characters that I don't use, all of which start with a backslash. So, backslashes usually won't print. If you WANT to print a backslash, use two -> \\ )

***Creating array of variables***

Arrays are indexed (i.e., elements referred to) using square brackets. These brackets (without anything in them) also are used to “declare” the arrays. (Which really just means you have defined what the array will be WHEN YOU GET AROUND TO CREATING IT.)

Actually creating arrays uses the "new" command in a clumsy way.

Example: Array of 45 doubles

 double[] a=new double[45];

 a[0]=1.;

 a[1]=2.3;

 ...

NOTE: Java arrays start with the element 0, not the element 1. This is a little screwy. You can, if you want, ignore this and just refer to element a[1] as the first one. BUT you have to realize that when you create an array a[45], it has 45 elements! So the last element is a[44]. (Of course, you can always create it one element larger, if you can remember to do this.)

The previous example started with a double-duty line that both DECLARED and CREATED the array. These could have been separated:

 double[] a;

 ...

 a=new double[45];

 a[0]=1.;

 a[1]=2.3;

 ...

***Special operations***

Java follows the C and C++ convention of avoiding repeating the variable name:

i++; is the same as i=i+1;

i--; is the same as i=i-1;

a\*=3.; is the same as a=a\*3.;

a/=3.; is the same as a=a/3.;

***IF-ELSE statements***

The easiest is the most straight-forward, as long as you remember the use of the braces:

if(*statement that evaluates to a boolean*)*statement you want to do*;

or

if(*statement that evaluates to a boolean*)

{

  *Multiple statements you want to do*;

}

Notice the syntax of comparisons:

 == Equal if(x == 1.0)…

 > Greater than

< Less than

<= Less than or equal

>= Greater than or equal

Comparing strings is more complicated:

String a=”university”;

if(a.compareTo(“college”) == 0) *statement;*

The “compareTo” is really an alphabetizing thing: It returns negative if “a” is earlier in the alphabet than the argument (“college”), zero if they are identical, positive if “a” is later in the alphabet than the argument.

***Loops***

I almost said “DO loops”, revealing my FORTRAN bias.

The most common loop I use has the strange syntax of for(…), where there are THREE statements inside the parentheses:

for(*initialization statement*;*boolean statement to continue while it is true*;*final statement*)

{

 … statements to do until the middle statement is FALSE

}

The rules are:

1. The first one is a statement that is run at the beginning of the WHOLE LOOP (so only once).

2. The second is an IF test used at the beginning of EACH PASS to see whether to continue.

3. The third is a statement that is run at the end of EACH PASS.

This seems (and is) a little kludgy, but you get used to it (and even forget the three rules!) if you always use the three statements in the usual do-loop way—using the three statements to modify an integer that COUNTS the number of passes through the loop.

Example: To loop the variable i from 0 to 44:

for(int i=0;i<45;i++)

{

}

x=1.;

Notice that you had to declare the variable i as an int. (Unless it already exists.) The variable i will only exist inside the loop in the example.

Most of my loops look exactly like the example above. Occasionally, though, I want to know the value of the variable i outside the loop (usually when I am testing to see which of the variables matches a particular condition) and use this syntax:

int i=0;

for(i=0;i<45;i++)

{

 if(student[i].compareTo(“Mary”) == 0)break;

}

// Now variable i holds the number of Mary’s entry in the

// “String[] student” list.

Extra statements of use INSIDE the loop:

break Unconditionally jumps to the next statement AFTER the loop (x=1.);

continue Skips the rest of THIS round, but continues with the next round.

Example: Break out of a loop if a condition is reached:

for(int i=0;i<45;i++)

{

 if(x[i] < 0.)break; // If this is true, I do not want to

 // continue the loop at all.

}

y=1.;

Example: Skip some elements, but continue:

for(int i=0;i<45;i++)

{

 if(x[i] < 0.)continue; // If this is true, go on to next

 System.out.println(“ The square root of ”+x[i]+ “ is ”+

 Math.sqrt(x[i]));

}

This is exactly the same as:

for(int i=0;i<45;i++)

{

 if(x[i] >= 0.)

 {

 System.out.println(“ The square root of ”+x[i]+ “ is ”+

 Math.sqrt(x[i]));

 }

}