OBJECT ORIENTED DESIGN
Projects 3

DUE DATE: Friday, Dec. 7, 2012 MIDNIGHT

REQUIREMENTS

Project #2

Using your aRandomNumberGenerator class designed in project #1, use **inheritance** to design two new classes. Those classes are to be called aDie and aCoin. Both aDie and aCoin should be derived classes from your aRandomNumberGenerator class. They are to simulate a die and a coin. The die should produce a random number between 1 and 6 when tossed. The coin should produce a random number that is 0 for heads and 1 for tails when flipped.

The aDie class will have a method called throw(). The prototype for throw() is: int aDie::throw()
The aCoin class will have a method called flip(). The prototype for flip() is: int aCoin::flip().
It is up to you to determine whether those methods should be defined as const or not.

Your project will also design a histogram class. We will call that class aHistogram.
For the histogram to know how many bins it might need you will provide it with a method that specifies the range of expected numbers. The prototype of this method will be: void setRange(int low, int high) where low represents the lowest number to be expected and high represents the highest number to be expected.
Your aRandomNumberGenerator class will now have to provide two methods that will specify the low and high numbers generated by the a class.
The prototypes for those two methods are:
int aRandomNumberGenerator::getLowNumber()
int aRandomNumberGenerator::getHighNumber()
Again, it is up to you to determine whether those methods should be defined as const or not.

All classes in the project should have proper constructors and destructors if needed.
The aRandomNumberGenerator will generate a random number through a method called generate() with prototype:

```
 int aRandomNumberGenerator::generate()
```

Determine its constness (const or not?)

The aHistogram class should see its range of potential numbers set through the setRange(int low, int high) method discussed above and should also update the appropriate bin count through a method called update() with prototype:

```
 void aHistogram::update(int number)
```

The histogram class should also have a clear() method to clear all the bins

```
 void aHistogram::clear();
```

It is up to you to determine whether the methods of the derived classes should be inherited from base class or should be defined for on their own.

In your main program, you will create one die and one coin. You will ask the user how many times s/he wishes to throw the die and how many times s/he will flip the coin.
With that information in hand you will throw the die and flip the coin the appropriate number of times and then use two separate histogram objects to display the histograms of the die tosses and coin flips.
The graphical display of the histograms will be as you did in project #2

Assuming 6000 tosses of the die, the histogram associated with the die tosses should look something like this:

```
1  ----- 990
2  ----- 995
.
.
.
6  ---- 1011
```

Meaning that the number one appeared 990 times (your mileage may vary), and so on.

The output should also have a section displaying a histogram.
The histogram should look something like this:

```
1 xxxxxxxxxxxxxx
2 xxxxxxxxxxxxxx
.
.
.
6 xxxxxxxxxxxxxx
```

Similarly, for the coin, assuming 1000 flips, the histogram associated with the coin flips should look something like this:

```
HEAD  ----- 505
TAIL   ----- 495
```

Meaning that Heads appeared 505 times and Tails appeared 495 times (your mileage may vary). Please note that this histogram does not use 1,2,…6 as it does for the die, but instead uses the full terms HEAD and TAIL

Again, in the histogram displays 'x' represents some number of occurrences of a particular face or coin side. Since you are likely to get 100's of occurrences of a face or coin side, you will not be able to use one x for one occurrence. Therefore you will have to scale your x to represent some number of occurrences for each of the histogram.
The recommendation made for project #1 was that you find out for the die and for the coin what the largest count was and then divide that count by 50 (or something like that) so that each the largest bin count would be represented by a line of 50 x's where each x would represent 1/50th of the largest bin count.
This leads to the suggestion that your aHistogram class should provide a method called count(int randomNumber) with prototype:

```
 int aHistogram::count(int randomNumber)
```

Again, you decide on the constness of this method.

This method would return the number of occurrences of a particular random number. Of course this would mean that you must make sure that given a request for the number of occurrences of a particular number, you will have to make sure that you return the count for the appropriate bin. In other words, the bin whose count you will return is not necessarily the bin whose index is the random number you passed as an argument.

Finally, since some of you seemed to have made that mistake, you need to seed the random number ONLY ONCE! This means that you should have main ask the user to provide a seed. In this way, you can always repeat a simulation run should something go wrong. This will also allow the T.A. to test your application under different conditions.

Guidelines:

You will need to make sure to use data hiding principles. Make sure you use Public and Private access rights appropriately.
Not using data hiding principles will result in a 10% penalty

Make sure each class is declared and defined in a separate header and a source files.
Not defining classes in separate files will result in a 50% penalty.

Make sure you have appropriate constructors and destructors.
Not having the appropriate constructors will result in a 10% penalty

Do not use a switch statement to update the counts in the histogram!
Using a switch statement in the updating method of your histogram will result in a 20% penalty.

You will need to make sure you have selected the appropriate constness for your methods. Not using the correct const attributes will cost you 10% in penalty.

You MUST use **inheritance** in designing the aDie and aCoin classes. If you do not use inheritance you will automatically receive a grade of 0.

Your code MUST compile! If your code doesn’t compile you will automatically receive a grade of 0.

Your executable MUST run. If not, you will automatically receive a grade of 0.

If your code doesn’t display the histogram and counts appropriately you will suffer a penalty of at least 50% depending on the cause of the error.

You will submit your project through the digital drop box feature of blackboard. You will zip your project and submit the zipped file. Please name your zip file by **lastname1_lastname2_3.zip** for each group or **lastname_3.zip** for individuals.

Each group should submit only one project file.