[0720] An Exceptional Solution Appears...

All of those primitive error handling problems disappeared when Exceptions came on to the scene.

Done correctly, structured exception handlers allow you to segregate your error handling code from the rest of your program, preserving the "normal, readable, logical" flow of your code. Yay!

(Done incorrectly, exception handlers can cause more problems than they solve, so you want to pay attention to this module!)

The basic idea about exceptions is that when a run-time problem happens inside of your program, the program is halted and the system generates a special "exception" object containing technical details about the error. That exception object is then "thrown" to the program - specifically, the block of code where the exception happened - by the system in the hopes that the program will "catch" it and take some corrective action. If the current block of code doesn't catch the exception, it is passed back to the previous method on the stack to see if it can catch it, and so on all the way back up the call stack.

If the exception reaches the top of the call stack without being caught inside the program, then it is caught by the system's default exception handler which prints a very technical message on the screen and halts the program. User's HATE seeing exception messages on their screen because it means that the program has crashed and their work has been lost. It is every programmer's responsibility to ensure users never see exception messages.

Here's what an exception message looks like for a console application:

```
Unhandled Exception: System.IndexOutOfRangeException: Index was outside the bounds of the array.
at Deck.Shuffle() in /PlayingCardSimulation/PlayingCardSimulation.cs:line 70
at PlayingCardSimulation.Main() in /PlayingCardSimulation/PlayingCardSimulation.cs:line 206
bash: line 1: 2829 Abort trap: 6 "/usr/local/share/dotnet/dotnet" "/PlayingCardSimulation/bin/Debug/netcoreap
p2.0/PlayingCardSimulation.dll"
```

Basically, that's the call stack with the more recent methods on the top and with line numbers showing you where each method was called from. It may be useful to you, the programmer. It is definitely horrifying to your users however.

In this case, we see that we were executing the code on line #70 of our program file - inside of a method called "Deck.Shuffle()" when we tried to access an array element using an index that was outside the bounds of the array. Hopefully, the code on and around line #70 will show us what went wrong. (Note: If you are running Visual Studio when an exception happens, it automatically displays the line with the problem.)

So our goal as programmers is to make sure that users never see exceptions. We do that by "catching" them - but we need to be sure we are catching them APPROPRIATELY!