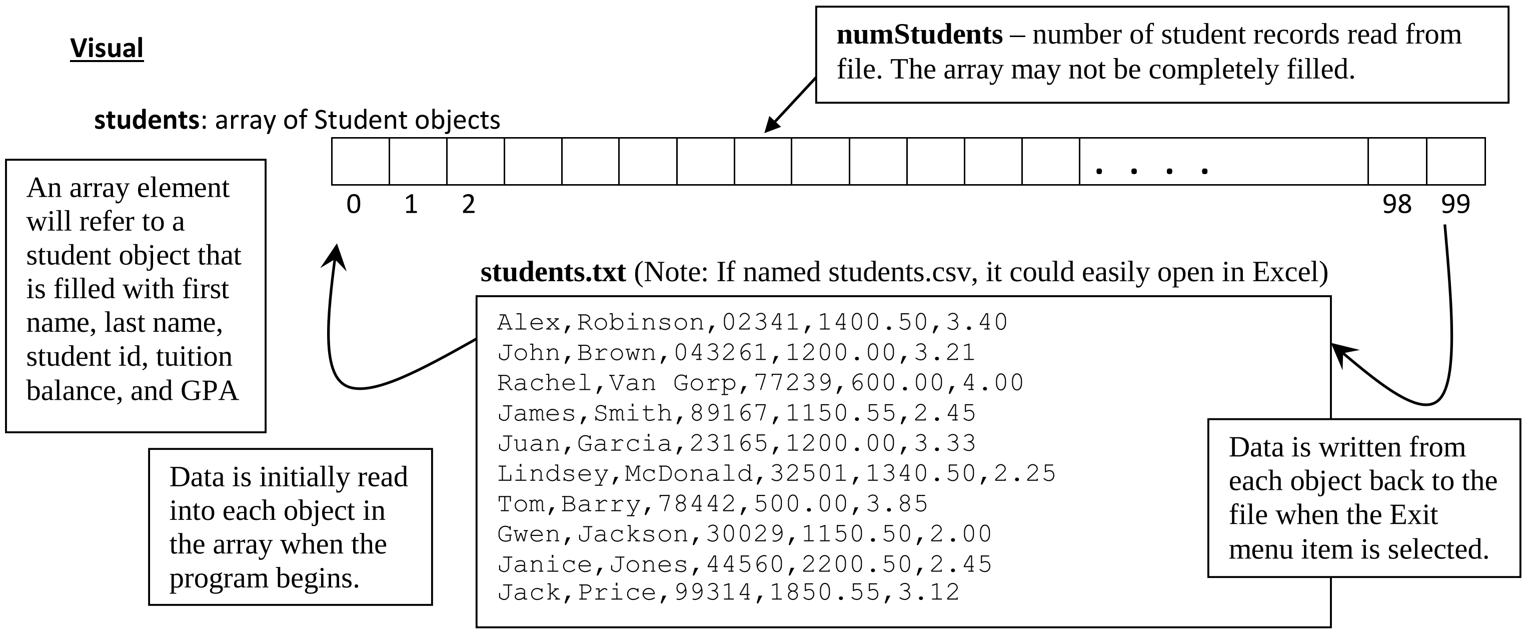
Student Information System Project (40 pts)   CS201

Overview

For this project, you will implement a small piece of a student information system. You will utilize files, classes, arrays, loops, if statements, and other coding constructs that we have used this semester.

Visual



numStudents – number of student records read from file. The array may not be completely filled.

An array element will refer to a student object that is filled with first name, last name, student id, tuition balance, and GPA

Data is initially read into each object in the array when the program begins.

Data is written from each object back to the file when the Exit menu item is selected.

students.txt (Note: If named students.csv, it could easily open in Excel)

```
Alex,Robinson,02341,1400.50,3.40
John,Brown,043261,1200.00,3.21
Rachel,Van Gorp,77239,600.00,4.00
James,Smith,89167,1150.55,2.45
Juan,Garcia,23165,1200.00,3.33
Lindsey,McDonald,32501,1340.50,2.25
Tom,Barry,78442,500.00,3.85
Gwen,Jackson,30029,1150.50,2.00
Janice,Jones,44560,2200.50,2.45
Jack,Price,99314,1850.55,3.12
```

Requirements

- Class MainClass is completed. Copy/paste the following code

```
using System;

/**
 * @author M. Van Gorp
 * Class MainClass will start the student information system
 */
public class MainClass
{
  public static void Main()
  {
    SISDriver driver = new SISDriver();
    driver.Run();
  } // end Main
} // end class
```

- Download students.txt from D2L and place into the source code folder of your project. Recall that this is the folder that will contain the .cs files.
- Add a public class named **Student** to the project in a file named **Student.cs**. The Student class has
  - class header documentation stating the author and a description.
  - public properties (not required to perform error checking such as for a negative tuition balance.)
    - FirstName (string)
    - LastName (string)
    - StudentId (string)
    - TuitionBalance (decimal)
    - Gpa (double)
  - a public constructor that has 5 formal parameters, each corresponding to a property. It sets each property to its corresponding parameter value.
  - a public void method named **MakePayment**
    - has header documentation.
    - receives a payment as a formal parameter and deducts the tuition balance by that amount.
  - a public method that <u>overrides</u> **ToString()**.

        - It returns a string in which the property values are separated by commas in the same order as shown in the data file. Use String.Format to help you.
        - Ensure that the GPA and tuition balance both have two digits of precision. Tuition will not use a currency format even though it is a decimal.
    - a public void method named **Store**
        - has header documentation.
        - has a StreamWriter formal parameter representing an opened output file stream.
        - writes the student's property values to the output file stream. This method contains only one statement as you are to utilize the Student's **ToString**() method.

Important Note: The SISDriver class shown next will work with an array of student objects. class Student has no knowledge of this array as it only pertains to one student. So when you create SISDriver please don't become somewhat confused from a design perspective and think that the Student class methods need to work with an array – they don't.

- Add a public class named **SISDriver** in a file named **SISDriver.cs**. SISDriver has
    - class header documentation stating the author and a description.
    - the following private fields:
        - a field named **students** that will eventually hold an array of Student objects. (The initialization is not here, but rather it is in the constructor)
        - a constant field named **MAX_STUDENTS** that is initialized to 100.
        - a field named **numStudents** that will eventually hold the total number of students that will be read from the file. It is initialized in the constructor. (You may assume that the number of students read from file will always be less than or equal to MAX_STUDENTS.)
        - An optional enumerated type for menu choices. Its constants will represent (1) displaying a record, (2)changing a gpa, (3)making a payment, and (4) exiting.

    - a public default constructor that
        - (has no parameters)
        - initializes the **numStudents** field to 0.
        - allocates the **students** array that may hold up to **MAX_STUDENTS** Student objects. (Note that the array elements will initially all be null.)

    - a public void method named **ReadData**
        - has header documentation
        - contains a string formal parameter that is the name of the input file (does not include a path).
        - opens the file from the source code folder in the project. (The folder that contains the .cs files)
        - each line in the file is one student record. As you loop to process each record,
            - you will need to create a **new** **Student** object, based upon the data just read, that is assigned to the current array element. In essence, you are filling array elements one student at a time from the file. The following code is helpful for the loop body, presuming that record was just read in the loop condition:

              ```
              string[] data = record.Split(',');
              students[i] = new Student(data[0], data[1], ...);
              ```

              data[0] contains the first name, data[1] contains the last name, and so on. You are not required to create named constants here for 2, 3, and 4. Do note that you will need to convert the data elements for the tuition and gpa.

            - keep track of the number of students read. This is held in the **numStudents** field. (You may assume that this will not exceed MAX_STUDENTS.)
        - close the file when finished
        - error/exception processing is not required, but may optionally be implemented.

- a public void method named **StoreData**
  - has header documentation
  - contains a string formal parameter that is the name of the output file (does not include a path).
  - opens this file in the folder that contains the project source files.
  - loops **numStudents** times to write each student array object to file. You must utilize the student's Store method. Example:
    ```
    students[i].store(outFile);
    ```
  - close the output file when finished with the loop.
  - error/exception processing is not required, but may optionally be implemented.

- a public int method named **LookUp**
  - has header documentation
  - contains a string formal parameter that represents a student id
  - Performs a linear search on the **students** array to find a student with that id. It returns the index of the student object if found, else it returns -1.
  - Note that to compare with each student object in the array, you will need
    ```
    students[i].StudentId
    ```

- A public void method named **Run**
  - has header documentation
  - (contains no formal parameters)
  - match the format shown for the sample run of the video online.
  - The algorithm to follow is:

```
// Declare any variables as needed. If not using an enumerated type
// for the menu choices, then also declare constants here that correspond
// to the menu choices. For example, rather than comparing a user's menu
// choice to the number 1, you could compare it to DISPLAY_STUDENT_RECORD
// be declaring const int DISPLAY_STUDENT_RECORD = 1;

// Call ReadData with "students.txt"

// Use a do-while to loop until the user chooses to exit.
// Inside this loop
//   1. Use Console.Clear() to clear the screen.
//   2. Display the menu and get the user's choice
//   3. If the choice is not to EXIT then
//      a. Prompt for and retrieve the desired student id
//      b. Call method LookUp to determine the array
//         index where that student is contained.
//      c. If LookUp did not return -1 then
//           1. If the menu choice was to DISPLAY THE STUDENT
//              record, then use Console.WriteLine and the ToString()
//              method of the student object at that array index to
//              display its data.
//           2. Else if the menu choice was to CHANGE THE GPA, then
//              a. Present the student’s Gpa with 2 digits of precision
//              b. Prompt for and retrieve the new gpa
//              c. Change the student's gpa to that value.
//           3. Else if the menu choice was to MAKE A PAYMENT, then
//              a. Present the current tuition balance in a currency format
//              b. Prompt for and retrieve the payment.
//              c. Call the student's MakePayment method with the payment.
//         Else
//           display that the Student was not found
//   4. Pause the program with
//         Console.Write("Press a key to continue ...");
//         Console.ReadKey();
// End loop
// Call StoreData with "students.txt"
// State a good-bye message after the loop
```

- **Remember to insert internal comments throughout your code; however, header comments for methods are only needed when specified above.**
- Note that when the program writes the output file when exiting, this file should be able to be read again when re-starting the program: The changed values should immediately be known and be read in without error.
- You may optionally work on this with one other student from the class – <u>but only if you both are providing equal thought and effort in developing the code</u>. If doing so, only hand in one copy of the project and upload to one of your D2L drop boxes.

<u>Submission</u>

- Before arriving to class:
    - Upload Student.cs and SISdriver.cs to D2L.
    - Print the code.
- Beginning of class: Staple and hand in Student.cs and SISdriver.cs