For this program you will work from the code base for the SchoolDB solution and project from recent class exercises. Here is the spec for the parts of the program that you are to submit when complete:

The test driver should make a List of 8 data objects, 4 of each subclass kind, for testing. Use good test data. Good test data means that values are logical and correlate to other field values so that correct output becomes intuitively obvious to a casual observer. See more about testing below.

* Your inheritance hierarchy for the data objects will have the following spec:
  + Student is the base class. Undergrad and GradStudent inherit from Student.
  + Student(s) have a first name, last name, email address, and a GPA.
  + Undergrad(s) have a year rank in school that can only take on the literal values Freshman, Sophomore, Junior, or Senior and a degree program major.
  + GradStudent(s) have a graduate faculty advisor and they have a financial tuition credit for the teaching they do while in their grad programs.
* Your database application has the following requirements for basic operations:
  + Email address is a good choice for the primary key
  + C - Create or add a record to the database
  + R - Read the details of a record (this find or search utility will be useful for other CRUD methods)
  + U - Update or modify some details of an existing record in the database.
  + D - Delete or remove a record from the database.
  + Ability to Save the database between program runs (we are using a plain ASCII text file for this).
* Some additional requirements:
  + The program must read from and write to plain ASCII text files for its persistent storage.
  + C# allows any extension for text files but you will use **.txt** so that it opens easily using Notepad
  + Any content, format, or layout of data or meta-data in the text file is allowed as long as it is readable text
  + No other database software is allowed (other than the app code you write yourself)
  + During run-time the objects must be stored and manipulated in a **List<Student>**, not in the files.

**Testing** - Test reports and results should be obvious, even to a person with little C# expertise - eg - your manager.

Include the output of good test runs of your program so that an observer can immediately see, in an obvious way, that your program produces correct results. Use Powershell's Start-Transcript/Stop-Transcript or simply copy and paste from the shell to notepad, etc.

Your testing should completely test the code, meaning that at least one execution of every method and property get or set should be covered in a single run of the test. At least one record should remain unchanged from the input file - through the complete test run - to the output file. This proves that your database can store a record safely without modification. Changes to records should be distributed evenly among them so the changes are easily traced back to the original data.

**Create, Delete, and Update all need to be tested "end to end" meaning that the contents of the output file need to reflect the change, not just the List<>. This is a database and changes need to be persistent across program "run" instances to represent effective storage and operations.**

Note that somehow every test procedure should show 3 things in some fairly obvious way:

1- Pre-test shows the original value  - before the operation being tested begins  
2- The operation or feature being tested - eg - Delete, Update, etc.  
3- Post-test shows the new value - after the test is complete

***Ask questions in class discussions so all may benefit (don't email me).*** It's one thing to work with this system a little at a time in class, but another to see all the requirements written down. The requirements can build up very quickly until it is overwhelming. Make sure you understand what is intended before you spend time writing code so that you don't waste time going down the wrong path. As always check the rubric before submission so you know what is being graded.

Rubric

**R-oop**

| R-oop | | |
| --- | --- | --- |
| **Criteria** | **Ratings** | **Pts** |
| This criterion is linked to a Learning Outcomezip contents |  | 3 pts |
| This criterion is linked to a Learning Outcomedebug and release |  | 3 pts |
| This criterion is linked to a Learning Outcomesolution file |  | 3 pts |
| This criterion is linked to a Learning Outcomecode formatting |  | 3 pts |
| This criterion is linked to a Learning Outcomechg history, comments |  | 3 pts |
| This criterion is linked to a Learning Outcometest driver correctness |  | 5 pts |
| This criterion is linked to a Learning Outcomesubclasses |  | 5 pts |
| This criterion is linked to a Learning Outcomepolymorphism |  | 5 pts |
| This criterion is linked to a Learning Outcomearray of base class refs |  | 5 pts |
| This criterion is linked to a Learning Outcomesuitable test data |  | 5 pts |
| Total Points: 40 | | |

[Previous](https://canvas.uw.edu/courses/1523736/modules/items/15057258)[Next](https://canvas.uw.edu/courses/1523736/modules/items/13799999)