This summer assignment is designed to allow you to “get your feet wet” programming “methods” in the Java programming language.

**method**

---noun

1. a procedure, technique, or way of doing something, especially in accordance with a definite plan: *There are three possible methods of repairing this motor.*
2. a manner or mode of procedure, especially an orderly, logical, or systematic way of instruction, inquiry, investigation, experiment, presentation, etc.: *the empirical method of inquiry.*
3. order or system in doing anything: *to work with method.*

**Anatomy of a Method**

A method that is written in any programming language is designed to compartmentalize a defined task or procedure. Some tasks return a value and others do not. Think of the absolute value feature on your calculator, for example. If you supply a negative number (input) it returns the opposite of that number (output). If you supply a positive number (input) it returns the number unchanged (output). What if the input number is zero? Well, zero is not negative so it is returned unchanged just as a positive number is returned. This is the essence of a method...

```
public double absValue(double val)
{
    if(val > 0)
        return val;
    else
        return -1 * val;
}
```
Now, let’s break it down…

This method header (the first line of the method) defines four things (from left to right).

```
public double absValue(double val)
{
    if(val > 0)
        return val;
    else
        return -1 * val;
}
```

1. The method’s “visibility”, public, which is a story for another day. 😊

2. The return “data type” of the information that will be returned (output) by the method, double. The double data type allows for any real number including decimals (versus the int data type that restricts values to integers). When a method only performs a task that does not require a value to be passed back the return “data type” in the method header is void (again, a discussion for another day).

3. The name of the method, absValue, so that it can be invoked (called) by another programming statement.

4. A definition of the methods “argument”, stating 1) the “data type” of the information that will be passed (input) to the method, and 2) a name to refer to the information by in the body (programming statements) of the method. Again, the data type is double to allow any real number to be passed to the method, and a meaningful name for the incoming value is val. A method’s argument is always bounded by parentheses. If additional arguments are required they are defined the same way, separated from each other by a comma. Some methods do not require any arguments so the parentheses are empty.

*** We refer to val as a variable, able to hold a value of the specified type. A variable is used to reference a value during the course of the program and its value may or may not change during the course of a program. ***
The method body (bounded within the { } symbols) defines the processing to be performed against the input to return the appropriate output. This method’s body consists of an if-else structure that enables different courses of action to take place based on a well-designed test (condition). These four lines of code consist of two statements – an if statement and an else statement, each taking up two lines of code and terminated by a semi-colon, ;. One and only one of these statements will be executed in its entirety depending on the outcome of a test (condition) on the input value, val.

```java
if(val > 0)
    return val;
else
    return -1 * val;
```

1. The first line starts the if statement and defines a condition to check whether the input value, val, is positive. The if “keyword” is used to define a check that must result in either “true” or “false”. The if must be spelled properly and be typed in lower case, and the check must be bounded by parentheses. Relational operators are very commonly used to define an if condition.

2. The second line completes the if statement and is processed only when the if condition returns a “true” result. In this case it will return the input value, val, unchanged, as output to whatever called the method. Notice the semi-colon.

3. The third line starts the else statement and is processed only when the if condition returns a “false” result. It signals that something different must happen when val is not positive. This “keyword” must also be spelled properly and be typed in lower case.

4. The fourth line completes the else statement. It will return the opposite of the input value, val, unchanged, as output to whatever called the method. The asterisk, *, is the universal symbol for multiplication across programming languages, and the standard way for producing the opposite of any number is to multiply it by -1. Notice the semi-colon.
Now we are going to program a typical method using an on-line programming utility, Javabat, that you can access using www.codingbat.com. This utility now supports two programming languages, Java and Python. We will use the Java “side of the house”.

If you click on **Warmup-1** you will see a list of problems, each that can be solved by writing a method. Each problem provides a description, a method header whose details you need to conform to in your programming statements, a process that checks your statements for programming “syntax” accuracy (valid use of the programming language keywords and punctuation) when you press “Go”, and a series of “test cases” that your method is run against once your syntax is determined accurate to test the accuracy of your program’s logic. Each test case results in a green light or red light, stating the obvious – the logic of your program is not completely accurate until you receive all green lights!! 😊 These warm-up problems also contain a **Solution** button for when you are really stuck.

**Let me walk you through coding the method named in1020… click on it and do what I do…**

The first thing you might notice about this problem is that it requires two values, \( a \) and \( b \). No problem… the method header allows for each of them by separating them by commas within the parentheses. The best way to learn how to program is to jump right in!!

Notice the first thing that I did – pushing the opening \{\ symbol to the next line. I prefer to write programs so that each \{ is in line with its \} although it is not necessary.
Complete the method by typing in the following statements.

```java
public boolean in1020(int a, int b) {
    if( (a >= 10 && a <= 20) || (b >= 10 && b <= 20) )
        return true;
    else
        return false;
}
```

There are all sorts of new things to learn from this method’s statements.

1. This method’s return data type is `boolean`, which can represent values of `true` or `false`, only. This should make sense to you – each pair of input values will either satisfy the problem’s range requirement or it won’t. There is no in-between possibility, right?

2. There are two input arguments, both guaranteed to hold integer values because their data types are both `int`. They are named `a` and `b` to be able to refer to them individually within the method body programming statements.

3. The `if` condition has lots of parts – let’s break them down:
   a. `≤` and `≥` must be typed as `<=` and `>=`
   b. `&&` represents the logical operator `and` to say “this and that”
   c. `||` (shift + \ on the keyboard) represents the logical operator `or` to say “this or that or both”
   d. Parentheses are used to separate the range check on `a` from the range check on `b`. The `&&` operator combines two simple conditions together to form a compound condition on each value to make the final range determination. Each of these two compound conditions will result in their own `true` or `false` value.
   e. Once the individual compound conditions on `a` and `b` are complete (each results in the value `true` or `false`), the `||` operator combines the two compound conditions into a higher-level compound condition to determine whether one, the other, or both satisfy the range requirement. If yes, the entire if condition results in `true`, otherwise it results in `false`.

4. If the `if` condition results in a `true` value the `if` statement will complete so the method returns a `true` value before terminating (the `else` statement is ignored). On the other hand, if the `if` condition results in a `false` value, control is passed to the `else` statement the method returns a `false` value before terminating.
Now press the Go…Save, Compile, Run button

If you typed everything in correctly, you should see the following. Success!!
Now let’s see what happens when there are “errors” in your programming statements.

**Syntax errors** are caused by misspelled keywords or variable names, punctuation problems, etc…

1. Remove one of the two required semi-colons (a very common error). The “compiler” will return an error message to help you identify the problem (to the best of its ability). Replace the semi-colon to resolve the error.

2. Misspell the `else` keyword or replace one of its lower-case letters with an upper-case letter. Notice this time that the compiler is not as helpful in finding the error. This is common when a keyword is misspelled – in this case the compiler can’t recognize that `Else` is “close” to `else` since an `if` does not have to have a corresponding `else`. Fix the error.
3. “Unbalanced” ( )’s and { }’s are very, very common. Delete one of the )’s in the if condition, check out the error message, then fix the error.

Given 2 int values, return true if either of them is in the range 10..20 inclusive.

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected</th>
<th>This Run</th>
<th>Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>12, 99</td>
<td>true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>21, 12</td>
<td>true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>8, 99</td>
<td>false</td>
<td>false</td>
<td>OK</td>
</tr>
<tr>
<td>99, 10</td>
<td>true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>20, 20</td>
<td>true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>21, 21</td>
<td>false</td>
<td>false</td>
<td>OK</td>
</tr>
<tr>
<td>9, 9</td>
<td>true</td>
<td>false</td>
<td>X</td>
</tr>
</tbody>
</table>

Public boolean in1020(int a, int b) {
    if (a > 10 & a <= 20) || (b > 10 & b <= 20)
        return true;
    else
        return false;
}

Compile problems:

Logic errors are “syntactically correct” so that the program “compiles” and “runs”, but usually cause unexpected results. Under some circumstances the program will start but later fail (but more on that later). Logic errors can be very hard to find and sometimes we don’t even know they exist because the program “seems” to be running fine.

1. Change b >= 10 to b >= 8 and see what happens!!!

Given 2 int values, return true if either of them is in the range 10..20 inclusive.

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected</th>
<th>This Run</th>
<th>Checked</th>
</tr>
</thead>
<tbody>
<tr>
<td>12, 99</td>
<td>true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>21, 12</td>
<td>true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>8, 99</td>
<td>false</td>
<td>false</td>
<td>OK</td>
</tr>
<tr>
<td>99, 10</td>
<td>true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>20, 20</td>
<td>true</td>
<td>true</td>
<td>OK</td>
</tr>
<tr>
<td>21, 21</td>
<td>false</td>
<td>false</td>
<td>OK</td>
</tr>
<tr>
<td>9, 9</td>
<td>true</td>
<td>false</td>
<td>X</td>
</tr>
</tbody>
</table>

Public boolean in1020(int a, int b) {
    if (a > 10 & a <= 20) || (b > 8 & b <= 20)
        return true;
    else
        return false;
}

Doesn’t match the requirement!! The “expected” return value is false, but this method returns true. Red light!!
Now you try.

1. Use the discussion of `in1020` to try `in3050`. This method is a little more complex because there are two different range possibilities for each pair of `a` and `b`. This can be accomplished in a couple of ways.

   The Solution button provides a strategy that checks for the two ranges individually (no else statements). The final (single) return statement handles the situation where both ranges fail.

   ```java
   if (. . .) 
     return . . .;
   if (. . .) 
     return . . .;
   return . . .;
   ```

   Another strategy involves an `in-else-if` structure. Notice the difference and try it out!!

   ```java
   if (. . .) 
     return . . .;
   else if (. . .) 
     return . . .;
   else
     return . . .;
   ```

2. I now challenge you to use the Solution button for an entirely different problem to teach yourself something new that you can use to solve at least one more problem (`posNeg` and `make10` are great problems that also return a boolean value, and `sumDouble` and `intMax` return a different data type, `int`). 😊

   My only caution is that the “structure” of some of the solutions is not what I would consider to be “good style”. So, be ready for me to teach you how to write well-structured code.

Print out your work for these problems (using File/Print from your browser) and bring them to class on the first day of school.