Practical 7: Java Exceptions

Before embarking on this practical you are introduced below to two new pieces of programming knowledge that will also be covered at greater length in the lectures very soon. You may find that you need these during the tasks of this practical.

Variable Scope

The *scope* of a *local* (or method) variable starts from the point where it is declared and ends at the end of the *block* that encloses it. Block boundaries are marked by `{}`. This is relevant to you at this point because if you declare a variable in the `try` block of a `try/catch` structure (for dealing with an exception) then you will not be able to access this variable outside this block. For example:

```
...
try {
    int value = 3;
    ...
} catch(Exception e) {
    System.out.println("An unknown exception has occurred!");
}
System.out.println(value);
...
```

The compiler will not compile the above code segment, because variable `value` is **out of scope** at the point of the last line. The correct implementation should be:

```
...
int value = 0;
try {
    value = 3;
} catch(Exception e) {
    System.out.println("An unknown exception has occurred!");
}
System.out.println(value);
...
```

Can you understand why the above will now work?

Exiting before the end of the program

You can force your program to terminate at a point of your choosing with the following instruction:

```
System.exit(0);
```

The value in the brackets (zero in the above example) is a code returned to the operating system to inform it of the exit status of your program. By convention, a nonzero status code indicates abnormal termination.

Task 1

An `ArithmeticException` is an **unchecked** exception in Java. Such an exception can be caused by dividing an **integer** by 0 (also represented as an integer).

Write a simple program to prove to yourself that both the above statements are true.

Improve your code by **catching** the `ArithmeticException` and displaying your own message when it occurs.

Task 2

Compare the above to what happens when you divide a **fractional** data type (i.e. a `float` or a `double`) by 0.

Task 3

Write a program that prompts the user to provide two fractional numbers and then outputs their ratio (i.e. the result of dividing one by the other).

Your program should be made to deal with the `NumberFormatException` and the `IOException` explicitly (i.e. using a `try/catch` structure for both exceptions).

Test your program.

Task 4

Every exception object has a string attribute associated with it that gives some explanation about the nature of the exception. You can print this string using:

```
...
System.out.println(e.toString());
...
```

Where `e` in the above example line is the name of the exception object. In fact, the above can be the only statement in the `catch` block of the exception handling mechanism.

Modify your code in task 3 in order to display the exception's own message string instead of (or as well as) your own when an exception happens. Test your program.

It is sometimes a good idea to print this string instead of (or as well as) our own. This is especially true when you are trying to catch **generic** exceptions such as the `IOException`. Can you explain why?