**Workshop Registration Project Requirements**

you have been tasked with creating a registration application for a client who runs a small workshop. This program will be created in Visual Studio using C#.

The application will have the following features:

1. Add a new student to the workshop.
2. Display a list of all students in the workshop, sorted by their grades, highest to lowest.
3. Clear the students from the workshop.
4. Update the grade of a student in the workshop.
5. Quit the application.

You should develop the application as follows:

**Create a student class**

Create a class to create student objects. This should be its own .cs file. The objects should contain the following properties.

| Student Class |
| --- |
| First Name – string  Last Name – string  Student ID – string  Grade – integer |

The student ID should be in the following format: First Initial + Last Initial + Random shuffling of your Conestoga student number

E.g.: If my Conestoga student number was 123456, and I was dealing with student Alice Brown, then a valid student ID would be AB524136.

**Create the application:**

The application should be structured as shown in Figure 1.

| **Main Method**  Ask what the user wishes to do:   1. Add a new student. 2. Update Grade 3. Display list of students by grade 4. Clear list of students 5. Exit     Create an array to hold the students enrolled in the workshop. Since it is a small workshop, you only need to hold up to 10 students. Students will always be stored in this array by their grade, from highest grade at index 0 to lowest grade at index 9.    When Adding a new student, the Main method should collect a first name and last name from the user, and then call the Create Student method.    The functionality for Displaying the list of students can be done within the Main method.  The functionality for Exiting the application can be done within the Main method. | Midterm application structure  **Figure 1. Application Structure**    When Updating a grade, the Main method should ask for the index of the student to update (0-9), and for the new grade (0-100), and then call the Update Grade method.  The functionality for Clearing the array can be done within the Main method. |
| --- | --- |
| **Create Student Method**  Input: First name, last name, student array  Output: None    Purpose: Create a student object and add it to the array. All new students begin with a grade of zero. Throw an exception if the array is full, you can catch this in the Main method. Call the generate value method to get the number portion of the student ID (you can collect the initials from the first name and last name you already have).    **Generate Value Method**  Input: None  Output: String (your rearranged student number)    Purpose: Create a randomly shuffled value of your Conestoga ID (You can hard code your Conestoga ID into the method). | **Update Grade Method**  Input: Student array, Index value, Grade  Output: None    Purpose: Update the grade of a student at a given index to a given new grade. After updating the grade, you need to call the sort method.    **Sort Method**  Input: Student array  Output: None    Purpose: Sort the array of students by grade. |

**Testing Tasks:**

**Task A**

Create a project containing just the four methods: Create Student, Generate Value, Update Grade, and Sort. You can use the "Class Library" project in Visual Studio when creating a new project. This project will contain 5 .cs files (the four methods and one for student class). Create **two** unit tests using MSTest for each method. Your MSTest project will have 4 .cs files, one matching each of the four methods. You do not need to unit test the student class .cs file.

**Task B**

Create a project containing a .NET Core console application. Create the Main method as described above. It should initially be created as a single unit, using stubs where required. You do not need to unit test the Main method.

Perform a series of top-down integration tests of the program. Demonstrate 1 test per integration. The tests should be performed through the Main method interface. Integrate until complete. Do one project in Visual studio per integration (you can copy and paste as you go).

**Task C**

Perform 3 manual system tests, demonstrating different functionalities of the completed program.