# Review: Classes (public/private, methods, data fields), Arrays

**The goal for this exercise** In this lesson, we're going to build up to creating a class that manages access to an array for us. We'll start by reviewing how to deal with an array of simple types (such as integers) that is managed by a class that we create.

**What you need to do for this exercise:**

In the starter solution, you will need to fill in the C# class named SmartArray. SmartArray should provide the methods listed below. The major goal of this exercise is make sure that you're familiar & comfortable with classes in C#, arrays / reference types, and that you can create a class that manages a collection of data. Thus, you should fully encapsulate your object-oriented SmartArray class, as you saw in previous lessons

|  |
| --- |
| For this exercise, you have been provided with a project that contains several tests for your SmartArray class. These are tests written using if statements to check results and Console.WriteLine to tell you whether the tests have passed or failed – **they are completely separate from the NUnit tests**, which you’ll see in a future section.  When you're done, your SmartArray class should pass 100% of the tests.  As the term goes on, we'll be looking at test-driven development, as a means of helping to get you in the habit of writing tests to verify that your code works (at least minimally). You may want to start this exercise by running the example code, and noting which tests pass, and which fail. For the tests that pass, think about whether it's a good test or not, given that it passes, even when the class clearly doesn't work. |

<The SmartArray class is described on the next page>

|  |  |  |
| --- | --- | --- |
| **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 integer, all other methods would function still correctly. |
| SetAtIndex | **true** if array slot was successfully set, **false** if it didn't work (for any reason) | 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 | The integer value at the array slot, or **Int32.MinValue** if the method is unable to access the slot, for any reason. | Parameter:   1. An integer that is the index of the element to get   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: None  Print 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** | | |