3-ExploringExceptions.doc

CS 239 Activity – Exploring Exceptions

Content Objectives: At the conclusion of this activity students will be able to:

- Read an exception message and identify the exception class that produced it.
- Design exception handlers to "catch" exceptions that may occur.
- Identify situations in which it is best to prevent exceptions and when it is best to handle them.

Roles for this activity – Assign one person to be the team manager for today.

- MANAGER: Keep the group on task. Begin the group quiz when all are finished with the individual quiz
- RECORDER: All students will record answers on their own worksheets.

Part 1 – Exploring exceptions –Review of exceptions

```
public class Example1
{
   public static void main(String[] args)
   {
      int     denominator, numerator, ratio;

      numerator   = 5;
      denominator = 0;

      ratio = numerator / denominator;
      System.out.println("The answer is: " + ratio);
      System.out.println("Done.");
   }
}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at Example1.main(Example1.java:10)
```

Using the example program that you find above and the output displayed just below it, answer the following questions.

1. This is an example of a compile time error, a runtime error, or a logic error? _____
2. On what line did the program fail? _____ How do you know? _____
3. What error occurred? _____ How do you know? _____
4. What is the formal name of this error? _____
5. In front of the name of the exception, what do you see? _____

Part 2 – Exploring the Java API's for exceptions

This is an example of an ArithmeticException, a special kind of class in Java.

Using the JAVA API displayed in the front of the room, answer the following questions:

1. In which package is the class ArithmeticExpression found? (In other words, if you were to need to import it, what package name would you use in the import.)_____

2. Do you have to import this class? Why or why not? _____

3. How many constructors do we have for an ArithmeticException?_____

Part 3 – How can we keep the program from failing? Recognizing and guarding against possible run-time errors.

```
 1 public class Argue
 2 {
 3    /*****************************************************************
 4     * A main method to demonstrate incoming command line arguments
 5     * @param args An array of String that may be empty or may contain
 6     *             values that we will use in a program
 7     *****************************************************************/
 8    public static void main(String args[])
 9    {
10       int count;
11
12       if (args.length > 0)     // must check to see if there is anything
13       {
14          System.out.printf("I am an argument: %s\n", args[0]);
15       }
16       else
17       {
18          System.out.printf("There are no arguments.\n");
19       }
20
21       count = Integer.parseInt(args[0]);
22
23       for (int ii = 1; ii <= count; ii++)
24       {
25          System.out.printf("Count: %d\n", ii);
26       }
27    }
28 }
```

1. In your own words, what is this program doing?

2. Which line(s) have the potential for run-time errors? For each, what is the run time error you may get?

3. What are some ways you could insure that you don't generate a run-time error?

3-ExploringExceptions.doc

Part 4 – How can we keep the program from failing? Exploring try / catch.

```
1 public class Argue
 2 {
 3    /*****************************************************************
 4     * A main method to demonstrate incoming command line arguments
 5     * @param args An array of String that may be empty or may contain
 6     *             values that we will use in a program
 7     *****************************************************************/
 8    public static void main(String args[])
 9    {
10       int count;
11       final int DEFAULT = 1;
12
13       if (args.length > 0)     // must check to see if there is anything
14       {
15          System.out.printf("I am an argument: %s\n", args[0]);
16       }
17       else
18       {
19          System.out.printf("There are no arguments.\n");
20       }
21       try
22       {
23          count = Integer.parseInt(args[0]);
24       }
25       catch (NumberFormatException nfe)
26       {
27          System.out.println("Bad argument " + args[0] + ", using " + DEFAULT);
28          count = DEFAULT;
29       }
30       for (int ii = 1; ii <= count; ii++)
31       {
32          System.out.printf("Count: %d\n", ii);
33       }
34    }
35 }
```

In this program, the parseInt() method call is surrounded by a “try” block. It is followed by a “catch” block. If an exception is generated by the actions in the try block, the system looks at the catch blocks associated. If a match of the exception class is made, the statements in the catch block are executed. In this case, if a NumberFormatException is raised by the parseInt() method, instead of failing, the program will execute the statements in the catch.

3-ExploringExceptions.doc

1. If you were to trace the execution of the program, Argue, using the following calls, what lines would execute and what would be output by the program? (List the line numbers, separated by commas.)

```
java argue 3
```


```
java argue cat
```


Part 5 – Constructing your own try/catch block.

Given the following documentation describing the substring() method in the String class, fill in the missing code in the StringPlay class.

substring

public String **substring**(int beginIndex)

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Examples:

"unhappy".substring(2) returns "happy" "Harbison".substring(3) returns "bison" "emptiness".substring(9) returns "" (an empty string)

Parameters:

`beginIndex` - the beginning index, inclusive.

Returns:

the specified substring.

Throws:

`IndexOutOfBoundsException` - if `beginIndex` is negative or larger than the length of this `String` object.

3-ExploringExceptions.doc

```
 1 import java.util.Scanner;
 2 public class StringPlay
 3 {
 4    /*****************************************************************
 5     * A main method to use command line arguments to print a substring
 6     * @param args An array of 2 values, the first is a String and the second
 7     * a starting position for the substring
 8     *****************************************************************/
 9    public static void main(String args[])
10    {
11       String sub;       // the substring
12       Scanner lineScan; // a Scanner to help parse the second argument
13       int start;        // the starting position for the substring.
14
15       if (args.length != 2)     // must check to see if there is anything
16       {
17          System.out.printf("You must have exactly two arguments. Program ending");
18          System.exit(1);
19       }
20       else
21       {
22          lineScan = new Scanner(args[1]); // set up to get the start position
23          if (lineScan.hasNextInt())
24          {
25             start = lineScan.nextInt();
26          }
27          else
28          {
29             start = 0;
30          }
31          // in the following lines, print the substring of the first argument
32          // beginning at the second argument position
33          // if the position is larger than the length of the string, catch the exception
34          // and print an error message, including the value of the argument 0 and the argument
1.
35          // you must use a try catch
36
37
38
39
40
41
43
44
45
46
47
48
49
50       }
51    }
52 }
```

Part 6 – Read more about it. 12.1 (Puzzle), 11.1(Watermelon)

HOMEWORK – Finish this worksheet then try Algorithm Workbench, #1, 2

Not homework, but pertinent to understanding, Review Questions and Exercises, questions, page #1, 2, 3, 4, 6, 8.