# Reimplement Your Stack, Using The ErrorCode enum

**The goal for this exercise** is to make sure that you can use an enum. You will also modify the SmartArray class – so much so that you'll actually put all this code into a new project.

**What you need to do for this exercise:**  On the page for this lesson you should find a link that reads similar to “Link to access the project for this assignment in GitHub”. Follow that link in order to access the starter Solution for this particular exercise. Please note that this starter Solution/project is completely separate from the starter files for the rest of the exercises in this lesson.

Please also note that in order to hand in this exercise you’ll need to commit your work into your local git repo and then push those changes back to your copy of the repo that’s stored in GitHub.
If you need a refresher on how to do that please review the material from the previous Lesson and/or make use of office hours, the tutoring hours in the Bock Learning Center, and/or Google/Bing/etc.

In that starter Solution you’ll find a project named **SmartArray\_Enums\_Test** (not the SmartArray\_**EH**\_Test project), create an enum to define a new type named ErrorCode, then re-implement the SmartArray class so that it uses the new enum.

The purpose of this enum is to list the possible responses when your SmartArray (or LinkedList, or Stack, or Queue,etc) class is asked to do something. (Note that you haven’t yet seen the Stack/Queue in this class, but you will soon). Hopefully, if someone tries to, say, add a new item to the LinkedList, everything will go fine, in which case the appropriate method will return the "No Error" error code. On the other hand, it's possible that there isn't any space left in your computer's memory, (and thus, no space in the LinkedList). In that case, your method should return the OverFlow error code.

Your SmartArray ( / LinkedList / Stack / Queue) class may generate the following types of errors, and so your ErrorCode enum should have a value for each:

* NoError
If a method (such as Add) does everything correctly,then it should return this Error Code, in order to indicate that everything went fine.
* OverFlow
If you try and add something to the data structure, and there's no more space left, the Add method should return this error code.
While the early versions of your SmartArray class clearly can return this error, it's possible that other classes (such as your LinkedList class) might never return this value. Similarly, if you build your Stack/Queue on a LinkedList, then your Stack/Queue will never return this. You should include this error code anyways, in case you decide to change how you implement things later (maybe to have your Stack use an array, for example), or in case you want to re-use your ErrorCode somewhere else.
* UnderFlow
The LinkedList/Stack/Queue is empty, meaning there's nothing there. Return this ErrorCode if the Delete method is called, and there's nothing in the LinkedList/Stack/Queue. This value should also be returned if the LinkedList/Stack/Queue is empty, and someone calls the "get an element" method on it.

Next, implement the SmartArray class, as defined below. You are encouraged to copy code from your existing SmartArray class.

 Note that most of the code in the **SmartArray\_Enums\_Test** project is commented out. If you select everything in the file (by pressing the Control+A keys), then use the Edit 🡪 Advanced 🡪 Uncomment Selection menu item, you can uncomment everything quickly and easily.

In the table below, you'll see that stuff that hasn't changed has been grayed out, stuff that has changed has been marked in red.

|  |
| --- |
| **SmartArray Data & Methods** |
| Data Field Name | Type | Description: |
| rgNums | Array of ints | A reference to an array of integers. |
| Note: all data fields should be marked **private** |
| Method Name | Returns | Description/Parameters: |
| Default constructor | <nothing> | Allocates an array of 5 integers.The rest of the class must be written so that if we were to change this to be, say, 10 integers, all other methods would function still correctly. |
| **Second, overloaded constructor** | **<nothing>** | **Parameters:**1. **An integer that specifies what the size of the array should be**

**Allocates an array (the size is given by the parameter).** |
| SetAtIndex | A value from the **ErrorCode** enum.This will be:1. **NoError**If everything went well
2. **Overflow**If the index is too high
3. **Underflow**If the index is too low
 | Parameters:1. An integer that is the index of the element to set
2. An integer that is the value to set that element to

Conceptually, **SmartArrayObject.SetAtIndex(10, 20)**this will accomplish the same sort of thing as saying**Array[10] = 20;**would accomplish with a normal array |
| GetAtIndex | A value from the **ErrorCode** enum.This will be:1. **NoError**If everything went well
2. **Overflow**If the index is too high
3. **Underflow**If the index is too low
 | Parameters:1. An integer that is the index of the element to get
2. An **out** parameter, that will hold the integer value at the array slot, or **Int32.MinValue** if the method is unable to access the slot, for any reason.

Conceptually, **SmartArrayObject.GetAtIndex(10);**this will accomplish the same sort of thing as saying**Array[10];**would accomplish with a normal array |
| PrintAllElements | Nothing (void) | Parameters: NonePrint all elements of the array, one per line, onto the console. |
| Find | **true** if at least one element in the array is the same value as the parameter, **false** if the given value isn't present in the array. | Parameter:1. An integer that may or may not be in the array.

This method takes it's parameter, and sees if that value is located anywhere within the array. If it finds even a single slot of the array with the same value as the parameter, it will return true. If the value isn't found anywhere within the array, it will return false. |
| Note: all methods should be marked **public** |