Exception Handling in Java
IFT 194: Lab 6

Brandon Doyle
bdoyle5@asu.edu
1215232174

Dr. Usha Jagannathan
Usha.Jagannathan@asu.edu

August 9, 2018

Summary

View the source of this document on GitHub.

Exceptions aren't always errors

For this section we're given a class `CountLetters` (cf. Figure 1) that reads a word from the user and prints the number of occurrences of each letter in the word. However, we're of course not guaranteed to *only* get letters, so it will throw an error if we provide any other input.

```java
package lab_6;

import java.util.Scanner;

public class CountLetters
{
    public static final char[] ALPHABET = "abcdefghijklmnopqrstuvwxyz".toCharArray();
    public static final int ALPHABET_LENGTH = ALPHABET.length;

    public static void main(String[] args)
    {
        try (var scnr = new Scanner(System.in)) {
            int counts[] = new int[ALPHABET_LENGTH];
            String input;
            final int bias = 'a';

            //do {
                System.out.print("Enter a single word (letters only): ");
                input = scnr.nextLine();

                // Ensure the input is only letters with regex
                //if (!input.matches("[a-zA-Z]*")) {
                //    System.out.println("*** Error: Please enter only letters");
                //    continue;
                //}

            //    break;
            //} while (true);

            {
                int i = 0;

                try {
                    // Subtract
                    for (char c : input.toLowerCase().toCharArray())
                        counts[c - bias]++;

                    // Print spectrum
                    for (; i < ALPHABET_LENGTH; ++i)
                        if (counts[i] != 0)
                            System.out.printf("%c: %d\n", ALPHABET[i], counts[i]);
                } catch (ArrayIndexOutOfBoundsException ex) {
                    System.out.printf(
                        "*** Error: Please enter only letters, received \"%c\"\n", counts[i]);
                }
            }
        }
    }
}
```

Figure 1: CountLetters.java

You may also uncomment the loop containing the call to `scnr.nextLine`, which uses a regular expression to ensure the input contains only a certain set of characters.

Placing Exception Handlers

In this section we're given a class `ParseInts` in Figure 2 for parsing a line of text containing integers (hopefully).

```java
package lab_6;

import java.util.Scanner;

public class ParseInts
{
    public static void main(String[] args)
    {
        try (var scnr = new Scanner(System.in)) {
            int sum = 0;
            String line;

            System.out.print("Enter a line of text: ");
            line = scnr.nextLine();

            String[] values = line.split(" ");
            for (String number : values)
                sum += Integer.parseInt(number);

            System.out.printf("The sum of the integers on this line is: %d", sum);
        }
    }
}
```

Figure 2: ParseInts.java

However, as it stands this program is extremely frail. For example, if I input a string "`20 56`", we receive the following output.

```
Enter a line of text: 20 56
The sum of the integers on this line is: 76
```

On the other hand, adding a single space to the beginning of the input, let alone a character that is not a number, produces a disastrous result.

```
Enter a line of text:  55 66
Exception in thread "main" java.lang.NumberFormatException: For input string: ""
  at java.base/java.lang.NumberFormatException.forInputString(Unknown Source)
  at java.base/java.lang.Integer.parseInt(Unknown Source)
  at java.base/java.lang.Integer.parseInt(Unknown Source)
  at ift_labs/lab_6.ParseInts.main(ParseInts.java:18)
```

One way to fix this error is to encapsulate the loop in a `try` block, as in Figure 3.

Throwing Exceptions

In this section we're given the two classes in Figure 4, which I've rewritten to compute factorials of input integers *safely*. Below is an example session with the program.

```
Enter an integer: hello!
*** Error: please enter an integer
Enter an integer: 15
15! = 2004310016
Compute another factorial? [y|n]: y
Enter an integer: 13
13! = 1932053504
Compute another factorial? [y|n]: n
Exiting.
```

```java
package lab_6;

import java.util.Scanner;

public class ParseIntsBetter
{
    public static void main(String[] args)
    {
        try (var scnr = new Scanner(System.in)) {
            int sum = 0;
            String line = "";

            System.out.print("Enter a line of text: ");
            line = scnr.nextLine();

            String[] values = line.split(" ");

            String damagingValue = "";
            try {
                for (String number : values)
                {
                    damagingValue = number;
                    sum += Integer.parseInt(number);
                }
                System.out.printf("The sum of the integers on this line is: %d", sum);
            } catch (NumberFormatException ex) {
                System.out.printf("*** Error: Expected a number, received \"%s\"",
                    damagingValue);
            }
        }
    }
}
```

Figure 3: ParseIntsBetter.java

Conclusion

This has been a great experience!

I spent approximately 3 hours cumulatively writing this lab. I learned about writing and throwing exceptions in my own methods. I also think it would be very interested writing our own exceptions to be thrown in our packages.

I also think that it would be useful learning more about text processing, so that we don't have to throw exceptions every time some data doesn't look as it's supposed to.

A feature of pure functional languages like Haskell that I really admire is *totality*. The most important part of totality is, in my opinion, the fact that functions are guaranteed to have an output for every input. In reality/practice this can be extremely difficult to guarantee; for instance, even while computing something as fundamental as a continuous factorial (cf. the gamma function), we can run into a plethora of issues.

```java
package lab_6;

import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * Encapsulate the computation of some factorials.
 *
 * @author Brandon Doyle
 */
public class Factorials
{
    public static void main(String[] args)
    {
        try (var scnr = new Scanner(System.in)) {
            String keepGoing = "y";
            int val = 0;

            // As long as input starts with a Y of some form, keep looping.
            while (!keepGoing.matches("^[^yY].*"))
            {
                System.out.print("Enter an integer: ");

                try {
                    val = scnr.nextInt();
                } catch (InputMismatchException ex) {
                    System.out.println("*** Error: please enter an integer");
                    scnr.next();
                    continue;
                }

                // Print the factorial of this integer.
                try {
                    System.out.printf("%d! = %d\n", val, MathUtils.factorial(val));
                } catch (IllegalArgumentException ex) {
                    System.out.println("*** Error: " + ex.getMessage());
                    continue;
                }

                System.out.print("Compute another factorial? [y|n]: ");
                keepGoing = scnr.next();
            }

            System.out.println("Exiting.");
        }
    }
}

/**
 * Provide local utils for computing various mathematical functions.
 *
 * @author Brandon Doyle
 */
class MathUtils
{
    /**
     * Compute the factorial of an integer.
     *
     * @param n Number to compute the factorial of.
     * @return  The factorial of the input integer.
     * @throws  IllegalArgumentException when n is less than 0 or greater than 16.
     */
    public static int factorial(int n)
        throws IllegalArgumentException
    {
        // Use some recursion :D
        if (n < 0)
            throw new IllegalArgumentException(
                String.format("Expected a value n >= 0, received %d", n));
        else if (n > 16)
            throw new IllegalArgumentException(
                String.format("Expected a value n <= 16, received %d", n));
        return (n <= 1) ? 1 : n * factorial(n - 1);
    }
}
```

Figure 4: Factorials.java