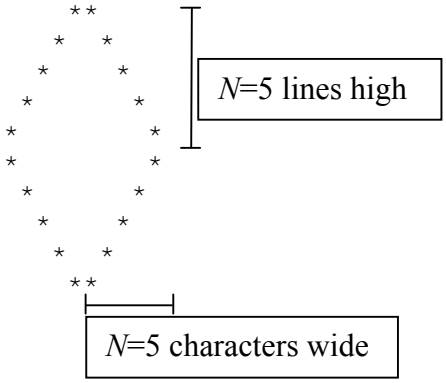
Practice Problems: Loop

1. (Source code java files are needed)

a. (Diamond.java) Write a program that reads in an integer *N* from the keyboard, and displays a diamond shape on the screen with width 2*N* and height 2*N*. For example, if *N*=5, it should display the following figure on the screen:




```
import java.util.Scanner;
public class Diamond {
  public static void main(String [] args) {
    Scanner kb = new Scanner(System.in);
    int N = kb.nextInt();

    // First part:
    // A loop that goes N times, to write the first N lines

       // Counter-controlled loop for each line?
       // Is body another loop?
              // Given the ith line, know how many spaces (' ')
              //       before *, in the middle before the 2nd *?
              //       i.e., i from 0 to n-1, we need n-1-i and
              //       2*i here, respectively!
              // Each part of space display needs a loop.

    // Second Part:
    // A loop that goes N times, to write the second N lines
       // This is basically a repeat of the loop above, except for the
       // change of counter control (values).
  }
}
```

b. (Prime.java) Write a program that reads in an integer *N* from the keyboard, and displays whether *N* is a prime number or not. A number is "prime" if its only factors are 1 and itself. A "factor" is a number that divides another number evenly.

Hint: Event control loop, what condition to terminate? … (Need to search for the next factor, until this factor reaches N! Then what is the expression in loop? How to control the event/factor change?)

Source code?

c. (Perfect.java) Write a program that reads in an integer *N* from the keyboard, and displays whether *N* is a "perfect number" or not. A number is "perfect" if it is equal to the sum of all of its factors (not including itself as a factor, but including 1 as a factor). 6 is the first perfect number, because its factors are 1, 2, and 3, and 1+2+3 = 6.

Hint: Counter control loop to add any possible factor to the sum (a check is needed to identify the required factor)!

Source code?

2. Design Strategy 1 – Read the following recipes for solving complicate loop problems. This part exercise is to verify your development and improve your programming skills for the future/next work with similar problems.

a. The ***Repeat-X Algorithm***: Repeat some set of Java commands *X* times.
Here is the recipe, written as an *algorithm*:

```
i := 1
while i <= X repeat:
    execute set of Java commands, which might depend on i and X
    i := i + 1
```

Suppose the set of Java commands that you had to repeat was just the single command:
`System.out.print("*");`
and suppose that the number of times to repeat was stored in a variable called `numStars`. Write the Java code to *implement* this algorithm.

```
int j = 1;
while(j<=numStars) {
System.out.print("*");
j += 1;
}
```

OR

```
for(int j=0; j<numStars; j++) {
System.out.print("*");
}
```

b. The ***Sum Algorithm***: Take some set of Java commands that computes a number. Repeat these commands *X* times, and compute the sum of the results from each repetition.

Here is the recipe, written as an algorithm:

```
j := 1
sum := 0
while j < X repeat:
    currentVal := execute set of Java commands, which might depend on j and X
    sum := sum + currentVal
    j := j + 1
```

Use this algorithm to compute the sum of the squares of the integers between 1 and 10.

```
int sum = 0;
for(int j=1; j<=10; j++) {
sum += j * j; }
```

c. The ***Accumulate Algorithm***: This is a slightly more general version of the Sum Algorithm. Let *f* be a function (like sum, product, min, max) that takes a set of numbers and returns a single value. We'll call *f* the accumulator function. This algorithm takes a set of Java commands that computes a number, and repeats this set of commands *X* times, and computes $f(\{result_1, \ldots, result_X\})$.

Here is the algorithm:

```
j := 1
finalResult := f({}) // the accumulator function applied to the empty set
while j < X repeat:
    currentVal := execute set of Java commands, which might depend on j and X
    finalResult := f({finalResult, currentVal})
                                                //comparison, calculation, etc.
    j := j + 1
```

Use this algorithm to read in 10 numbers from the keyboard, and find the largest one.

```
import java.util.Scanner;
public class MaxOf10 {
public static void main(String [] args) {
Scanner kb = new Scanner(System.in);
double max = kb.nextDouble();

for(int j=0; j<9; j++) {
      double val = kb.nextDouble();
      if(val>max)
         max = val;
}
}
}
```

3. Design Strategy 2 -- Breaking problems down into manageable parts. Read the following parts and think over the details in the previous part 2 of our loop development. Understand the use of design strategy (part 3) with counter/event controlled loop template.

a. Problem 1a (drawing the diamond) can be solved using only these parts:
   - System.out.println() and System.out.print() --- Body
   - the Repeat-X algorithm ---- Loop control, either counter or event
   - variables and assignment statements --- initialization

   See if you can determine how to break the problem down into the following steps. Specifically,

```
1 repeat-X loop (counter-controlled)
-  print the spaces before the first * on each line (body)
-  the number of spaces (X) depends on which line you're on (initialization
   and body)

1 repeat-X loop (counter-controlled)
-  print the spaces between the *'s on each line (body)
-  the number of spaces (X) depends on which line you're on (initialization
   and body)

1 repeat-X loop (upper part of shape, counter-controlled)
-  print the first N lines, the commands that get repeated are the two
   repeat-X loops above (also called nested loop, which is embedded in the
   loop for the entire upper part of diamond shape), plus commands to print
   the two stars (body)
-  determine the initialization by considering the need for sub-parts.

1 repeat-X loop (counter-controlled)
-  print the second N lines, the commands that get repeated are the two
   repeat-X loops above (nested loop), plus commands to print the two stars
   (body)
-  determine the initialization by considering the need for sub-parts.
```

b. The follow problem can be solved with the Repeat-X algorithm:

   Find the first prime number larger than 1000.
   - the Repeat-X algorithm for its main control (every possible number, +1)
   - the Accumulate Algorithm in Prime.java as its sub-part (i.e., testing)
   - the initial set – "1000."

c. The follow problem can be solved with the Repeat-X algorithm:

   Find the next perfect number after 6.
   - the Repeat-X algorithm for its main control
   - the Sum Algorithm in Perfect.java as its sub-part (i.e., testing)
   - the check of perfect number (sum == number)
   - the initial set – "7."

4. Practice – Write a simple program to simulate the dice game of "Craps".

(Craps.java) The program should roll two 6-sided dice and compute the sum. If the sum is 7, it should keep rolling until the sum is something different than a 7. That value is called the "point".

Once the point is established, the program should keep rolling and printing the results, until either another 7 shows or the point shows again. If a 7 shows, print "You lose!". If the point shows, print "You win!".

Source code?