GDAPS1 – Practice Exercise

Exceptions

# Objective

Practice catching existing C# exceptions, and then practice throwing your own exceptions and catching them.

# Causing Exceptions

This part can be done in the Main() method. You will write code that definitely throw some exceptions.

Start off by creating a List of strings using the parameterized constructor that specifies initial capacity. Use a **negative number** for its capacity.

Next, add three names to your list.

Lastly, print some of the values using the list's *indexer* (square brackets with an index, like so: names[0]). Use one good index, an index that's too low, and an index that's too high.

An **indexer** is a special property that allows you to get a single element/index from a data structure.  
Whenever you write [ ] with an index inside the brackets, you are utilizing the data structure’s indexer.

The last thing you should do is print a line that says "Done!"

Run your program. It should crash.

# Catching Exceptions

You can obviously fix your errors by using correct parameters to the constructor or the indexer. That would normally be the correct thing to do. However, today we're interested in utilizing *try/catch* to see how it works.

Put the List instantiation inside its own *try* block.

List declaration and instantiation are 2 different steps.   
The List declaration simply declares the List, like so: List<string> myList;  
Whereas initialization creates the new List object: myList = new List<string>(-5);

Remember how try/catch affects variable scope and adjust your code accordingly.

Inside the *catch* block, print a friendly error message to the user, followed by the message associated with the exception you've caught. Then instantiate the List using its default constructor instead - still inside the catch.

*You* ***don't*** *need to put a try block around the calls to .Add(), as that method doesn't throw exceptions.*

Write another *try* block around the code that prints out the list's contents using the good and invalid indices. The catch block will contain a statement that prints the phrase "Error! Stopping all code in TRY".

Run your program. It should **not** crash this time.

# When to use Try/Catch

In both of these cases – using a negative number for a starting list size and accessing invalid indices in an array/list – you could have absolutely fixed your errors instead of relying on try/catch. If you were working with user input, you could have written conditional statements to verify the data is valid before using it.

*That's the preferred way to do things.* But today we wanted you to practice with try and catch, so we asked you to do it this \*wrong\* way first. ☺

Next you'll practice throwing your own exceptions in a class and catching them in Main.

# Creating your own Custom List

Create a class called CustomList. This class will be the very start of a List class that you write yourself – it’s a brief introduction into writing your very first data structure!

CustomList needs the following class members:

**Fields:** An integer for the count of your list, and an array of strings to hold the list’s data.

**Default constructor:** Initialize the string array to hold 4 elements and set the count to zero.

**Parameterized constructor:** Receives the initial size of the array as a parameter. Check to see if the initial size is less than zero. If it is, throw an ArgumentException and place an appropriate error message as a parameter in the exception’s constructor, like: “Error creating your list! Non-negative number required. Parameter name: capacity”. If the initial size is greater than 0, initialize the array to the given size and set the count to zero.

**Methods:**   
**string GetElement(int index):** This method should return the data at an index. Be sure to test the given index to ensure validity – if it's too high or too low, throw an IndexOutOfRangeException and pass an appropriate error message into the exception’s constructor. Otherwise, return the data found that that index in the array.

**void Add(string item):** Add the item to the proper place in the data array. For simplicity in this exercise, you can simply skip adding an item if the array is out of room. No need to throw an exception here, as you'd normally resize the array as necessary when full.

# Testing your Custom List

Back in your main method, you should be able to replace the List with your newly created CustomList.

Code inside the first try block should run just fine, as long as you are using your CustomList object instead of the built-in List. Code in the catch should be updated to use the CustomList, too.

Code inside your second try block will need to change a tiny bit. Instead of using the built-in-List class’s indexer to retrieve data, call GetElement() and pass the index in as a parameter.

This will not work with your *CustomList*: yourListName[4];  
But this will: yourListName.GetElement(4);

Run the program to ensure that your CustomList is properly throwing exceptions, and that your Main() method is properly catching them.

# Sample Run

Welcome to the Exception exercise!

Error creating your list!   
Non-negative number required.   
Parameter name: capacity

Printing names in list:   
Timothy  
Error! Stopping all code in TRY

Done!

# Submission

All of your work must be commented and follow this course’s coding standards. **Read through the Coding Standards document (located in MyCourses) to check over your code before you complete your program. Make sure you follow the coding standards for all code you create.**

1) Submit: Submit your program to the appropriate Assignments dropbox in MyCourses.

2) Check-off: Show your working program to the instructor or TA. If you do not finish before class ends, complete the exercise for homework and show one of us in-class on the next class period. If your program works as expected, you will be “checked off” to earn credit for the exercise.