Milestone #1 – Part A

Due Sunday June 11th, 2017 – at 23:00
Late policy: 10%/day, maximum 2 days, including weekends.
Demo in class is a must at a time announced by the teacher.

Create 1 C# console application project with a menu offering options 1 to 3 with 0 to quit.

1. **Complete the BankAccount Class as seen in class.**
    a. Add a default constructor for the class starting with $0 balance.
    b. Add a non-default constructor.
    c. Use Properties
    d. Add class-level validation and application-level validation:
        i. Do not accept negative values.
        ii. User cannot withdraw more money than they have.
        iii. User cannot deposit $10000 or more in one transaction.
        iv. Balances cannot be negative.
        v. Constructors should call Properties for added input validation.
        vi. Properties throw errors when input is invalid.
    e. Error messages are displayed when appropriate.
    f. Once you have written the class, write a separate program that creates an instance of a BankAccount and interact with all its methods.

IMPORTANT (for #2 and #3):

- Code Properties instead of classic setters and getters.
- Use all methods when creating any instances of these classes: use all constructors and use all Properties.

2. Employee Class

Write a class named Employee that has the following backing fields:

- **name**. A string that holds the employee's name.
- **idNumber**. An int variable that holds the employee's ID number.
- **department**. A string that holds the name of the department where the employee works.
- **position**. A string that holds the employee's job title.

The class should have the following constructors:

- **A default constructor** that assigns empty strings ("") to the name, department, and position backing fields, and 0 to the idNumber member variable.
- **A constructor** that accepts the following values as arguments and assigns them to the appropriate backing fields: employee's name and ID number, department, and position.
- **A constructor** that accepts the following values as arguments and assigns them to the appropriate backing fields: employee's name and ID number. Empty strings ("") to department and position.

Create Properties for each backing fields that provide read and write access. No Property write access for Id number.

Once you have written the class, write a separate program that creates three Employee objects to hold the following data. You can hardcode the data instead of asking for input.

Create each employee by a different constructor.

Populate your data in a static way or from user input with this information:

Name	ID Number	Department	Position
Susan Meyers	47899	Accounting	Vice President
Mark Jones	39119	IT	Programmer
Joy Rogers	81774	Manufacturing	Engineer

The program should store this data in the three objects and then display the data for each employee on the screen, using the Properties or by calling its own implemented .ToString()

3. Car Class

Write a class named `Car` that has the following backing fields:

- `yearModel`. An `int` that holds the car's model year.
- `make`. A `string` that holds the make of the car.
- `speed`. An `int` that holds the car's current speed.

In addition, the class should have the following methods:

- **A default Constructor -** it assigns empty strings ("") to `yearModel` and `make` backing fields, and 0 to the `speed`.

- **A Constructor** - The constructor should accept the car's `yearModel` and `make` as arguments. These values should be assigned to the object's `yearModel` and `make` backing fields. The constructor should also assign 0 to the `speed` member variable.

- **Properties**
  - read access for `yearModel`, `make`, and `speed`, backing fields.
  - write access only to `yearModel` and `make`.

- `Accelerate` – This method should add 5 to the `speed` member variable each time it is called. The car is limited to a top speed of 160.

- `Brake` – This method should subtract 5 from the `speed` member variable each time it is called. Make sure to never allow the `speed` to go below 0.
- **Note** – `speed` member variable, can only be modified using the `Accelerate`, and `Brake` methods. Use constants instead of having "magic numbers" in your code.

Demonstrate the Car class in action by making a program that does the following:

1. Creates a `Car` object with default `make` and and `yearModel`.
2. Show a menu to the user with 3 options: `Accelerate`, `Brake or Exit`.
3. If the user chooses to `Accelerate`, the program outputs the current (faster) speed of the car.
4. If the user chooses to `Brake`, the program outputs the current (slower) speed of the car.

5. Separate the GUI components from the Class design. In other words, do not put Console.WriteLine() inside the Car Class. It should be in the void Main() program.

Deliverables

========

- **Find a teammate to test your application before submitting and before demoing to the teacher. Include the name of the tester as comments in your main() program header.**
- **Demo in class when the teacher requests it.**
- **Compress the Milestone project folder and submit on Lea. If the zipped file is bigger than 25MB then: use 7zip compression level Ultra and split to volumes of 25 MB each. It will create 2 or more 7zip files. Upload one by one on Lea (limit per file is 25 MB).**