**Character Creation Save Load**

**Project Description & Goal**

The goal of this project is to develop a project that simulates character creation and retention. We will be creating characters and saving them into a text file so that when we re-open our application we can continue where we left off.

**Project Specifications**

* Character class with 5 attributes
  + Can do something like age, height, eye color, etc.
  + Can also do something like strength, intelligence, dexterity, etc
* Menu with the following options
  1. Create a character
  2. Modify a character
  3. Delete a character
  4. Load Character (or load all if you are doing EC)
  5. Print Character (or print all characters if you are doing EC)
  6. Exit

**Required Methods**

* **ReturnType MethodName(Paramters)**
* **Character CreateCharacter(Character character)**
  + This method will ask the user for the attributes and create the character.
  + Return the created character.
  + Just before you return the character, save it to disk using the SaveCharacter() method.
* **void ModifyCharacter(Character character)**
  + As a parameter, send a character that you want to modify. Remember, because you are sending an object by reference as the parameter, any changes you make to this object persist throughout the program so you don’t have to “return” it.
  + This can be done a few ways, you get to make that choice. Here are some examples:
    - Do another menu along the lines of, “What attribute do you want to change? str, agi, int” and based on their input, change that attribute.
    - Go through each attribute 1 by 1. For example, “Bob currently has 5 strength, what would you like to change it to:”
  + Make sure to call SaveCharacter() again to overwrite changes on disk.
* **void SaveCharacter(Character character)**
  + This is a helper method, used in CreateCharacter() and ModifyCharacter() to take the Character object that is stored in RAM and save it to disk.
  + IMPORTANT: This method should first check to see if the file exists. If it does, prompt the user with, “This file already exists, are you sure you want to overwrite it? y, n”
* **bool DeleteCharacter(string characterName)**
  + Returns true if a character was successfully deleted, false otherwise
  + if a character file has the name characterName:
    - delete the file associated with that character
  + Main should reply with the result from this method in one of the following ways:
    - “Bob has been removed”
    - “Bob doesn’t exist, therefore was not removed”
* **Character LoadCharacter(string characterName)**
  + No Console.WriteLine() in this method, it should literally just create a character and return that character.
* **List<Character> LoadCharacters() – ONLY IF DOING EXTRA CREDIT!**
  + Finds all txt files in that location, and calls LoadCharacter() for all of them to populate a list which you will return to main.
  + See the slides to see how to do this.
* **void PrintCharacter(Character character) or void PrintAllCharacters(List<Character> characters)**
  + Print out to the console in a readable way

**Project Learning Objectives**

* Learn how to put together many things we have learned thus far.
* Learn how to look through files programmatically and manipulate them as you see fit.
* Learn how to load and save data.
* Learn how to handle edge cases.

**Project Demonstrated Competencies**

1. Project contains the methods listed in the “Required Methods” section. Additional methods can be used.
2. User can create a character to store in RAM, and eventually save that character on disk.
3. User can modify the character’s attributes stored in RAM, which is then saved back to disk.
4. User can delete characters, both in RAM and on disk.
5. With a single user input, every character file on disk can be loaded into a list.
6. Have an option to print the list of characters to show what data is available.

**Rubric**

|  |  |  |
| --- | --- | --- |
|  | **Description of perfect implementation** | **Score** |
| Competency #1 | Every method listed above is used in the program. Additional methods can be created/used as well. | \_\_\_  25 |
| Competency #2 | Character object in RAM is correctly created and attributes applied. This same character is then translated into a .txt file on disk | \_\_\_  25 |
| Competency #3 | Character object in RAM can be modified to have altered attributes. After the change, the .txt file associated with this object will be updated to match. | \_\_\_  25 |
| Competency #4 | User can delete characters from RAM (remove them from the list), and at the same time remove them from disk (delete the .txt file) | \_\_\_  25 |
| EC: Competency #5 | Given a set of .txt files with character data, a fresh instance of the program can be started and correctly load that data into a List<Character>. | \_\_\_  +25 |
| EC: Competency #6 | Iterate through the list of characters to print an easily readable portrayal of the characters and their attributes. | \_\_\_  +25 |
| Extra credit: +50 is the maximum amount of EC and it will require a very interesting and complex idea. Simple solutions like the examples will result in ~15 points. | Create some sort of game or tool that utilizes the list of characters in an interesting way.  Example: Create a simple “Rock Paper Scissors” type of game where intelligence beats strength, agility beats intelligence, strength beats agility and have your characters face up against each other.  Example: Create tasks that need specific attributes to complete, and look to see if you have a character that can complete the task. A task might be, “Help friend move into 3rd floor apartment”. This task might require 7 strength (for moving the objects), 5 intelligence (for figuring out how to get around those tricky corners), and 2 agility (considering you have all day to do this). | \_\_\_  +50 |