CS 100: Programming Assignment P3

Due: Friday, February 26, 5pm, Carpenter Lab (or in lecture 2/25)

You may work in pairs. Do not submit your assignment for grading unless you have read and understand the CS100 webpage on Academic Integrity. Follow the course rules for the submission of assignments or lose points.

Background

The number of possible 5-card poker hands is given by

$$\frac{52 \cdot 51 \cdot 50 \cdot 49 \cdot 48}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5} = 2598960$$

The number of different ways to form a 6-person hockey team from a pool of 550 CS 100 students is

$$\frac{550 \cdot 549 \cdot 548 \cdot 547 \cdot 546 \cdot 545}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6} = 37407576030825$$

In general, the number of possible ways to pick k objects from a set of n objects is prescribed by the *binomial coefficient*

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+2)(n-k+1)}{1 \cdot 2 \cdots (k-1)k}$$

Sometimes this quantity is referred to as "n-choose-k". Get familiar with the formula for n-choose-k by evaluating it for small examples like 10-choose-4.

Note that with the convention

$$\binom{n}{0} = 1$$

we have the "update" formula

$$\binom{n}{k} = \binom{n}{k-1}\frac{n-k+1}{k} \tag{1}$$

assuming that $k > 0$. This suggests that binomial coefficients can be computed by repeated multiplication/division:

```
b = 1;          // b has value 52-choose-0
b = b*52/1;     // b has value 52-choose-1
b = b*51/2;     // b has value 52-choose-2
b = b*50/3;     // b has value 52-choose-3
b = b*49/4;     // b has value 52-choose-4
b = b*48/5;     // b has value 52-choose-5
```

(In the world of integer arithmetic it is important to do the multiplication first. Why?)

It is not hard to show that n-choose-k can also be computed from the formula

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2}$$

Note that with the Stirling approximation

$$s_j = \sqrt{2\pi\, j}\left(\frac{j}{e}\right)^j \approx j! \tag{3}$$

we can approximate *n*-choose-*k* with *b*(*n*,*k*) where

$$b(n,k) = \frac{s_n}{s_k s_{n-k}} \ . \tag{4}$$

This is useful in situations where an approximation to a large binomial coefficient is sufficient.

In this assignment you develop a class MyMath with methods that can be used to solve a selection of problems that involve binomial coefficients. These include methods that produce *n*-choose-*k*, the Stirling approximation for factorial, and the approximation *b*(*n*,*k*). The Stirling approximation requires a square root and a power and you will write methods for these calculations in lieu of using Math.sqrt and Math.pow. The Java goal of the assignment is for you to learn about static methods.

Getting Set Up

At the end of the handout you will find listings of two files that contain the classes P3A and MyMath. They are available on the website. In the CodeWarrior environment proceed as follows:

- Create a file P3A.java and copy into it from the web the class P3A (including the preceding import statement.)
- Create a file MyMath.java and copy into it from the web the class MyMath.
- Create a project using the application stationary. Remove CUCSApplication.java from the project. Add P3A.java and MyMath.java to the project.
- Set the main class target to P3A.

Improving the sqrt Method

Notice P3A.java is practically all comments except for a few lines that involve a call to MyMath.sqrt. When the program is run you should observe that the method MyMath.sqrt isn't very good, at least when applied to the problem of computing the square root of one million.

Before we can rectify this we need to understand the underlying algorithm which is *Newton's method for square roots.* The main idea is quite intuitive. Think of y as the current "best guess" of the required square root. To visualize the quality of y, picture a rectangle with base y and height x/y. This rectangle has area x. If the sides had equal length, then y would be an exact square root. Note that the "perfect" side length we are after is in between the base and the height. So we can make the rectangle "more square" by resetting the base to be the average of the "current base" y and the current height" x/y. This averaging process is repeated twelve times in MyMath.sqrt. Apparently, twelve iterations is not enough.

One solution would be to set the iteration maximum KMAX to some huge number just to be safe. But that would be inefficient because it turns that the number of iterations required to get full precision (about 16 significant digits) depends on the input value x. To address this contingency you are to change MyMath.sqrt so that the iteration is under the control of a while-loop. The while loop should terminate if either of the following criteria is met:

1. The number of iterations exceeds KMAX = 100 where KMAX is a class constant.
2. The value of y is close to the square root in that |x-y*y|<=TOL*x where TOL is a class constant set to 10^{-16}.

Use Math.abs for the absolute value. Rerun P3A and you should observe an improvement. Notice that since the specification of MyMath.sqrt didn't change, programs that use the method (like P3A) do not have to be rewritten.

Write Your Own Absolute Value Method

Your improvement to Math.sqrt now requires access to the absolute value method in the class Math. Our plan is to remove this dependency. Proceed by implementing a method abs in the class MyMath that computes the absolute value. It should have a single type double input parameter and it should return a type double value that is its absolute value. Then "uncover" the statements in P3A.java that are placed there to test MyMath.abs. (Do this by removing the appropriate "//" symbols.) When you are confident that it is working, modify Math.sqrt so that it refers to the abs method in MyMath and not the abs method in Math.

A Method for Binomial Coefficients

Implement in MyMath a method for computing binomial coefficients with the following specification:

```
public static long binCoeff ( int n, int k )
```

It should assume 0 <= *k* <= *n* and return *n*-choose *k*. Make your implementation efficient by exploiting the fact that

$$\binom{n}{k} = \binom{n}{n-k} \quad .$$

In particular, if $k > n/2$, then your method should compute *n*-choose-*k* via the right hand side since it involves less arithmetic. For example, the identity says that

$$\binom{10}{7} = \binom{10}{3}$$

It is clear that the quotient

$$\binom{10}{7} = \frac{10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4}{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7} = 120$$

involves more arithmetic than

$$\binom{10}{10-7} = \binom{10}{3} = \frac{10 \cdot 9 \cdot 8}{1 \cdot 2 \cdot 3} = 120$$

The formula that involves the least amount of work should be used in your implementation.

To check that things are working, uncover the statements in P3A.java that compute and print the number of possible 5-card poker hands. Run the program and confirm that it computes 52-choose-5 correctly and that the same value is produced when it is applied to the 52-choose-47 problem.

A Method for Powers and the Stirling Approximation

The Stirling approximation (3) requires an exponentiation. For that purpose, implement in the class MyMath a method

```
public static double pow(double x,int n)
```

that returns the value of *x* raised to the *n*th power. The method may assume that *n* is nonnegative. Check things out by uncovering the statements in P3A.java that compute and print 2-to-the-50. Once you are confident that your power method is working correctly, implement a method

```
public static double stirling(int n)
```

in the class MyMath that returns the Stirling estimate of *n*!. For the exponentiation of (*n*/*e*), it should use the pow method in the class MyMath. For the square root, it should use the sqrt method in MyMath developed above. For e and π make E = 2.71828182845905 and PI = 3.14159265358979 type double class constants and reference them instead of Math.PI and Math.exp(1.0). Check things out by running P3A.java with the statements uncovered that compute and print the Stirling approximation to 5!.

An Approximate Method for Binomial Coefficients

Implement in the class MyMath a method

```
public static double binCoeffApprx( int n, int k )
```

that returns the approximate binomial coefficient $b(n, k)$ defined in (4). The input parameters must satisfy 0 <= k <= n. Uncover the statements in P3A.java that compute and print $b(52,5)$ and $b(550,30)$ and run the program.

The P3A Submission

Submit a listing of the class MyMath which at this point should include the constants TOL, KMAX, E, and PI, and the static methods sqrt, abs, binCoeff, pow, stirling, and binCoeffApprx. Submit a listing of the "completely uncovered" P3A.java and a copy of the output it produces which should feature

The square root of one million.
The absolute value of –10.
The absolute value of 10.
The number of possible 5-card poker hands.
The binomial coefficient 52-choose-47.
The 50th power of two.
The Stirling approximation to 5!.
The approximate number of 5-card poker hands.
The approximate number of 30-student CS100 sections.

Remember that nowhere in your implementation of MyMath should there be a reference to anything from the class Math. In addition, major style points will be deducted if the methods in MyMath are insufficiently commented. The specification for a method must be clearly written and detailed enough so that one can use the method just by reading its specification.

The P3B Submission

Suppose you are dealt n cards from a conventional deck of 52 cards. For $n = 2,\ldots,14$ the probability that you are holding exactly one pair is given by

$$p_n = \frac{13\binom{4}{2}\binom{12}{n-2}4^{n-2}}{\binom{52}{n}}.$$

Write a program P3B.java that prints a table which displays n and p_n (to four decimal places) for $\boldsymbol{n} = 2,\ldots,14$. Add P3B.java to the current project and change the main class target setting so that P3B.java is executed when the project runs. P3B.java should make effective use of the methods in the class MyMath. In particular, all powers should be computed using MyMath.pow and all binomial coefficients should be computed using MyMath.binCoeff. It should not reference any method in the Math class. Submit a listing of P3B.java and the output that it produces. Make sure youget the computation of 4-choose-2 "out of the loop."

Listings of P3A.java and MyMath.java

```
// P3A: Tests the methods in MyMath

import java.io.*;
public class P3A
{
   public static void main(String args[])
   {
   TokenReader in = new TokenReader(System.in);

   // The square root of 1,000,000.
   double y = MyMath.sqrt(1000000.0);
   Format.println(System.out,"The square root of one million = %20.15f", y);

   // The absolute value of -10.
   // double ten = MyMath.abs(-10.0);
   // System.out.println("\nThe absolute value of -10 is " + ten);

   // The absolute value of 10.
   // ten = MyMath.abs(10.0);
   // System.out.println("\nThe absolute value of 10 is " + ten);

   // Number of poker hands
   // long nPokerHands = MyMath.binCoeff(52,5);
   // System.out.println("\nThe number of possible 5-card poker hands = " + nPokerHands);
   // nPokerHands = MyMath.binCoeff(52,47);
   // System.out.println("\nThe binomial coefficient 52-choose-47     = " + nPokerHands);

   // Two-to-the-50.
   // long twoE50 = (long) MyMath.pow(2,50);
   // System.out.println("\nThe 50th power of two = " + twoE50);

   // Stirling approximation to 5!
   // double z = MyMath.stirling(5);
   // System.out.println("\nThe Stirling approximation to 5! = " + z);

   // Number of poker hands via Stirling
   // long nPokerHandsApprx = (long) MyMath.binCoeffApprx(52,5);
   // System.out.println("\nThe approximate number of 5-card poker hands = " + nPokerHandsApprx);

   // The number of possible 30-student sections in CS100
   // double nSections = MyMath.binCoeff(550,30);
   // System.out.println("\nThe approximate number of 30-student CS 100 sections = " + nSections);

    in.waitUntilEnter();
   }
}


// This class contains methods that can be used to answer various
// combinatoric questions that involve binomial coefficients.
public class MyMath
{
   public static final int KMAX = 12;  // #Newton square root iterations

   // Yields the square root of x.
   // The value of  x must be  nonnegative.
   public static double sqrt(double x)
   {
      double y = x;  // The current approximation to the square root.
      int k;
      // Perform KMAX steps of Newton’s method.
      for(k=1;k<=KMAX;k++)
      {
         y = (y+x/y)/2.0;
      }
      return y;
   }
}
```