**Programming Exercises for Chapter 8**

THIS PAGE CONTAINS several exercises for Chapter 8 in [Introduction to Programming Using Java](http://math.hws.edu/eck/cs124/javanotes7/index.html). For each exercise, a link to a possible solution is provided. Each solution includes a discussion of how a programmer might approach the problem and interesting points raised by the problem or its solution, as well as complete source code of the solution.

**Exercise 8.1:**

Write a program that uses the following subroutine, from [Subsection 8.3.3](http://math.hws.edu/eck/cs124/javanotes7/c8/s3.html#robustness.3.3), to solve equations specified by the user.

/\*\*

\* Returns the larger of the two roots of the quadratic equation

\* A\*x\*x + B\*x + C = 0, provided it has any roots. If A == 0 or

\* if the discriminant, B\*B - 4\*A\*C, is negative, then an exception

\* of type IllegalArgumentException is thrown.

\*/

static public double root( double A, double B, double C )

throws IllegalArgumentException {

if (A == 0) {

throw new IllegalArgumentException("A can't be zero.");

}

else {

double disc = B\*B - 4\*A\*C;

if (disc < 0)

throw new IllegalArgumentException("Discriminant < zero.");

return (-B + Math.sqrt(disc)) / (2\*A);

}

}

Your program should allow the user to specify values for A, B, and C. It should call the subroutine to compute a solution of the equation. If no error occurs, it should print the root. However, if an error occurs, your program should catch that error and print an error message. After processing one equation, the program should ask whether the user wants to enter another equation. The program should continue until the user answers no.

[See the Solution](http://math.hws.edu/eck/cs124/javanotes7/c8/ex1-ans.html)

**Exercise 8.2:**

As discussed in [Section 8.1](http://math.hws.edu/eck/cs124/javanotes7/c8/s1.html), values of type int are limited to 32 bits. Integers that are too large to be represented in 32 bits cannot be stored in an int variable. Java has a standard class,java.math.BigInteger, that addresses this problem. An object of type *BigInteger* is an integer that can be arbitrarily large. (The maximum size is limited only by the amount of memory available to the Java Virtual Machine.) Since *BigIntegers* are objects, they must be manipulated using instance methods from the *BigInteger* class. For example, you can't add two *BigIntegers* with the +operator. Instead, if N and M are variables that refer to *BigIntegers*, you can compute the sum of N and M with the function call N.add(M). The value returned by this function is a new *BigInteger* object that is equal to the sum of N and M.

The *BigInteger* class has a constructor new BigInteger(str), where str is a string. The string must represent an integer, such as "3" or "39849823783783283733". If the string does not represent a legal integer, then the constructor throws a *NumberFormatException*.

There are many instance methods in the *BigInteger* class. Here are a few that you will find useful for this exercise. Assume that N and M are variables of type BigInteger.

* N.add(M) -- a function that returns a *BigInteger* representing the sum of N and M.
* N.multiply(M) -- a function that returns a *BigInteger* representing the result of multiplying N times M.
* N.divide(M) -- a function that returns a *BigInteger* representing the result of dividing N by M, discarding the remainder.
* N.signum() -- a function that returns an ordinary int. The returned value represents the sign of the integer N. The returned value is 1 if N is greater than zero. It is -1 if N is less than zero. And it is 0 if N is zero.
* N.equals(M) -- a function that returns a boolean value that is true if N and M have the same integer value.
* N.toString() -- a function that returns a *String* representing the value of N.
* N.testBit(k) -- a function that returns a boolean value. The parameter k is an integer. The return value is true if the k-th bit in N is 1, and it is false if the k-th bit is 0. Bits are numbered from right to left, starting with 0. Testing "if (N.testBit(0))" is an easy way to check whether N is even or odd. N.testBit(0) is true if and only if N is an odd number.

For this exercise, you should write a program that prints 3N+1 sequences with starting values specified by the user. In this version of the program, you should use *BigIntegers* to represent the terms in the sequence. You can read the user's input into a *String* with the TextIO.getln() function. Use the input value to create the *BigInteger* object that represents the starting point of the 3N+1 sequence. Don't forget to catch and handle the *NumberFormatException* that will occur if the user's input is not a legal integer! You should also check that the input number is greater than zero.

If the user's input is legal, print out the 3N+1 sequence. Count the number of terms in the sequence, and print the count at the end of the sequence. Exit the program when the user inputs an empty line.

[See the Solution](http://math.hws.edu/eck/cs124/javanotes7/c8/ex2-ans.html)

**Exercise 8.3:**

A Roman numeral represents an integer using letters. Examples are XVII to represent 17, MCMLIII for 1953, and MMMCCCIII for 3303. By contrast, ordinary numbers such as 17 or 1953 are called Arabic numerals. The following table shows the Arabic equivalent of all the single-letter Roman numerals:

M 1000 X 10

D 500 V 5

C 100 I 1

L 50

When letters are strung together, the values of the letters are just added up, with the following exception. When a letter of smaller value is followed by a letter of larger value, the smaller value is subtracted from the larger value. For example, IV represents 5 - 1, or 4. And MCMXCV is interpreted as M + CM + XC + V, or 1000 + (1000 - 100) + (100 - 10) + 5, which is 1995. In standard Roman numerals, no more than three consecutive copies of the same letter are used. Following these rules, every number between 1 and 3999 can be represented as a Roman numeral made up of the following one- and two-letter combinations:

M 1000 X 10

CM 900 IX 9

D 500 V 5

CD 400 IV 4

C 100 I 1

XC 90

L 50

XL 40

Write a class to represent Roman numerals. The class should have two constructors. One constructs a Roman numeral from a string such as "XVII" or "MCMXCV". It should throw a *NumberFormatException* if the string is not a legal Roman numeral. The other constructor constructs a Roman numeral from an int. It should throw a *NumberFormatException* if the int is outside the range 1 to 3999.

In addition, the class should have two instance methods. The method toString() returns the string that represents the Roman numeral. The method toInt() returns the value of the Roman numeral as an int.

At some point in your class, you will have to convert an int into the string that represents the corresponding Roman numeral. One way to approach this is to gradually "move" value from the Arabic numeral to the Roman numeral. Here is the beginning of a routine that will do this, where number is the int that is to be converted:

String roman = "";

int N = number;

while (N >= 1000) {

// Move 1000 from N to roman.

roman += "M";

N -= 1000;

}

while (N >= 900) {

// Move 900 from N to roman.

roman += "CM";

N -= 900;

}

.

. // Continue with other values from the above table.

.

(You can save yourself a lot of typing in this routine if you use arrays in a clever way to represent the data in the above table.)

Once you've written your class, use it in a main program that will read both Arabic numerals and Roman numerals entered by the user. If the user enters an Arabic numeral, print the corresponding Roman numeral. If the user enters a Roman numeral, print the corresponding Arabic numeral. (You can tell the difference by using TextIO.peek() to peek at the first character in the user's input (see [Subsection 8.2.2](http://math.hws.edu/eck/cs124/javanotes7/c8/s2.html#robustness.2.2)). If the first character is a digit, then the user's input is an Arabic numeral. Otherwise, it's a Roman numeral.) The program should end when the user inputs an empty line.

[See the Solution](http://math.hws.edu/eck/cs124/javanotes7/c8/ex3-ans.html)

**Exercise 8.4:**

The source code file [*Expr.java*](http://math.hws.edu/eck/cs124/javanotes7/source/chapter8/Expr.java) defines a class, *Expr*, that can be used to represent mathematical expressions involving the variable x. The expression can use the operators +, -, \*, /, and ^ (where ^represents the operation of raising a number to a power). It can use mathematical functions such as sin, cos, abs, and ln. See the source code file for full details. The *Expr* class uses some advanced techniques which have not yet been covered in this textbook. However, the interface is easy to understand. It contains only a constructor and two public methods.

The constructor new Expr(def) creates an *Expr* object defined by a given expression. The parameter, def, is a string that contains the definition. For example, new Expr("x^2") or new Expr("sin(x)+3\*x"). If the parameter in the constructor call does not represent a legal expression, then the constructor throws an *IllegalArgumentException*. The message in the exception describes the error.

If func is a variable of type Expr and num is of type double, then func.value(num) is a function that returns the value of the expression when the number num is substituted for the variable x in the expression. For example, if Expr represents the expression 3\*x+1, then func.value(5) is 3\*5+1, or 16. If the expression is undefined for the specified value of x, then the special value Double.NaN is returned; no exception is thrown.

Finally, func.toString() returns the definition of the expression. This is just the string that was used in the constructor that created the expression object.

For this exercise, you should write a program that lets the user enter an expression. If the expression contains an error, print an error message. Otherwise, let the user enter some numerical values for the variable x. Print the value of the expression for each number that the user enters. However, if the expression is undefined for the specified value of x, print a message to that effect. You can use the boolean-valued function Double.isNaN(val) to check whether a number, val, is Double.NaN.

The user should be able to enter as many values of x as desired. After that, the user should be able to enter a new expression.

[See the Solution](http://math.hws.edu/eck/cs124/javanotes7/c8/ex4-ans.html)

**Exercise 8.5:**

This exercise uses the class *Expr*, which was described in [Exercise 8.4](http://math.hws.edu/eck/cs124/javanotes7/c8/ex4-ans.html) and which is defined in the source code file [*Expr.java*](http://math.hws.edu/eck/cs124/javanotes7/source/chapter8/Expr.java). For this exercise, you should write a GUI program that can graph a function, f(x), whose definition is entered by the user. The program should have a text-input box where the user can enter an expression involving the variable x, such as x^2 or sin(x-3)/x. This expression is the definition of the function. When the user presses return in the text input box, the program should use the contents of the text input box to construct an object of type *Expr*. If an error is found in the definition, then the program should display an error message. Otherwise, it should display a graph of the function. (Recall: A JTextField generates an ActionEvent when the user presses return.)

The program will need a *JPanel* for displaying the graph. To keep things simple, this panel should represent a fixed region in the xy-plane, defined by -5 <= x <= 5 and -5 <= y <= 5. To draw the graph, compute a large number of points and connect them with line segments. (This method does not handle discontinuous functions properly; doing so is very hard, so you shouldn't try to do it for this exercise.) My program divides the interval -5 <= x <= 5 into 300 subintervals and uses the 301 endpoints of these subintervals for drawing the graph. Note that the function might be undefined at one of these x-values. In that case, you have to skip that point.

A point on the graph has the form (x,y) where y is obtained by evaluating the user's expression at the given value of x. You will have to convert these real numbers to the integer coordinates of the corresponding pixel on the canvas. The formulas for the conversion are:

a = (int)( (x + 5)/10 \* width );

b = (int)( (5 - y)/10 \* height );

where a and b are the horizontal and vertical coordinates of the pixel, and width and height are the width and height of the panel.

[See the Solution](http://math.hws.edu/eck/cs124/javanotes7/c8/ex5-ans.html)