PROBLEM SET 9
More Loops

1. Using a for loop, write a program that sums up all the numbers from 7 to 70,000. This should only take a few lines of code!
2. Write a program that will compute an NxM multiplication table (based on user input). For instance, if the user enters 4 and 3 you should get:

```
1   2       3       4
2   4       6       8
3   6       9       12
```

3. **Fibonacci Numbers-** Fibonacci numbers are a famous series of numbers that are used in a wide array of computer applications. For those of you who don't remember, the Fibonacci series looks like this: 0, 1, 1, 2, 3, 5, 8, 13, 21, etc. Using for loops, write a program that can find the nth Fibonacci number based on user input. Prints the entire series up to that number. Don't forget to use a for-loop!
4. **Interest Rate Calculator-** It is amazing how many apps for phones and the web just simply calculate the compound interest on some amount of money (be it savings, mortgages, credit card debt, etc.). For this problem, write a program that lets the user enter a starting amount of cash in their bank account, and interest rate (in percent) and calculates the value of their account after 30 years (assume the interest is credited to the account at the end of the year). Print out the account balance at the end of each year. Note: check the Java Math methods for methods that work with powers (exponents) to make this easier.
5. Suppose we want to make this interest calculator to be more realistic. Quite often a person will open a bank account (initial deposit) and then add a fixed amount each month from their paycheck. Also, most banks give compound interest (monthly, daily, continuous) etc. Modify the program as follows:
   a. Ask the user for the initial deposit.
   b. Ask the user to enter the amount they want to save every month.
   c. Change the interest rate to be monthly rather than annually.
   d. At the end of the list of account balances, let the user know how much they had at the end of 30 years vs. if they hadn't done their monthly deposit.
6. **One more picture pattern!** The user should enter a height and width!

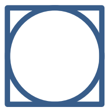
```
 ***********
 *0*0*0*0*0*
 ***********
 *0*0*0*0*0*
***********
```

7. **YET MORE PICS.** Another picture pattern. Ask the user whether they want to make a diamond or an X shape and how tall they would like it to be. Write some loops that will draw each shape, make sure to check your base cases! (Note: this only needs to work on odd numbers for the height).

```
     *          *    *
    * *          *  *
   *    *          *
    *  *         *  *
      *         *     *
```

8. **Pot Shots at Pi-**Let's calculate Pi! We're not going to do it by measuring circumferences, though; we're going to do it with Monte Carlo methods- using random numbers to run a simulation. This is a very common technique. Many probability problems are easier to calculate by letting a computer simulate the game. If the game is simulated enough times (many thousands) then a very specific mathematical rule called the "Law of Large Numbers" comes into play and makes the probabilities match the theoretical probabilities. The use of random numbers and the simulation technique is known as Monte Carlo.

   Let's take a circle of radius 1 around its origin, circumscribed by a square with side length 2, like so:



Imagine that this square is a dartboard, and that you are tossing many, many darts at it at random locations on the board. With enough darts, the ratio of darts in the circle to total darts thrown should be the ratio between the area of the circle and the area of the square. Since you know the area of the square, you can use this ratio to calculate the area of the circle, from which you can calculate pi using:

$$\pi = \frac{a}{r^2}$$

We can make the math a little easier. All we really need to calculate is the area of one quadrant and multiply by 4 to get the full area. This allows us to restrict the domain of our x and y coordinates to [0,1] (instead of having to deal with negative numbers).

In the past we have used random integers. Now, we want to use a random number that gives us back a number between 0 and 1. (consult the Math methods).

To make this work you will have to follow a few steps.

1. Write two variable declarations representing the x-coordinate and y-coordinate of a particular dart throw. Each one should be named appropriately, and should be initialized to a random double in the range [0,1].

2. Place your variable declarations within a loop that increments a variable named totalDartsinCircle if the dart’s location is within the circle’s radius. You will need to use the Euclidean distance formula

$$d^2 = x^2 + y^2$$

3. Now, use your loop to build a program that asks the user to specify the number of “dart throws” to run, and returns the decimal value of pi, using the technique outlined above. You should get pretty good results for around 5,000,000 dart throws. Experiment with this to see how accuracy changes by changing the number.