Hello,

Welcome to AP Computer Science A. This summer assignment will be graded and will be due at the end of the first week of school. Don't worry, this won't be too harsh of an assignment (I promise. ☺) This assignment is simply to introduce you to the basics of Java. I will be reviewing all this information when you start class.

To Begin With: You need to download Java, Jgrasp, and Junit

A. Download Java first: You need the **Java SE Development Kit 8u211**. Go to the link below, scroll down to first block as titled above and click "Accept License Agreement."
https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Then click the bottom link to download the Windows x64:

Windows x64	215.29 MB	jdk-8u211-windows-x64.exe

B. Next Download jGrasp: Follow the link below:
https://spider.eng.auburn.edu/user-cgi/grasp/grasp.pl?;dl=download_jgrasp.html
Then go to the first box as shown below and select jGrasp.exe if you have windows or jGrasp.pkg if you are using a Mac.

jGRASP 2.0.5_05 (March 12, 2019) - requires Java 1.6 or higher

jGRASP exe **Windows (Vista or Higher)**: self-extracting executable (5,896,808 bytes).

jGRASP pkg **macOS**: pkg install file (requires admin access to install) (6,903,572 bytes).

jGRASP zip **Linux**, **UNIX**, and **other systems**: zip file (7,148,344 bytes).

TIME TO BEGIN!!!! ☺

Topic 1: "Hello World"

Program Skeleton:
Enter the following program skeleton.

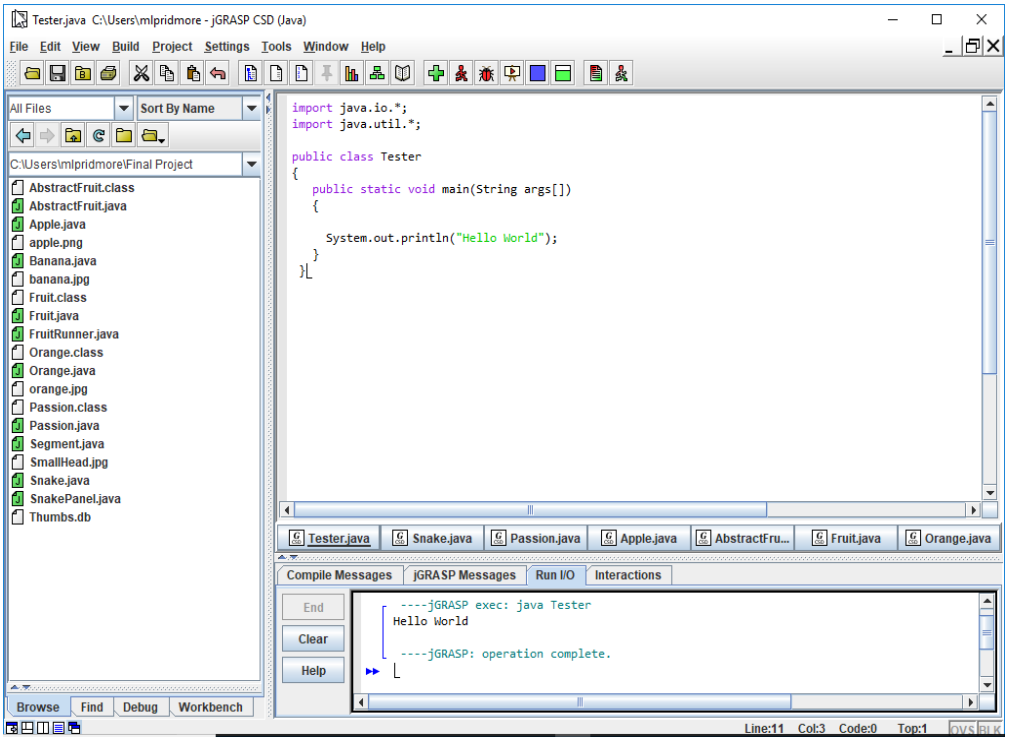
```
public class Tester
{
        public static void main(String args [ ])  //We could put any comment here
        {
                System.out.println("Hello World");
        }
}
```

Select File -> Save As -> Pick a place to save your programs and the file's name MUST match the class name: In this case, "Tester" should be the file name.

Let's run this very short program.

At the top of your window is a tool bar: compile (prepare it to run use the green plus to compile "+"), and then run (execute, the red running figure).

This is what your run should look like:



At this point don't worry about what any of this means. It's just something we must do every time. Soon we will learn the meaning of all of this. For now it's just the skeleton that we need for a program.

Remarks: Notice the rem (remark) above that starts with //. You can put remarks anywhere in the program without it affecting program operation. Remarks are also called comments or notes.

Printing:
System.out.println("Hello world"); is how we get the computer to printout something.

Notice the trailing semicolon. Most lines of code are required to end in a semicolon.
Now, let's explore……

Now try putting in some other things in the *println* parenthesis above. Each time recompile and run the program:
1. "Peter Piper picked a peck of pickled peppers."
2. "I like computer science."
3. 25/5
4. 4 / 7.0445902
5. 13 * 159.56

Show the output for 1 – 5 here:
1.

2.

3.

4.

5.

Two *println*s for the price of one:
Next, modify your program so that the *main* method looks as follows:

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

Run this and note that it prints :

```
Hello world
Hello again
```

Printing "Sideways":
Now remove the *ln* from the first *println* as follows:

```
public static void main(String args[])
{
System.out.print("Hello world");
System.out.println("Hello again");
}
```

Run this and note that it prints:

```
Hello worldHello again
```

Here are the rules concerning *println* and *print*:

• *System.out.println( )* completes printing on the current line and pulls the print position down to the next line where any subsequent printing continues.
• *System.out.print( )* prints on the current line and stops there. Any subsequent printing continues from that point.

An in-depth look at rems:
Let's take a further look at rems. Consider the following program (class) in which we wish to document ourselves as the programmer, the date of creation, and our school:

```
public class Tester
{
//Programmer: Kosmo Kramer
//Date created: Sept 34, 1492
//School: Charles Manson High School; Berkley, Ca
        public static void main(String args[])
        {
                System.out.println("Hello again");
        }
}
```

Block rems:
It can get a little tedious putting the double slash rem-indicator in front of each line, especially if we have quite a few remark lines. In this case we can "block rem" all the comment lines as follows:

```
public class Tester
{
/*Programmer: Kosmo Kramer
Date created: Sept 34, 1492
School: Charles Manson Junior High; Berkley, Ca*/
        public static void main(String args[])
        {
                System.out.println("Hello again");
        }
}
```

Notice we use /* to indicate the start of the block and */ for the end.
Everything between these two symbols is considered to be a remark and will be ignored by the computer when compiling and running.

Assignment for Topic 1:

Write code to produce the output below. Write what you entered in the program skeleton provided. Also include remarks above *public class Tester* that identifies you as the author along with the date of creation of this program:

```
//Author: Charles Cook
//Date created: Mar 22, 2005
public class Tester
{
        public static void main(String args[])
        {

        …

        }
}
```

Supply code in the place of … that will produce the following printout:

```
From: Bill Smith
Address: Dell Computer, Bldg 13
Date: April 12, 2005
To: Jack Jones
Message: Help! I'm trapped inside a computer!
```

Topic 2: Variable Types (String, int, double)

Three variable types:
(A good way to learn the following points is to modify the code of the "Hello World" program according to the suggestions below.)

1. ***String***….used to store things in quotes….like "Hello world"
Sample code:

```
public static void main(String args[])
{
        String s = "Hello cruel world";
        System.out.println(s);
}
```

2. ***int*** ….used to store integers (positive or negative)
Sample code:

```
public static void main(String args[])
{
        int age = 59;
        System.out.println(age);
}
```

With the advent of Java 7.0, numbers that were previously very awkward and difficult to read can now be entered with underscore separators.
For example,
int bigNum = 1389488882; can now be entered as
int bigNum = 1_389_488_882;
Unfortunately, commas still cannot be used as separators.

3. ***double*** ….used to store "floating point" numbers (decimal fractions). *double* means "double precision".
Sample code:

```
public static void main(String args[])
{
        double d = -137.8036;
        System.out.println(d);
        d = 1.45667E23; //Scientific notation…means 1.45667 X 10^23
}
```

Declaring and initializing:
When we say something like

```
double x = 1.6;
```

we are really doing **two** things at once. We are **declaring** *x* to be of type *double* **and** we are **initializing** *x* to the value of 1.6. All this can also be done in **two** lines of code (as shown below) instead of one if desired:

```
        double x; //this declares x to be of type double
        x = 1.6; //this initializes x to a value of 1.6
```

What's legal and what's not:

```
int arws = 47.4; //illegal, won't compile since a decimal number cannot "fit" into an
//integer variable.

double d = 103; //legal…same as saying the decimal number 103.0
```

Rules for variable names:
Variable names must begin with a letter (or an underscore character) and cannot contain spaces. The only "punctuation" character permissible inside the name is the underscore ("_"). Variable names cannot be one of the reserved words (key words…see Appendix A) that are part of the Java language.

Legal names	**Illegal names**	
Agro	139	can't lead with a number
D	139Abc	can't lead with a number
d31	fast One	can't have a space between words
hoppergee	class	this word is a reserved word used in Java code
hopper_gee	slow.Sally	can't use periods, commas or dashes – can only use underscore
largeArea	double	this word is a reserved word representing a type of numeric value
goldNugget	gold;Nugget	can't use periods, commas, dashes, semicolons
	hopper-gee	can't use periods, commas or dashes – can only use underscore

Variable naming conventions:
It is traditional (although not a hard and fast rule) for variable names to start with a lower case letter. If a variable name consists of multiple words, combine them in one of two ways:

bigValue… jam everything together. First word begins with a small letter and subsequent words begin with a capital.

big_value… separate words with an underscore.

Assignment for Topic 2:

1. In your Tester program type:
   double 1Hat = 5;
   Compile this and write what happens:

2. Key in the following declaration and initialization:
   double hop = 7.45;
   System.out.println(hop);

   Compile then run this and write what the output is:

3. Key in the following declaration and initialization:
   int happyDay = 7.45;
   System.out.println(happyDay);

   Compile then run this and write what happens:

4. Key in the following declaration and initialization:
   int cat21 = 23;
   System.out.println(cat21);

   Compile then run this and write what the output is:

Topic 3: Simple String Operations

In this lesson we will learn just a few of the things we can do with *String*s.

Concatenation:
First and foremost is **concatenation.** We use the plus sign, +, to do this. For example:
String mm = "Hello";
String nx = "good buddy";
String c = mm + nx;
System.out.println(c); //prints **Hellogood buddy**…notice no space between o & g

The above code could also have been done in the following way:
String mm = "Hello";
String nx = "good buddy";
System.out.println(mm + " " + nx); //prints **Hello good buddy**…notice the space

We could also do it this way:
System.out.println("Hello" + " good buddy"); // prints **Hello good buddy**

The *length* method:
Use the ***length( )*** method to find the number of characters in a *String*:

String theName = "Donald Duck";
int len = theName.length( );
System.out.println(len); //prints 11…notice the space gets counted

Right now we don't see much value in this length thing…just wait! Seriously!!

A piece of a *String (substring*):

We can pick out a piece of a *String*…**substring**

String myPet = "Sparky the dog";
String smallPart = myPet.substring(4);
System.out.println(smallPart); //prints **ky the dog**

Why do we get this result? The various characters in a *String* are numbered starting on the left with 0. These numbers are called **indices**. (Notice the spaces are numbered too.)

S p a r k y t h e d o g … so now we see that the 'k' has **index** 4 and we go from
0 1 2 3 4 5 6 7 8 9 10 11 12 13 k all the way to the end of the string to get "ky the dog".

A more useful form of *substring*:
But wait! There's another way to use ***substring***

String myPet = "Sparky the dog";
String smallPart = myPet.substring(4, 12);
System.out.println(smallPart); //prints **ky the d**

How do we get ***ky the d***? Start at *k*, the 4th index, as before. Go out to the 12th index, 'o' in this case and pull back one notch. That means the last letter is *d*.

Conversion between lower and upper case:

toLowerCase converts all characters to lower case (small letters)

String bismark = "Dude, where's MY car?";
System.out.println(bismark.toLowerCase( )); // prints **dude, where's my car?**

toUpperCase converts all characters to upper case (capital letters)

System.out.println("Dude, where's My car?".toUpperCase( ));
//prints **DUDE, WHERE'S MY CAR?**

Note: *length*, *substring*, *toLowerCase*, and *toUpperCase* are all **methods** of the *String* class. There are other methods we will learn later. This will all make sense as we go through the class. For now become familiar with the terminology. No worries.

Concatenating a *String* and a numeric:

It is possible to concatenate a *String* with a numeric variable as follows:
int x = 27;
String s = "Was haben wir gemacht?"; //German for "What have we done?"
String combo = s + " " + x;
System.out.println(combo); //prints **Was haben wir gemacht? 27**

Escape sequences:

How do we force a **quote** character (") to printout…. or, to be part of a *String*. Use the **escape sequence**, \", to print the following (note escape sequences always start with the \ character…see Appendix B for more on escape sequences):

```
What "is" the right way?
```

String s = "What \"is\" the right way?";
System.out.println(s); //prints **What "is" the right way?**

Another **escape sequence,** \n, will create a **new line** (also called **line break**) as shown below:

String s = "Here is one line\nand here is another.";
System.out.println(s);
Prints the following:

```
Here is one line
and here is another.
```

The **escape sequence**, \\, will allow us to print a backslash within our *String*. Otherwise, if we try to insert just a single \ it will be interpreted as the beginning of an escape sequence.

System.out.println("Path = c:\\nerd_file.doc");
Prints the following:

```
Path = c:\nerd_file.doc
```

The **escape sequence**, \t, will allow us to "tab" over. The following code tabs twice.
System.out.println("Name:\t\tAddress:");
Prints the following:

```
Name:		Address:
```

Assignment for Topic 3: (Practice these in your Tester class but write what you did here.)

1. In your Tester class, key in code in which a *String* variable *s* contains "The number of rabbits is". An **integer** variable *argh* has a value of 129. Concatenate these variables into a *String* called *report*.
Then print *report*. The printout should yield:
`The number of rabbits is 129.`
Note that we want a period to print after the 9.

Your code here:

2. What will be the value of *c*? Write out the String and label the indices. Refer to the first page for how to do this from the example "Sparkythedog…"
String c;
String m = "The Gettysburg Address";
c = m.substring(4);

String with the indices noted:

Value of c:

3. What will be the value *c*? Write out the String and label the indices. Refer to the first page for how to do this from the example "Sparkythedog…"

String b = "Four score and seven years ago,";
String c = b.substring(7, 12);

String with the indices noted:

Value of c:

4. What is the value of *count*? Write out the String and label the indices.

```
int count;
String s = "Surface tension";
count = s.length( );
```

String with the indices noted:

Value of count:

5. Write code that will look at the number of characters in *String m = "Look here!";* and then print

```
"Look here!" has 10 characters.
```

Use the *length( )* method to print the 10 ….you must also force the two quotes to print.

Your code here:

The assignment operator:
The assignment operator is the standard equal sign (**=**) and is used to "assign" a value to a variable.

int i = 3; // Ok,…assign the value 3 to i. **Notice the direction of data flow.**

3 = i; // **Illegal!** Data never flows this way!

double p;
double j = 47.2;
p = j; // **assign** the value of j to p. Both p and j are now equal to 47.2

Multiple declarations:
It is possible to declare several variables on one line:

double d, mud, puma; //the variables are only declared

double x = 31.2, m = 37.09, zu, p = 43.917; //x, m, & p declared and initialized
// zu is just declared

Fundamental arithmetic operations:
The basic arithmetic operation are +, -, * (multiplication), / (division), and % (modulus).

Modulus is the strange one. For example, System.out.println(5%3); will print 2. This is because when 5 is divided by 3, the **remainder** is 2. **Modulus gives the remainder.** Modulus also handles negatives. The answer to *a*%*b* always has the same sign as *a*. The sign of *b* is ignored.

PEMDAS:
The algebra rule, PEMDAS, applies to computer computations as well. (PEMDAS stands for the order in which numeric operations are done. P = parenthesis, E = exponents, M = multiply, D = divide, A = add, S = subtract. Actually, M and D have equal precedence, as do A and S. For equal precedence operation, proceed from left to right. A mnemonic for PEMDAS is, "Please excuse my dear Aunt Sally"… See Appendix H for the precedence of all operators.)

System.out.println(5 + 3 * 4 –7); //10

System.out.println(8 – 5*6 / 3 + (5 –6) * 3); //-5

Not the same as in Algebra:
An unusual assignment….consider the following:

count = count +3; //this is illegal in algebra; however, in computer science it
//means the **new** count equals the **old** count + 3.

int count =15;
count = count + 3;
System.out.println(count); //**18**

Increment and Decrement:
The increment operator is ++, and it means to add one. The decrement operator is --, and it means to subtract one:
x++; means the same as x = x +1;
x--; means the same as x = x – 1;
x++ is the same as ++x (the ++ can be on **either** side of x)
x-- is the same as --x (the -- can be on **either** side of x)
int y = 3;
y++;
System.out.println(y); //**4**

Compound operators:

Syntax Example	Simplified meaning
a. +=	
x += 3;	x = x + 3;
b. -=	
x -= y - 2;	x = x – (y - 2);
c. *=	
z*= 46;	z = z * 46;
d. /=	
p/= x-z;	p = p / (x-z);
e. %=	
j%= 2	j = j%2;

Code Examples

int g = 409;
g += 5;
System.out.println(g); //**414**

double d = 20.3;
double m =10.0;
m*=d –1;
System.out.println(m); //**193.0**

The whole truth:
Actually, the full truth was not told above concerning *x*++. It does not always have the same effect as does ++*x*. Likewise, *x*-- does not always have the same effect as does --*x*.
x++ increments *x* **after** it is used in the statement.
++*x* increments *x* **before** it is used in the statement.

Similarly,
x-- decrements *x* **after** it is used in the statement.
--*x* decrements *x* **before** it is used in the statement.

Code Examples

int q = 78;
int p = 2 + q++;
System.out.println("p = " + p + ", q = " + q); //**p = 80, q = 79**

int q = 78;
int p = ++q + 2;
System.out.println("p = " + p + ", q = " + q); //**p = 81, q = 79**

Integer division truncation:
When dividing two integers, the fractional part is truncated (thrown away) as illustrated by the following:

```
int x = 5;
int y = 2;
System.out.println(x / y); //Both x and y are integers so the "real" answer of 2.5
//has the fractional part thrown away to give 2
```

Assignment for Topic 4: Key in the code into your Tester class to see the output.

1.
```
int h = 103;
int p =5;
System.out.println(++h + p);
System.out.println(h);
```

Your output here:

2.
```
int h = 103;
int p =5;
System.out.println(p + h++);
System.out.println(h);
```

Your output here:

What is the difference between this output and the output from #1 (Explain):

3.
```
int a = 100;
int b = 200;
b/=a;
System.out.println(b + 1);
```

Your output here:

4. Write a **single** line of code that uses the compound operator, -=, to subtract *p-30* from the integer value *v* and store the result back in *v*.

Your code here:

5. Write a single line of code that does the same thing as #6 but without using - =

Your code here:

6. int p = 40;
int q = 4;
System.out.println(2 + 8 * q / 2 - p); \\Do the calculation by hand first and see if it matches the output when run

Your calculation (show work) and output here:

7. int sd = 12;
int x = 4;
System.out.println(sd%(++x)); \\Do the calculation by hand first and see if it matches the output when run

Your calculation (show work) and output here:

System.out.println(x);

Your output here:

8. int m = 36;
int j = 5;
m = m / j; // new m is old m divided by j
System.out.println(m);

Your output here:

9. System.out.println(3/4 + 5*2/33 –3 +8*3); \\Do the calculation by hand first and see if it matches the output when run

Your calculation (show work) and output here:

Your output here:

10. Write a statement (code) that stores the remainder of dividing the variable *i* by *j* in a variable named *k*.

Your code here:

Topic 5: Mixed Data Types, Casting, and Constants

So far we have looked mostly at simple cases in which all the numbers involved in a calculation were either **all** integers or **all** *doubles*. Here, we will see what happens when we **mix** these types in calculations.

Java doesn't like to lose data:
Here is an important principle to remember: Java **will not** normally store information in a variable if in doing so it would **lose** information. Consider the following two examples:

1. An example of when we would **lose** information:
double d = 29.78;
int i = d; //**won't compile** since i is an integer and it would have to chop-off
// the .78 and store just 29 in i….thus, it would **lose** information.

There is a way to make the above code work. We can **force** compilation and therefore result in 29.78 being "stored" in *i* as follows (actually, just 29 is stored since *i* can only hold integers):

int i = (int)d; //(int) "**casts**" *d* as an integer… It converts *d* to integer form.

2. An example of when we would **not** lose information:
int j = 105;
double d = j; //legal, because no information is lost in storing 105 in the
// double variable d.

The most precise:
In a math operation involving **two different data types**, the result is given in terms of the **more precise** of those two types…as in the following example:

int i = 4;
double d = 3;
double ans = i/d; //ans will be 1.33333333333333…the result is double precision
20 + 5 * 6.0 returns a *double*. The 6.0 might look like an integer to us, but because it's written with a decimal point, it is considered to be a floating point number…a *double*.

Some challenging examples:

What does 3 + 5.0/2 + 5 * 2 – 3 return? **12.5**
What does 3.0 + 5/2 + 5 * 2 – 3 return? **12.0**
What does (int)(3.0 + 4)/(1 + 4.0) * 2 – 3 return? **-.2**

Don't be fooled:
Consider the following two examples that are very similar…but have different answers:

double d = (double)5/4; //same as 5.0 / 4…(double) only applies to the 5
System.out.println(d); //**1.25**

```
int j = 5;
int k = 4;
double d = (double)(j / k); //(j / k) is in its own little "world" and performs
//integer division yielding 1 which is then cast as
//a double, 1.0
System.out.println(d); //1.0
```

Constants:
Constants follow all the rules of variables; however, once initialized, they **cannot be changed**. Use the keyword ***final*** to indicate a constant. Conventionally, constant names have all capital letters. The rules for legal constant names are the same as for variable names. Following is an example of a constant:

```
final double PI = 3.14159;
```

The following illustrates that constants can't be changed:

```
final double PI = 3.14159;
PI = 3.7789; //illegal
```

When in a method, constants may be initialized after they are declared.

```
final double PI; //legal
PI = 3.14159;
```

Constants can also be of type *String*, *int* and other types.

```
final String NAME= "Peewee Herman";
final int LUNCH_COUNT = 122;
```

The real truth about compound operators:

In the previous lesson we learned that the compound operator expression *j+= x;* was equivalent *to j = j + x;*. Actually, for **all compound operators** there is also an **implied cast** to the type of *j*. For example, if *j* is of type *int*, the real meaning of *j+= x;* is:

```
j = (int)(j + x);
```

Assignment for Topic 5:

1. What's wrong, if anything, with the following code in the *main* method?

```
final double Area;
Area = 203.49;
```

What's wrong with the above code:

Write the correct way to declare and initialize a Constant:

2. int cnt = 27.2;
System.out.println(cnt);

Write the output here:

3. double d = 78.1;
int fg = (int)d;
System.out.println(fg);

Write the output here:

4. The following code stores a 20 in the variable *j*:
double j = 61/3;

What small change can you make to this single line of code to make it produce the "real" answer to the division? (When you see the word "real" they mean the actual calculation of the division including the decimal part)

Your code here:

5. double d = 78.1;
int fg = (int)d;
System.out.println(fg);

Write the output here:

6. Write a line of code in which you divide the double precision number *d* by an integer variable called *i*. Type cast the *double* so that strictly integer division is done. Store the result in *j*, an integer.

Write your code here:

Topic 6: Methods of the Math Class

One of the most useful methods of the *Math* class is *sqrt( )* …which means square root. For example, if we want to take the square root of 17 and store the result in *p*, do the following:

double p = Math.sqrt(17);

Notice that we must store the result in a *double*…. *p* in this case. We must store in a *double* since square roots usually don't come out even.

Signature of a method:
Below we will give the description of some methods of the *Math* class… along with the signatures of the method. First, however, let's explain the meaning of **signature** (also called a **method declaration**). Consider the signature of the *sqrt( )* method:

double sqrt(double x)

double	sqrt	(double x)
type returned	method name	type of parameter we send to the method

Method	**Signature**	**Description**
abs	int **abs**(int x)	Returns the absolute value of x. (int value type)
abs	double **abs**(double x)	Returns the absolute value of x (double value type)
pow	double **pow**(double b, double e)	Returns b raised to the e power (double value type)
sqrt	double **sqrt**(double x)	Returns the square root of x (double value type)
ceil	double **ceil**(double x)	Returns next highest whole number from x (double value)
floor	double **floor**(double x)	Returns next lowest whole number from x (double value)
min	double **min**(double a, double b)	Returns the smaller of a and b (double value type)
max	double **max**(double a, double b)	Returns the larger of a and b (double value type)
min	int **min**(int a, int b)	Returns the smaller of a and b (double value type)
max	int **max**(int a, int b)	Returns the larger of a and b (double value type)

(For both *min* and *max* there are also versions that both accept and return types *float*, *short*, and *long*.)

random	double **random**( )	Returns a random double (range $0 \leq r < 1$)
round long	**round**(double x)	Returns x rounded to nearest whole number
PI	double **PI**	Returns 3.14159625…..

Now, we offer examples of each (most of these you can do on a calculator for verification): Key some of these in and get a feel for the coding. Change the numbers. Explore. ☺

1. double d = -379.22;
System.out.println(Math.abs(d)); //**379.22**

2. double b = 42.01;
double e = 3.728;
System.out.println (Math.pow(b, e)); //**1126831.027**

3. double d = 2034.56;
System.out.println(Math.sqrt(d)); //**45.10609715**

4. double d = 1.4;
System.out.println(Math.ceil(d)); //**2.0**

6-2
5. double d = -1.6;
System.out.println(Math.ceil(d)); //**-1.0**

6. double d = 1.4;
System.out.println(Math.floor(d)); //**1.0**

7. double d = -1.6;
System.out.println(Math.floor(d)); //**-2.0**

The last four examples illustrating *floor* and *ceil*ing are best understood with the following drawing:

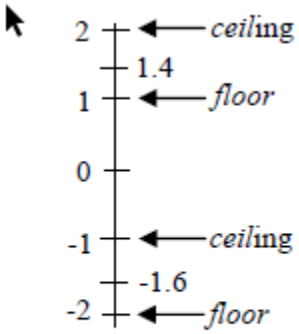


Figure 6-1 Relationship of *ceil*ing and *floor*

Just think of the *ceil*ing as it is in a house… on top. Likewise, think of the *floor* as being on the bottom.

Therefore, *Math.ceil(-1.6)* being *-1* makes perfect sense since *-1* is above. Similarly, *-2* is below *-1.6* so it makes sense to say that *-2* is *Math.floor(-1.6)*.

8. double d = 7.89;
System.out.println(Math.log(d)); //**2.065596135** …log is base e.

9. double x = 2038.5;
double y = -8999.0;
System.out.println(Math.min(x,y)); //**-8999.0**

10. double x = 2038.5;
double y = -8999.0;
System.out.println(Math.max(x,y)); //**2038.5**

11. double x = 148.2;
System.out.println(Math.round(x)); //**148**

double x = 148.7;
System.out.println(Math.round(x)); //**149**

double x = -148.2;
System.out.println(Math.round(x)); //-**148**

double x = -148.7;
System.out.println(Math.round(x)); //-**149**

12. System.out.println(Math.PI); //**3.14159265…**

Advanced *Math* methods:
Below are some additional *Math* methods that advanced math students will find useful:

Method	**Signature**	**Description**
log	double log(double x)	Returns log base e of x
sin	double sin(double a)	Returns the sine of angle a… a is in rad
cos	double cos(double a)	Returns the cosine of angle a… a is in rad
tan	double tan(double a)	Returns the tangent of angle a… a is in rad
asin	double asin(double x)	Returns arcsine of x…in range -PI/2 to PI/2
acos	double acos(double x)	Returns arccosine of x…in range 0 to PI
atan	double atan(double x)	Returns arctan of x. in range -PI/2 to PI/2
toDegrees	double toDegrees(double angRad)	Converts radians into degrees
toRadians	double toRadians(double angDeg)	Converts degrees into radians

Assignment for Topic 6: Just key in the above problems and see how they work. Please note any questions you have on this.

Bonus Topic 7: Input from the Keyboard

We will consider how to input from the keyboard the three data types…. *int*, *double*, and *String*.

Inputting an integer:
Use the *nextInt* method to input an **integer** from the keyboard:

```
import java.io.*; //This allows us to use the Scanner methods
import java.util.*;
public class Tester
{
	public static void main( String args[] )
	{
		Scanner kbReader = new Scanner(System.in);

		System.out.print("Enter your integer here. "); //enter 3001

		int i = kbReader.nextInt( );
		System.out.println(3 * i); //prints 9003
	}
}
```

Inputting a *double*:

Use the *nextDouble* method to input a ***double*** from the keyboard:

```
import java.io.*;
import java.util.*;
public class Tester
{
	public static void main( String args[] )
	{
		Scanner kbReader = new Scanner(System.in);

		System.out.print("Enter your decimal number here. "); //1000.5
		double d = kbReader.nextDouble( );
		System.out.println( 3 * d ); //prints 3001.5
	}
}
```

Inputting a *String*:

Use the *next* method to input a ***String*** from the keyboard:

```
import java.io.*;
import java.util.*;

public class Tester{
	public static void main( String args[] )
	{
		Scanner kbReader = new Scanner(System.in);

		System.out.print("Enter your String here. "); //Enter One Two

		String s = kbReader.next( ); //inputs up to first white space

		System.out.println( "This is the first part of the String,… " + s);

		s = kbReader.next( );
		System.out.println( "This is the next part of the String,… " + s);
	}
}
```

Output would be as shown below:

```
Enter your String here. One Two
This is first part of the String,... One
This is next part of the String,... Two
```

Multiple inputs:

In a similar way *nextInt( )* and *nextDouble( )* can be used multiple times to parse data input from the keyboard. For example, if **34 88 192 18** is input from the keyboard, then *nextInt( )* can be applied four times to access these four integers separated by white space.

Inputting an entire line of text:

Inputting a ***String*** (it could contain spaces) from the keyboard using *nextLine( )*:

```
import java.io.*;
import java.util.*;
public class Tester
{
	public static void main( String args[] )
	{
		Scanner kbReader = new Scanner(System.in);

		System.out.print("Enter your String here. "); //Enter One Two

		String s= kbReader.nextLine( );
		System.out.println( "This is my string,… " + s);
	}
}
```

Output would be as shown below:

```
Enter your String here. One Two
This is my string,... One Two
```

Imports necessary:

We must **import** two classes,….*java.io.* and java.util.** that provide methods for inputting integers, *double*s, and *String*s. See Appendix I for more on the meaning of "importing".

Mysterious objects:

In the above three examples we used the following code:

Scanner kbReader = new Scanner(System.in);

It simply creates the keyboard reader **object** (we arbitrarily named it *kbReader*) that provides access to the *nextInt( ), nextDouble( ), next( ),* and *nextLine( )* methods. For now just accept the necessity of all this…it will all be explained later.

The *Scanner* class used here to create our keyboard reader object only applies to1.5.0_xx or higher versions of Java. For older versions, see Appendix M for an alternate way to obtain keyboard input.

An anomaly:

Using a **single** *Scanner* object, the methods *nextInt( ), nextDouble( ), next( ),* and *nextLine( )* may be used in **any sequence** with the following exception:

It is not permissible to follow *nextInt( )* or *nextDouble( )* with *nextLine( )*. If it is necessary to do this, then a new *Scanner* object must be constructed for use with *nextLine( )* and any subsequent inputs.

Project… What's My Name? (Assignment: Try to do this)

From the keyboard enter your first and then your last name, each with its own prompt. Store each in a separate *String* and then concatenate them together to show your full name. Call both the project and the class *FullName*. When your program is finished running, the output should appear similar to that below:

```
What is your first name? Cosmo
What is your last name? Kramer
Your full name is Cosmo Kramer.
```