Lab 10 Exceptions

Exceptions Aren't Always Errors

File *CountLetters.java* contains a program that reads a word from the user and prints the number of occurrences of each letter in the word. Compile and run it to see how it works. In reading the code, note that the word is converted to all upper case first, then each letter is translated to a number in the range 0..25 (by subtracting 'A') for use as an index. No test is done to ensure that the characters are in fact letters.

1. Run CountLetters and enter a phrase, that is, more than one word with spaces or other punctuation in between. It should throw an ArrayIndexOutOfBoundsException, because a non-letter will generate an index that is not between 0 and 25. It might be desirable to allow non-letter characters, but not count them. Of course, you could explicitly test the value of the character to see if it is between 'A' and 'Z'. However, an alternative is to go ahead and use the translated character as an index, and catch an ArrayIndexOutOfBoundsException if it occurs. Since you want don't want to do anything when a non-letter occurs, the handler will be empty. Modify this method to do this as follows:
   - Put the body of the first for loop in a try.
   - Add a catch that catches the exception, but don't do anything with it.

   Compile and run your program.
2. Now modify the body of the catch so that it prints a useful message (e.g., "Not a letter") followed by the exception. Compile and run the program. Although it's useful to print the exception for debugging, when you're trying to smoothly handle a condition that you don't consider erroneous you often don't want to. In your print statement, replace the exception with the character that created the out of bounds index. Run the program again; much nicer!

Placing Exception Handlers

File *ParseInts.java* contains a program that does the following:

- Prompts for and reads in a line of input
- Uses a second Scanner to take the input line one token at a time and parses an integer from each token as it is extracted.
- Sums the integers.
- Prints the sum.

Compile and run ParseInts. If you give it the input

```
10 20 30 40
```

it should print

```
The sum of the integers on the line is 100.
```

Try some other inputs as well. Now try a line that contains both integers and other values, e.g.,

```
We have 2 dogs and 1 cat.
```

You should get a NumberFormatException when it tries to call Integer.parseInt on "We", which is not an integer. One way around this is to put the loop that reads inside a try and catch the NumberFormatException but not do anything with it. This way if it's not an integer it doesn't cause an error; it goes to the exception handler, which does nothing. Do this as follows:

- Modify the program to add a try statement that encompasses the entire while loop. The try and opening { should go before the while, and the catch after the loop body. Catch a NumberFormatException and have an empty body for the catch.
- Compile and run the program and enter a line with mixed integers and other values. You should find that it stops summing at the first non-integer, so the line above will produce a sum of 0, and the line "1 fish 2 fish" will produce a sum of 1. This is because the entire loop is inside the try, so when an exception is thrown the loop is terminated. To make it continue, move the try and catch inside the loop. Now when an exception is thrown, the next statement is the next iteration of the loop, so the entire line is processed. The dogs-and-cats input should now give a sum of 3, as should the fish input.

Throwing Exceptions

File *Factorials.java* contains a program that calls the factorial method of the MathUtils class to compute the factorials of integers entered by the user. Compile and run Factorials to see how it works. Try several positive integers, then try a negative number. You should find that it works for small positive

integers (values < 17), but that it returns a large negative value for larger integers and that it always returns 1 for negative integers.

1. Returning 1 as the factorial of any negative integer is not correct—mathematically, the factorial function is not defined for negative integers. To correct this, you could modify your factorial method to check if the argument is negative, but then what? The method must return a value, and even if it prints an error message, whatever value is returned could be misconstrued. Instead it should throw an exception indicating that something went wrong so it could not complete its calculation. You could define your own exception class, but there is already an exception appropriate for this situation – IllegalArgumentException, which extends RuntimeException. Modify your program as follows:
   - Modify the header of the factorial method to indicate that factorial can throw an IllegalArgumentException.
   - Modify the body of factorial to check the value of the argument and, if it is negative, throw an IllegalArgumentException. Note that what you pass to throw is actually an instance of the IllegalArgumentException class, and that the constructor takes a String parameter. Use this parameter to be specific about what the problem is.

- Compile and run your Factorials program after making these changes. Now when you enter a negative number an exception will be thrown, terminating the program. The program ends because the exception is not caught, so it is thrown by the main method, causing a runtime error.
- Modify the main method in your Factorials class to catch the exception thrown by factorial and print an appropriate message, but then continue with the loop. Think carefully about where you will need to put the try and catch.

2. Returning a negative number for values over 16 also is not correct. The problem is arithmetic overflow—the factorial is bigger than can be represented by an int. This can also be thought of as an IllegalArgumentException—this factorial method is only defined for arguments up to 16. Modify your code in factorial to check for an argument over 16 as well as for a negative argument. You should throw an IllegalArgumentException in either case, but pass different messages to the constructor so that the problem is clear.

Copying a File

Write a program that prompts the user for a filename, then opens a Scanner to the file and copies it, a line at a time, to the standard output. If the user enters the name of a file that does not exist, ask for another name until you get one that refers to a valid file. Some things to consider:

- Remember that you can create a Scanner from a File object, which you can create from the String representing the filename.
- The Scanner constructor that takes a File may throw a FileNotFoundException -- this is how you will know if the file does not exist. Think carefully about how to use the try/catch structure in combination with a loop that asks for a new filename if the current file does not exist.
- Remember that the scope of a variable declared inside a try is the try itself -- it does not extend to the following code. Furthermore, the compiler knows that an initialization that occurs inside a try may or may not get executed, as the try may be thrown out of first. So any variable that you will want to use both in and after the try must be declared and initialized before the try.

Reading from and Writing to Text Files

Write a program that will read in a file of student academic credit data and create a list of students on academic warning. The list of students on warning will be written to a file. Each line of the input file will contain the student name (a single String with no spaces), the number of semester hours earned (an integer), the total quality points earned (a double). The following shows part of a typical data file:

```
Smith 27 83.7
Jones 21 28.35
Walker 96 182.4
```

```
Doe 60 150
```

The program should compute the GPA (grade point or quality point average) for each student (the total quality points divided by the number of semester hours) then write the student information to the output file if that student should be put on academic warning. A student will be on warning if he/she has a GPA less than 1.5 for students with fewer than 30 semester hours credit, 1.75 for students with fewer than 60 semester hours credit, and 2.0 for all other students. The file *Warning.java* contains a skeleton of the program. Do the following:

1. Set up a Scanner object scan from the input file and a PrintWriter outFile to the output file inside the try clause (see the comments in the program). Note that you'll have to create the PrintWriter from a FileWriter, but you can still do it in a single statement.
2. Inside the while loop add code to read and parse the input—get the name, the number of credit hours, and the number of quality points. Compute the GPA, determine if the student is on academic warning, and if so write the name, credit hours, and GPA (separated by spaces) to the output file.
3. After the loop close the PrintWriter.
4. Think about the exceptions that could be thrown by this program:
   - A FileNotFoundException if the input file does not exist
   - A NumberFormatException if it can't parse an int or double when it tries to – this indicates an error in the input file format
   - An IOException if something else goes wrong with the input or output stream

   Add a catch for each of these situations, and in each case give as specific a message as you can. The program will terminate if any of these exceptions is thrown, but at least you can supply the user with useful information.
5. Test the program. Test data is in the file *students.dat*. Be sure to test each of the exceptions as well.

A Movable Circle

File *MoveCircle.java* contains a program that uses CirclePanel.java to draw a circle and let the user move it by pressing buttons. Save these files to your directory and compile and run MoveCircle to see how it works. Then modify the code in CirclePanel as follows:

1. Add mnemonics to the buttons so that the user can move the circle by pressing the ALT-l, ALT-r, ALT-u, or ALT-d keys.
2. Add tooltips to the buttons that indicate what happens when the button is pressed, including how far it is moved.
3. When the circle gets all the way to an edge, disable the corresponding button. When it moves back in, enable the button again. Note that you will need instance variables (instead of local variables in the constructor) to hold the buttons and the panel size to make them visible to the listener. Bonus: In most cases the circle won't hit the edge exactly; check for this (e.g., $x<0$) and adjust the coordinates so it does.

A Currency Converter

Your are headed off on a world-wide trip and need a program to help figure out how much things in other countries cost in dollars. You plan to visit Canada, several countries in Europe, Japan, Australia, India, and Mexico so your program must work for the currencies in those countries. The files *CurrencyConverter.java* and *RatePanel.java* contain a skeleton of a program to do this conversion. Complete it as follows:

1. CurrencyPanel currently contains only two components—a JLabel for the title and a JLabel to display the result of the calculations. It also contains two parallel arrays—one is an array of currency names (an array of strings) and the other an array of corresponding exchange rates (the value of one unit of the currency in U.S. Dollars). Compile and run the program to see what it looks like (not much!!).
2. Add a combo box to let the user select the currency. The argument to the constructor should be the array of names of the currencies. Note that the first currency name is actually an instruction to the user so there is no need to have an additional label.
3. Modify actionPerformed in ComboListener so that index is set to be the index of the selected item.
4. Test what you have so far. It should show what one unit of the given currency is in U.S. dollars.
5. Now add a text field (and label) so the user can enter the cost of an item in the selected currency. You need to update the ComboListener so that it gets this value from the textfield and computes and displays the equivalent amount in dollars.
6. Test your program.
7. Modify the layout to create a more attractive GUI.

Listing Prime Numbers

The file *Primes.java* contains a program to compute and list all prime numbers up to and including a number input by the user. Most of the work is done in the file *PrimePanel.java* that defines the panel. The GUI contains a text field for the user to enter the integer, a button for the user to click to get a list of primes, and a text area to display the primes. However, if the user puts in a large integer the primes do not all fit in the text area. The main goal of this exercise is to add scrolling capabilities to the text area.

Proceed as follows:

1. Compile and run the program as it is. You should see a GUI that contains the components listed above but nothing happens when you click on the button. Fix this.
2. Modify PrimePanel.java so that the text area for displaying the primes is in a scroll pane. To do this, keep your JTextArea but create a JScrollPane from it, and add the scroll pane to the panel. (If you look at the JScrollPane documentation on p. 834-835 you'll see that one of the constructors takes a Component, in this case your JTextArea.) Test your modification.
3. You should see that the scrollbars don't appear unless the output is longer than the text area. The default is for the scrollbars to appear only as needed. This can be changed by setting the

scroll bar policy of the JScrollPane object. Use the setVerticalScrollBarPolicy method of the JScrollPane class to specify that the vertical scroll bar should always be displayed. The method takes an integer argument that represents the policy. The ScrollPaneConstants class has several static integer constants for the policies. These include VERTICAL_SCROLLBAR_ALWAYS, VERTICAL_SCROLLBAR_AS_NEEDED, VERTICAL_SCROLLBAR_NEVER. It should be clear which of these to use as the parameter. Remember how to access static members of a class!

4. The code to generate the list of primes could be improved some. Two things that should be done are:
   - A exception should be caught if the user enters non-integer data. An appropriate message should be displayed in the text area.
   - The loop that looks for divisors of the integer i should not go all the way up to i. Instead it should stop at the square root of i (if a divisor hasn't been found by then there isn't one).

   Make these modifications.