CS100M Fall 2006  Project 4
Due: October 26, 2006 (Thursday) at 6pm

Submit your files on-line in CMS before the project deadline. Both correctness and good programming style contribute to your project score.

You must work either on your own or with one partner. You may discuss background issues and general solution strategies with others, but the project you submit must be the work of just you (and your partner). If you work with a partner, you and your partner must register as a group in CMS and submit your work as a group.

Turn off the file backup feature in DrJava—it causes problems on some system configurations! Go to menu item Edit→Preferences, choose the last category, “Miscellaneous,” then uncheck the box “Keep Emacs-style Backup Files.”

Submit your four files Calculator.java, FPIterator.java, RandGen.java, and api.txt on-line in CMS under Project 4 before the project deadline. For java code be careful to submit .java files and not the .class files. Both correctness and good programming style contribute to your grade.

Objectives

In this project, you will learn to write Java programs in the “procedural” style—the way we have written MATLAB programs up to this point. (So this is not object-oriented programming yet!) You will start with a simple program with one class and one method only to work with arithmetic operators, methods in the Math class, type conversion, and printing. The second question deals with fixed-point iteration. This part is similar to binomial root finding. Here you will deal with selection statement, loops, and writing and calling static methods. The third question deals with random number generation. You will implement a static method for random number generation and test it. Here you will also use loops, selection statement, and static methods.

Do not use arrays in this project. In each class write the methods exactly as specified--do not write any additional methods not requested in the specifications. Do not modify the provided code unless specified.

1 Calculator

Write a program Calculator.java (the class name is Calculator) to perform the following operations and print the results. The output should be “labeled.” For example, the output format for part (a) below may be  a:  324

a.  (125+77)*11
b.  55/17
c.  55.0/17
d.  The remainder of 33.234/7.12
e.  The quotient of 95/29
f.  The remainder of -89/7
g.  Find the quotient (integer) of 78.35/11.33. Hint: use casting.
h.  The result of evaluating “11 greater than 56”.
i.  The result of evaluating “11 not equal to 56”.
j. Evaluate the expression \((\sqrt{17.9999999})^2\) equal to 17.9999999. (To do this you may use Math.sqrt but not Math.pow)

k. Return the bigger of the two values, \(10^{15}\) and \(15^{10}\). (Use Math.max and Math.pow)

l. Evaluate the expression \(2^\sin^2(\pi/6)+\cos(\pi/3)\) (Use Math.sin, Math.cos and Math.PI)

m. Generate a random number on the interval \([-e^2, \ln(e^3+5)]\) (Use Math.random, Math.log and Math.exp)

n. Randomly print one letter from the set \{'a', 'b', 'c', 'd', 'e'\}. (The probability of each letter appearing should be equal.) Use Math.random.

2. Fixed-Point Iteration

Approximation is an important solution strategy for many problems. We have seen root finding using the Bisection Method already. Fixed-Point iteration is also a root finding method, but here we seek to find a point where a curve intersects the diagonal line \(y=x\).

A fixed-point of a function \(g\) is a value \(x\) such that

\[ x = g(x) \]

It’s called a fixed-point of \(g\) because the value of \(x\) is not changed when \(g\) is applied to it.

The general idea behind fixed-point iteration for finding a root of a function \(f\) is simple: (1) start with some initial guess of the root \(x\), (2) evaluate \(g(x)\), where \(g\) is some function (described below), (3) set \(x\) to be \(g(x)\), i.e., the value \(g(x)\), and go back to step (2). The hope is that \(g\) is chosen in such a way that these repeated evaluations of \(g(x)\) cause \(x\) to converge gradually to a root of the original function \(f\).

So how do we find \(g\) given \(f\)? Newton’s method gives the following simple definition:

\[ g(x) = x - f(x)/f'(x) \]

Does it actually work? Suppose we have the nonlinear function \(f(x) = x^2 - 2\). One root of \(f\) is the square root of 2. According to Newton’s method, if we guess a value \(x\) to be the root, then a good next guess will be \(x - (x^2 - 2) / 2x\). Let’s try this with an initial guess of \(x = 100\).

on iteration 1 function is 25.024996000799838
on iteration 2 function is 12.552458046745903
on iteration 3 function is 6.355894694931141
on iteration 4 function is 3.335281609280434
on iteration 5 function is 1.967465562311492
on iteration 6 function is 1.4920008896897232
on iteration 7 function is 1.416241320389438
on iteration 8 function is 1.4142150140500531
on iteration 9 function is 1.4142135623738403
on iteration 10 function is 1.4142135623730951
Function one converged in 10 steps to the value 1.4142135623738403

Math.sqrt(2) returns 1.41421356237310.

Write a program FPIterator.java to calculate the cube root of an input number \(n\) via Newton’s method and fixed-point iteration. Finding a cube root of a number \(n\) is to find \(x\) such that \(x^3 = n\). Re-
arranging the equality gives \( x^3 - n = 0 \). In other words, we are looking for the root of non-linear function \( f(x) = x^3 - n \).

Some notes on your implementation:

- For each iteration, output the current "guess."
- You must determine when your answer is "good enough" so that the program stops and doesn't loop forever. Keep track of how many iteration it takes to reach this approximate value. You cannot calculate the root directly and compare it with the approximated values. As a hint, notice that as we approach the "real" answer, the change in the guess becomes very small. For example, in finding the square root of 2 with an initial guess of 100, the changes in the subsequent guesses became smaller and smaller, as shown above. (You might also want to have a limit on the maximum number of iterations allowed, but be sure that this limit is big enough.)
- You must write (and use) a static method that will take two parameters, \( x \) and \( n \), and return the value of \( g(x) \).

A run with an example implementation of FPIterator.java follows here. Treat this sample output as a loose guideline for how your output should look like. Depending on your stopping criteria, your program may stop with a different number of iterations given the same \( n \) and initial guess.

```
Enter the number which cube root to find: 9
Enter the first guess for the root: 3
1: 2.3333333333333335
2: 2.1065759637188206
3: 2.080415589606098
4: 2.0800838759563316
5: 2.0800838230519054
Function converged in 5 steps.
Cubic root of 9.0 is 2.0800838230519054.
```

3 Random Number Generator

At this point we have seen and used random number generators. Now let’s think about how random number generators work. What we call a “random number generator” is actually a pseudorandom number generator. Pseudorandom numbers are not exactly random; the sequence simply appears random, i.e., indistinguishable from a truly random sequence. In many real world applications this is good enough.

Linear congruential generators (LCGs) represent one of the oldest and best-known pseudorandom number generator algorithms. The theory behind this algorithm is easy to understand and the algorithm is easily implemented and fast. We will implement an LCG here due to its simplicity. However, this class of generators has some problematic properties, so one may choose another algorithm when high quality random numbers are needed.

LCGs are defined by the recurrence relation

\[
V_{j+1} = (A \cdot V_j + B) \mod M
\]

where \( V_j \) is the \( j \)th value in the sequence of random values, and \( A, B \) and \( M \) are generator-specific integer constants. \( \mod \) is the modulo (remainder) operation. This recurrence relationship yields a sequence of pseudorandom numbers. To scale the \( j \)th random values to the range \([0, 1)\), divide \( V_j \) with \( M \). But what should be the initial value, \( V_0 \), to start the sequence?
We need a seed value to initialize the pseudorandom sequence. A reasonable seed is the system time, or how many seconds have elapsed since a particular start time, since this value is “never” the same. The statements

```
Date now= new Date();
long time= now.getTime(); //number of milliseconds since Jan 1, 1970
```

stores the number of milliseconds since January 1st, 1970, in variable time. Note that we use the integer type long for time, a rather large value. With this seed (variable time), the initial value \(V_0\) could be

time modulo \(M\)

**Again, scale \(V_0\) by dividing by \(M\) to get a number in \([0,1)\).** We recommend some values for the constants, \(A, B,\) and \(M\), below, but for completeness we mention the conditions necessary for generating good pseudorandom numbers:

1. \(B\) and \(M\) are relatively prime (two numbers are relatively prime if their greatest common divisor is 1).
2. \(A-1\) is divisible by all prime factors of \(M\).
3. \(A-1\) is a multiple of 4 if \(M\) is a multiple of 4.
4. \(M > \max(A, B, V_0)\)
5. \(A > 0, B > 0\)

While LCGs are (theoretically) capable of producing decent pseudorandom numbers, they are extremely sensitive to the choice of \(B, M\) and \(A\). The recommended values (use type long) are

\[
\begin{align*}
A &= 1664525 \\
B &= 1013904223 \\
M &= 2^{32}
\end{align*}
\]

Complete method `my_rand()` in class `RandGen` (file `RandGen.java`) to generate and return one random number at a time. You will also write a `main` method to demonstrate your `my_rand` method. The first call to `my_rand` gives \(V_0\) scaled to \([0,1)\), the next call gives \(V_1\) scaled to \([0,1)\), the following call gives \(V_2\) scaled to \([0,1)\), and so forth, as determined by the recurrence formula given above. To achieve this, we need to make use of two class variables, also called static variables, in the `my_rand()` method.

Class variable `nextNum` is already declared for you and can be used to store the last random number generated. (Recall that the scope of a class variable is the entire class, so a class variable’s value does not change or disappear in between function calls. In other words, we use class variable `nextNum` as what we called in MATLAB a global, persistent variable.) A second class variable that we have provided for you is `isInitialized`. Class variable `isInitialized` is set to `false` initially to indicate that the random number generator has not been initialized (has not generated a first value, \(V_0\) yet. Recall from the description of the LCG above that the calculation for \(V_0\) is different from the calculation of the subsequent values \(V_1, V_2, \ldots\) Think about how your `my_rand` method should make use of class variable `isInitialized`.

To test the quality of this generator, use `my_rand` to generate at least 1 million random numbers from the set \{1, 2, 3, 4, 5\}. Do this test in the `main` method. Each of these numbers should be generated with equal probability. Report the actual probability with which each number appears and the differences between the actual and expected probabilities. Really good generators will have differences that are close to 0. We’ll see how our LCG does!

Below is the output of an example run of the program:
4 Java API

Read about the Java API in Section 2.2 (page 36) of the Gaddis text. Next, go to the documentation of the Java API at http://java.sun.com/j2se/1.5.0/docs/api/ to find the answers to the questions below. You do not need to read the API documentation in detail to answer these questions. The objective of this exercise is for you to learn about the API and to learn how to find detailed specifications should you have the need in the future. To answer most of these questions, you only need to skim through the first pages of the description or tables for each class. Save your answers in a plain text file api.txt and submit it to CMS.

1. What does API stand for? Briefly explain what the Java API is.
2. Take a look at the Math class. How many abs methods (absolute value) are there? What are the two fields in the Math class?
3. Take a look at the System class. Which method do you use to get the current system time?
4. Take a look at the Double class. What does the field MIN_VALUE store?
5. Take a look at the String class. What does method trim do?
6. Take a look at the DecimalFormat class. How do the pattern characters 0 and # differ?