Case 1 : 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. Save it to your directory and study it, then 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!

```
// ****************************************************************
// CountLetters.java
//
// Reads a words from the standard input and prints the number of
// occurrences of each letter in that word.
//
// ****************************************************************
import java.util.Scanner;
public class CountLetters{
    public static void main(String[] args){
       int[] counts = new int[26];
       Scanner scan = new Scanner(System.in);

       //get word from user
       System.out.print("Enter a single word (letters only, please): ");
       String word = scan.nextLine();

       //convert to all upper case
       word = word.toUpperCase();

       //count frequency of each letter in string
       for (int i=0; i < word.length(); i++)
          counts[word.charAt(i)-'A']++;

       //print frequencies
       System.out.println();
       for (int i=0; i < counts.length; i++)
          if (counts [i] != 0)
             System.out.println((char)(i +'A') + ": " + counts[i]);
    }
}
```

Case 2 : 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.

Save ParseInts to your directory and compile and run it. 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.

```
// ************************************************************
// ParseInts.java
//
// Reads a line of text and prints the integers in the line.
//
// ************************************************************
import java.util.Scanner;
public class ParseInts{
    public static void main(String[] args){
     int val, sum=0;
     Scanner scan = new Scanner(System.in);
     String line;

     System.out.println("Enter a line of text");
     Scanner scanLine = new Scanner(scan.nextLine());

     while (scanLine.hasNext()){
        val = Integer.parseInt(scanLine.next());
        sum += val;
     }
     System.out.println("The sum of the integers on this line is " + sum);
    }
}
```

Case 3 : 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. Save these files to your directory and study the code in both, then 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.

```
// ************************************************************
// Factorials.java
// Reads integers from the user and prints the factorial of each.
//
// ************************************************************
import java.util.Scanner;
public class Factorials{
   public static void main(String[] args){
      String keepGoing = "y";
      Scanner scan = new Scanner(System.in);
      while (keepGoing.equals("y") || keepGoing.equals("Y")){
         System.out.print("Enter an integer: ");
         int val = scan.nextInt();
         System.out.println("Factorial(" + val + ") = "+ MathUtils.factorial(val));
         System.out.print("Another factorial? (y/n) ");
         keepGoing = scan.next();
     }
   }
}
```

```
// ********************************************************************
// MathUtils.java
//
// Provides static mathematical utility functions.
//
// ********************************************************************
public class MathUtils{
    //-----------------------------------------------------------------
    // Returns the factorial of the argument given
    //-----------------------------------------------------------------
    public static int factorial(int n){
        int fac = 1;
        for (int i=n; i>0; i--)
        fac *= i;
        return fac;
    }
}
```