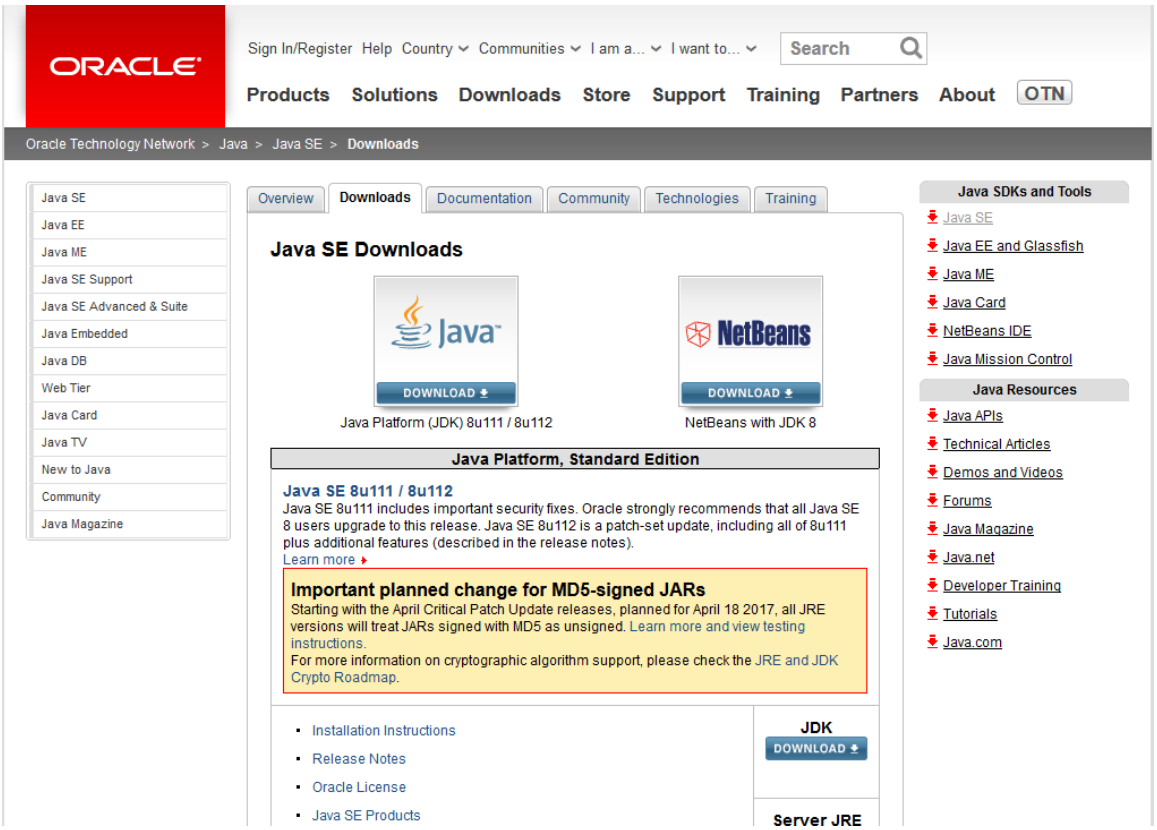
Get JAVA

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.

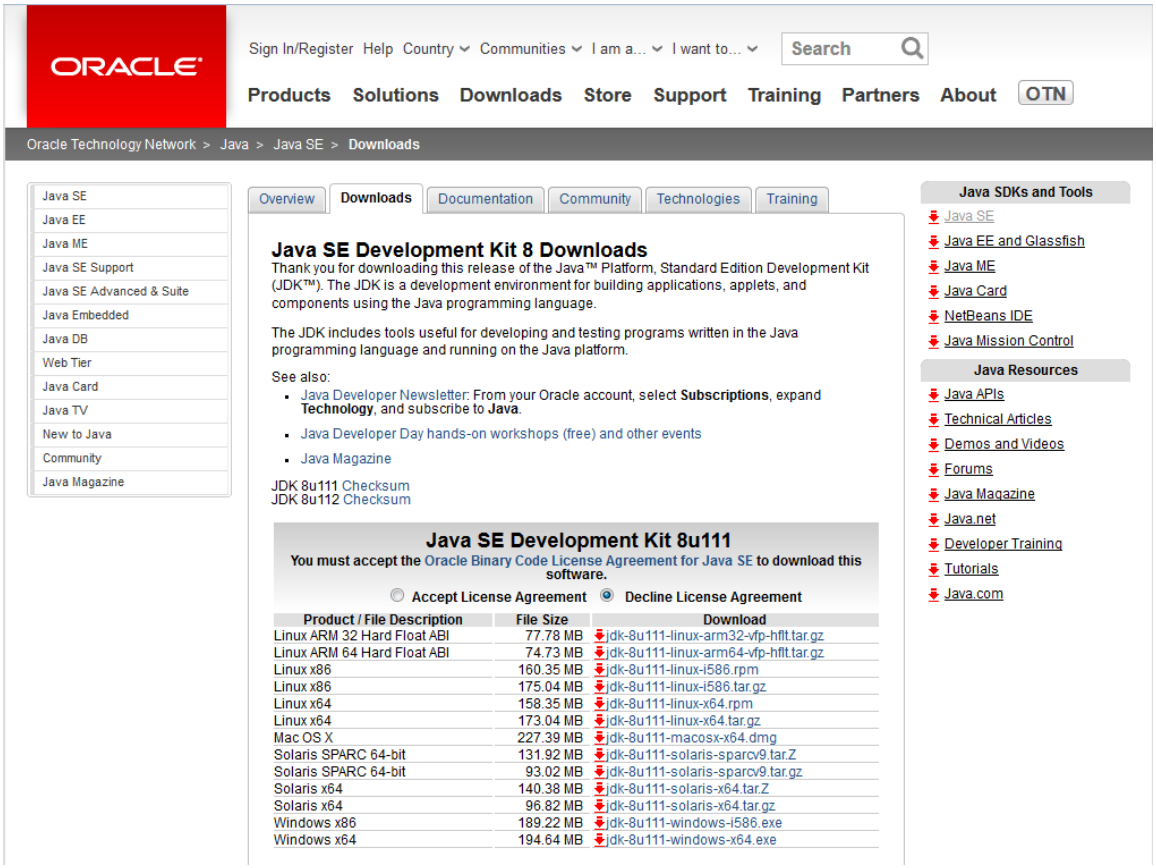
I will just tell you what I did (on January 10, 2017).

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

and saw this:



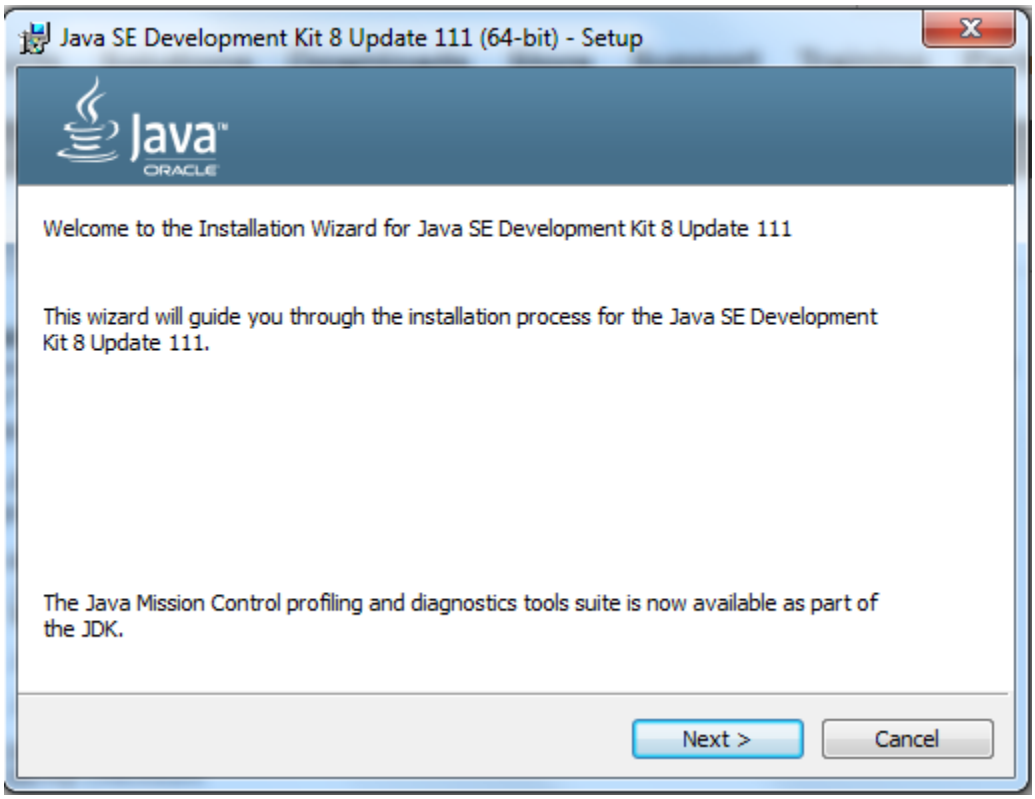
I clicked on the button under **Java SE Downloads** to download Java Platform (JDK) 8u111). Then I saw:



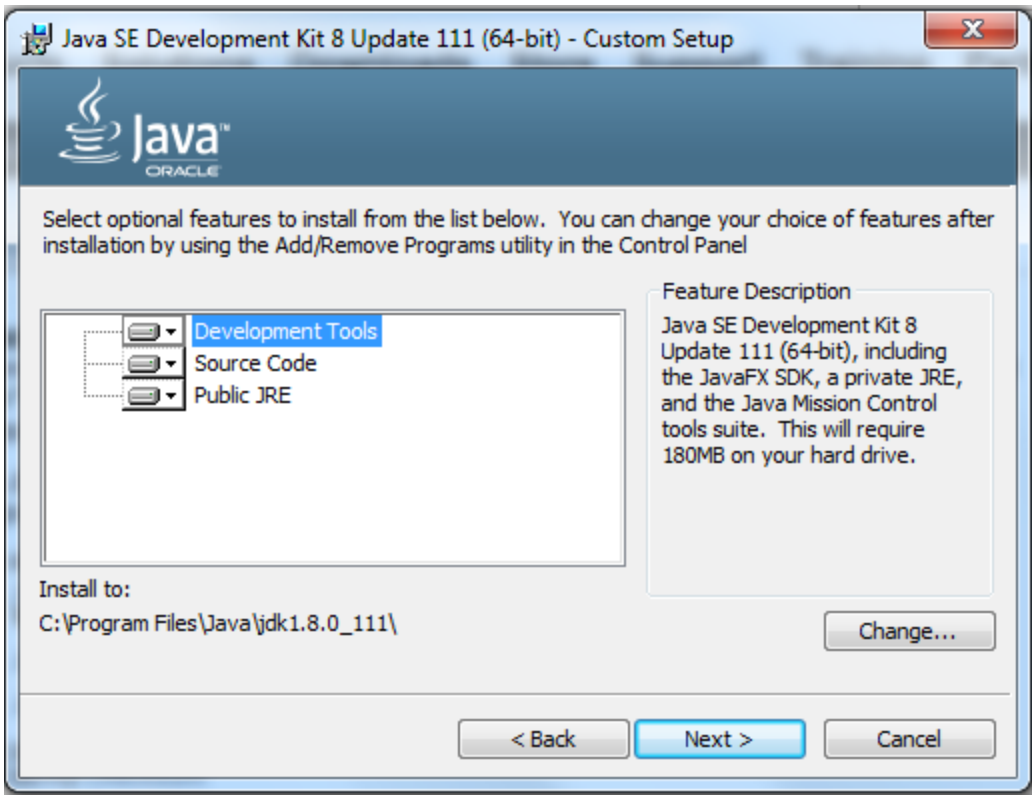
I accepted the License Agreement and picked the bottom one: …-x64.exe, but your computer may need a different one (check your system properties).

Depending on which web browser you use, you may be asked different things upon download. If it asks to 'Run' or 'Save' choose 'Run'. If it gives you a security warning, confirm that you trust the software or developer and wish to run it.

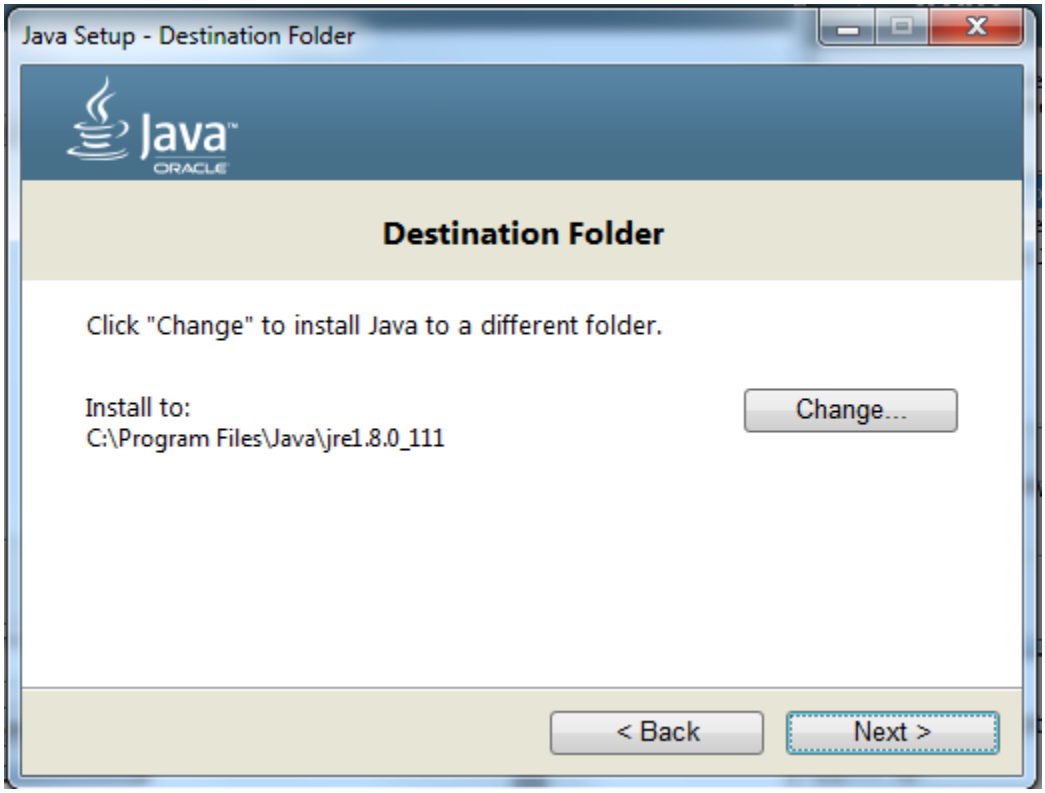
The Java Installation Wizard should pop up. There were a couple of windows I didn't capture ("verifying" or something), and then it said:



I picked "Next" for this. Then it gave me this window.

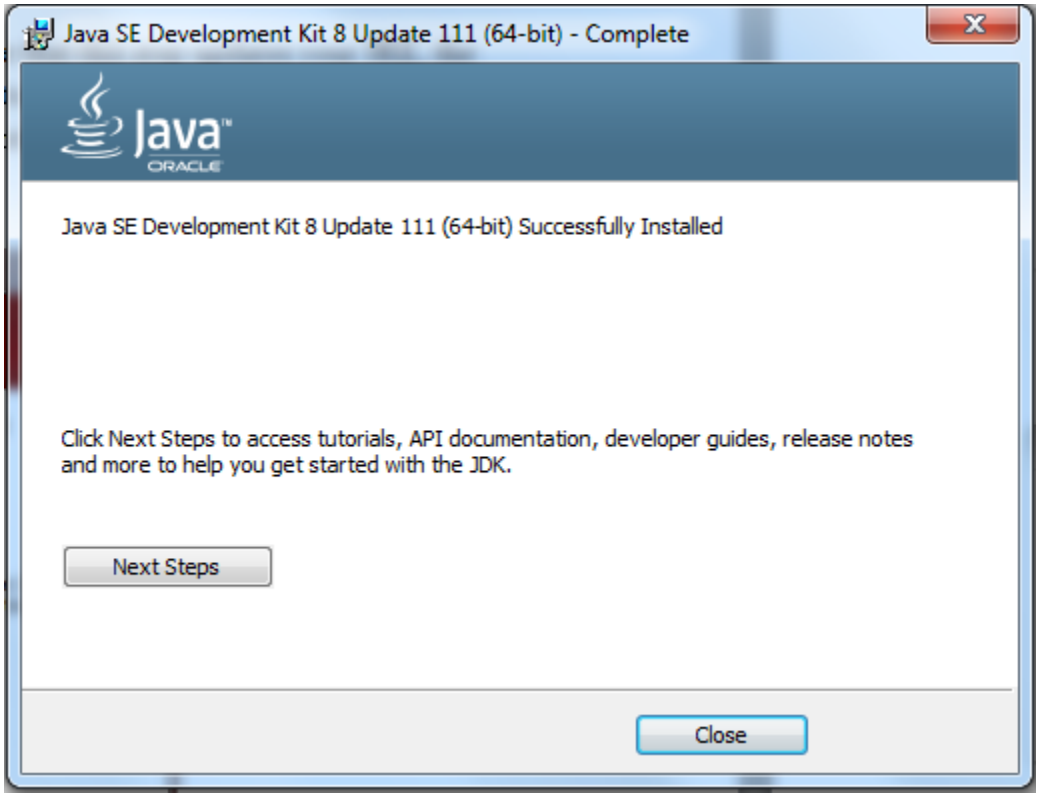


I didn't change anything and picked "Next" again. After some installation, this window popped up.



I clicked “Next” again without any changes. 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!

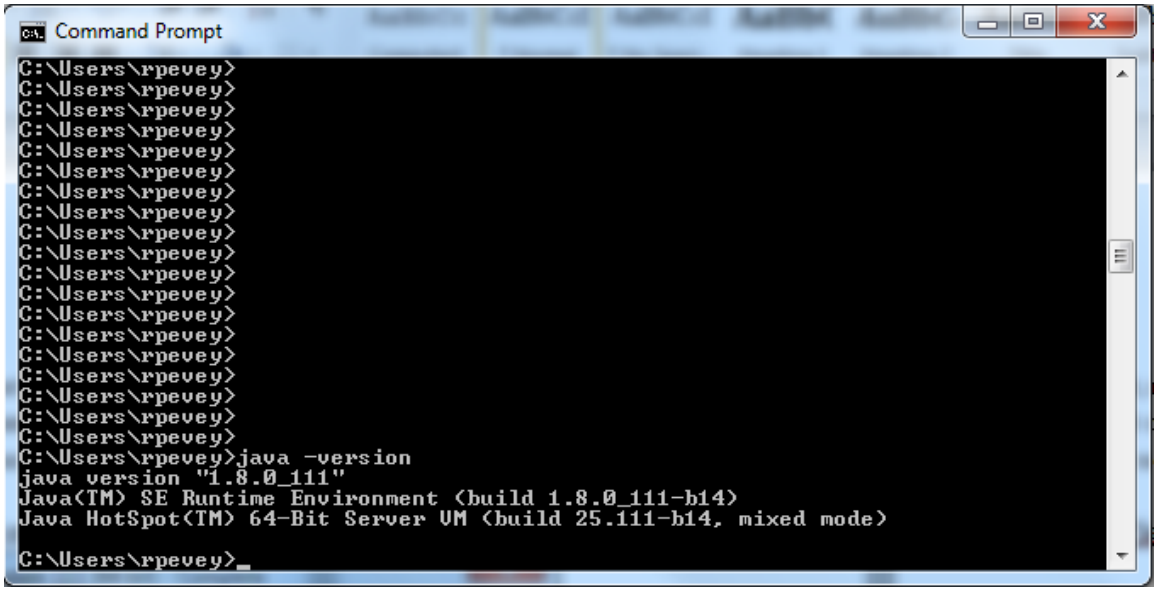
Then, after some gyrations, I got:



I clicked on “Close”. (Don’t know if I will regret that later

Assuming you followed my example, let’s see what you have. Open an MSDOS window (“Command Prompt”) and enter “java -version”. It should return with something like

what I got:



Notice the version number, 1.8.0_111.

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.

Now 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.

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

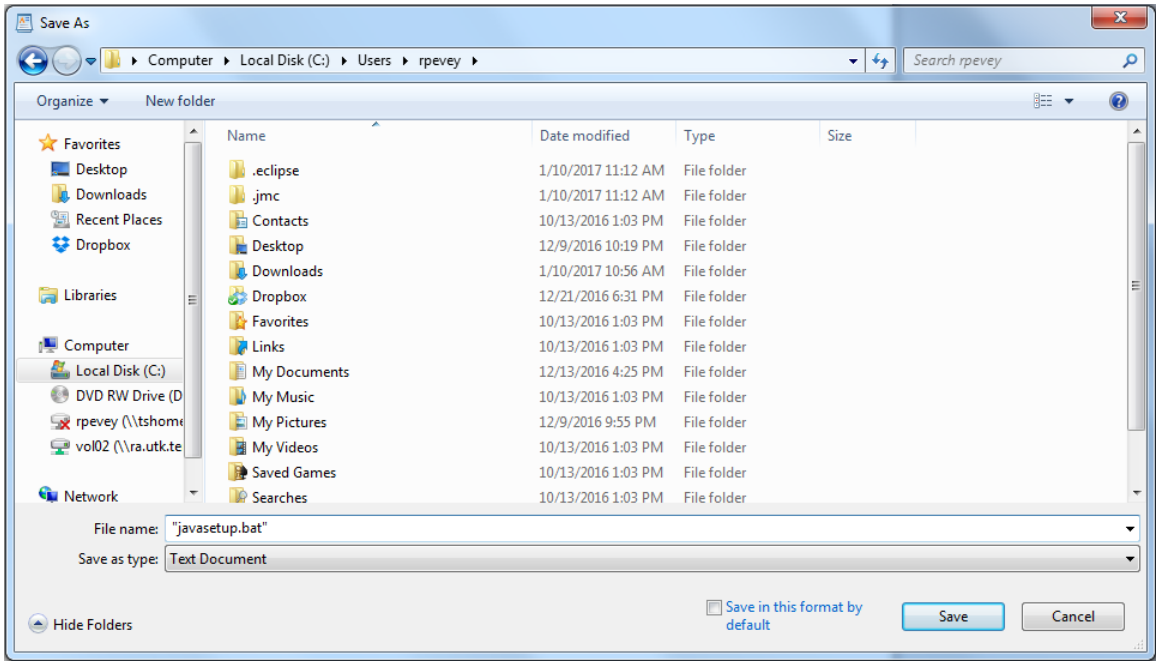
It cannot see the compiler, so you have some work to do to change your PATH. If you know how to do this, go ahead and do it. If not, then here is what I did:

1. Open a MSDOS window. Write down the name of the directory you are in. (For me it is \Users\rpevey.
2. Enter "cd \prog*". The "/Program files" should become the MSDOS directory.

3. Enter “cd java”, which will make “/Program files/java” the MSDOS directory.
4. Enter “dir” to see what existing subdirectories are. There should be a couple with today’s date. Note the most recent one that starts with “jdk”. For me this is “jdk1.8.0_111”. Write this down.
5. Enter “write” to open a WordPad session (or use your editor of choice) to create and open a new file.
6. Enter the following lines to make it look like this:

   @echo off
   path=%PATH%;c:\Program files\Java\jdk1.8.0_111\bin;
   DOSKEY

   Be sure to replace the “jdk1.8.0_111” with the name you wrote down in step 4.
7. Save the file in the directory you wrote down in Step 1, as a TEXT file with the name “javasetup.bat”. INCLUDE THE QUOTATION MARKS in the name. It should look something like this:



Press “Save”.

Now, to make sure it works, do this:

1. Open a new MSDOS window
2. Enter “javasetup”.
3. Enter “javac” and make sure you get the pages of syntax chiding (rather than the “not recognized” message).

Compile your first program

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

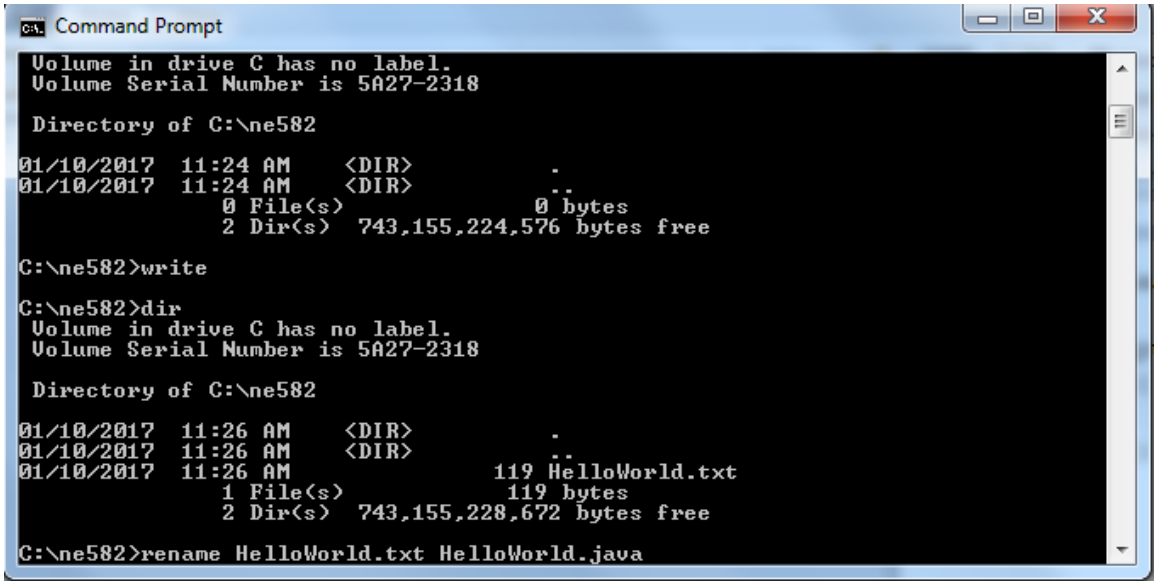
Do the usual startup:

1. Open a new MSDOS window
2. Enter "javasetup".
3. 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!");
  }
}
```

(NOTE: I actually had to save it as a text file, which had the default name HelloWorld.txt, which I then had to rename using the rename command:



)

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 (for us this is just a program) named HelloWorld. Due to a Java requirement 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 actuall 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 as the line before, 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 routine 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.

We are not going to dig deeply into classes this course, so you can get 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.

What this line does is to define a variable named a that is a double precision real number.

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 use 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)
boolean which can only have the values true or false.

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", which is generally their "braced" level..

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 class object, which has print and println commands (i.e., printing with and without a line feed afterwards).

```
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, you can concatenate another variable in, too. Java will just convert it to a String (if there is another String present).

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)… NOTE: There are TWO equal signs.
> 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;
```

Loops

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

```
for(initialization statement;boolean statement to
continue;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.

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;
`continue` Skips the rest of THIS round of the loop, 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.
}
x=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]));
  }
}
```

The Math class

Besides the System class (that has member out that is the computer screen), the only other pre-written Java class that I regularly use is the Math class. On the previous page, I used it to compute a square root. There are other functions available, all of which use the syntax:

Math.xxxx(argument1,argument2,…)

Some of the ones you might have occasion to use (which I found at https://docs.oracle.com/javase/7/docs/api/ under "Math") are given on the next couple of pages. (We will use "random()" a lot.) The "static" out front just means that any program can use them at any time.

Modifier and Type	Method and Description
static double	abs(double a) Returns the absolute value of a double value.
static float	abs(float a) Returns the absolute value of a float value.
static int	abs(int a) Returns the absolute value of an int value.
static long	abs(long a) Returns the absolute value of a long value.
static double	acos(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
static double	asin(double a) Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	atan(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	atan2(double y, double x) Returns the angle *theta* from the conversion of rectangular coordinates (x, y) to polar coordinates (r, *theta*).
static double	cbrt(double a) Returns the cube root of a double value.
static double	ceil(double a) Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integ
static double	copySign(double magnitude, double sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static float	copySign(float magnitude, float sign) Returns the first floating-point argument with the sign of the second floating-point argument.
static double	cos(double a) Returns the trigonometric cosine of an angle.
static double	cosh(double x) Returns the hyperbolic cosine of a double value.
static double	exp(double a) Returns Euler's number *e* raised to the power of a double value.
static double	expm1(double x) Returns e^x -1.
static double	floor(double a) Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer.
static int	getExponent(double d) Returns the unbiased exponent used in the representation of a double.
static int	getExponent(float f) Returns the unbiased exponent used in the representation of a float.
static double	hypot(double x, double y) Returns sqrt(x^2 +y^2) without intermediate overflow or underflow.

`static double`	`IEEEremainder(double f1, double f2)` Computes the remainder operation on two arguments as prescribed by the IEEE 754 standard.
`static double`	`log(double a)` Returns the natural logarithm (base e) of a `double` value.
`static double`	`log10(double a)` Returns the base 10 logarithm of a `double` value.
`static double`	`log1p(double x)` Returns the natural logarithm of the sum of the argument and 1.
`static double`	`max(double a, double b)` Returns the greater of two `double` values.
`static float`	`max(float a, float b)` Returns the greater of two `float` values.
`static int`	`max(int a, int b)` Returns the greater of two `int` values.
`static long`	`max(long a, long b)` Returns the greater of two `long` values.
`static double`	`min(double a, double b)` Returns the smaller of two `double` values.
`static float`	`min(float a, float b)` Returns the smaller of two `float` values.
`static int`	`min(int a, int b)` Returns the smaller of two `int` values.
`static long`	`min(long a, long b)` Returns the smaller of two `long` values.
`static double`	`nextAfter(double start, double direction)` Returns the floating-point number adjacent to the first argument in the direction of the second argument.
`static float`	`nextAfter(float start, double direction)` Returns the floating-point number adjacent to the first argument in the direction of the second argument.
`static double`	`nextUp(double d)` Returns the floating-point value adjacent to `d` in the direction of positive infinity.
`static float`	`nextUp(float f)` Returns the floating-point value adjacent to `f` in the direction of positive infinity.
`static double`	`pow(double a, double b)` Returns the value of the first argument raised to the power of the second argument.
`static double`	`random()` Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.
`static double`	`rint(double a)` Returns the `double` value that is closest in value to the argument and is equal to a mathematical integer.
`static long`	`round(double a)` Returns the closest `long` to the argument, with ties rounding up.
`static int`	`round(float a)` Returns the closest `int` to the argument, with ties rounding up.

`static double`	**`rint`**`(double a)` Returns the `double` value that is closest in value to the argument and is equal to a mathematical integer.
`static long`	**`round`**`(double a)` Returns the closest `long` to the argument, with ties rounding up.
`static int`	**`round`**`(float a)` Returns the closest `int` to the argument, with ties rounding up.
`static double`	**`scalb`**`(double d, int scaleFactor)` Return $d \times 2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the double value set.
`static float`	**`scalb`**`(float f, int scaleFactor)` Return $f \times 2^{scaleFactor}$ rounded as if performed by a single correctly rounded floating-point multiply to a member of the float value set.
`static double`	**`signum`**`(double d)` Returns the signum function of the argument; zero if the argument is zero, 1.0 if the argument is greater than zero, -1.0 if the argument is less than zero.
`static float`	**`signum`**`(float f)` Returns the signum function of the argument; zero if the argument is zero, 1.0f if the argument is greater than zero, -1.0f if the argument is less than zero.
`static double`	**`sin`**`(double a)` Returns the trigonometric sine of an angle.
`static double`	**`sinh`**`(double x)` Returns the hyperbolic sine of a `double` value.
`static double`	**`sqrt`**`(double a)` Returns the correctly rounded positive square root of a `double` value.
`static double`	**`tan`**`(double a)` Returns the trigonometric tangent of an angle.
`static double`	**`tanh`**`(double x)` Returns the hyperbolic tangent of a `double` value.
`static double`	**`toDegrees`**`(double angrad)` Converts an angle measured in radians to an approximately equivalent angle measured in degrees.
`static double`	**`toRadians`**`(double angdeg)` Converts an angle measured in degrees to an approximately equivalent angle measured in radians.
`static double`	**`ulp`**`(double d)` Returns the size of an ulp of the argument.
`static float`	**`ulp`**`(float f)` Returns the size of an ulp of the argument.