COMP123 – Programming 2

Final Exam - Practical
Character Builder

Due Week #14 (August 14th, 2019) by end of class

Value 25%

Character Builder – Version C

Time Limit: 1 hours 40 minutes **Maximum Mark:** 48

Overview: Using the Project Template Provided, complete the project that generates the first few pages of a Role-Playing Game Character. You will complete the project by adding the missing code, where appropriate, any controls and forms that are missing.

Instructions :

(11 Marks: GUI, 60 Marks: Functionality, 5 Marks: Internal Documentation, 4 Marks: Version Control)

Part A – Use the **`CharacterGenerationForm`** for this part. A Tab Control has been added to the Form (**`MainTabControl`**) and includes a series of **`TabPages`** including **`IdentityPage`**, **`AbilityPage`**, **`SkillsPage`**, and **`CharacterSheetPage`**.

1. **IdentityPage: This Page is partially complete (SubTotal: 20 Marks: Functionality)**
    a. Create a two string lists: FirstNameList and LastNameList (1 Mark Functionality).
    b. Create a method called **LoadNames** that loads the entire **firstNames.txt** and **lastNames.txt** file into **FirstNameList** and **LastNameList** respectively (7 Marks Functionality).
    c. Create a method called **GenerateNames**. When called, this method will randomly pick First Names and Last Names from the **FirstNameList** and **LastNameList** controls and set the **FirstNameDataLabel** and **LastNameDataLabel** with these values (6 Marks: Functionality).
    d. In the **CharacterGenerationForm's Load** event handler. Call the **LoadNames** method and call the **`GenerateNames`** method (2 Marks: Functionality).
    e. In the **GenerateNameButton's** click event handler, call the GenerateNames method (1 Mark: Functionality).
    f. In the **Program.cs** file, create a public static instance of the **CharacterPortfolio** class named character. Ensure that the constructer is set to public (1 Mark Functionality).
    g. In the **GenerateNameButton's** click event handler, set the value of the **Identity**.**FirstName** property of the **Program.character** object to the value of text property of the **FirstNameDataLabel** control (1 Mark: Functionality).
    h. In the **GenerateNameButton's** click event handler, set the value of the **Identity**.**LastName** property of the **Program.character** object to the value of text property of the **LastNameDataLabel** control (1 Mark: Functionality).

2. **AbilityPage: This Page is partially complete (SubTotal: 8 Marks: Functionality)**
    a. Write a method that allows the user to generate random numbers ranging from 1 to 15 for each Ability (Strength, Dexterity, Endurance, Intellect, Education and Social Standing). (6 Marks: Functionality).
    b. Store the values that are generated in the Program.character container class (2 Marks: Functionality)
3. **SkillsPage: You must complete this page (SubTotal: 5 Marks: GUI, 11 Marks: Functionality)**
    a. Add a **TableLayoutPanel** to the form for Page Layout purposes. Adjust the Table Layout to include at least 4 columns and 4 rows. (2 Marks: GUI).
    b. Add a series of Labels so that at least 4 skills can be displayed on the page. Include a header label that displays "Skills" in the text property (2 Marks: GUI).
    c. Add a GenerateSkills Button to the page (1 Mark: GUI).
    d. Create a string **List** named SkillsList (1 Mark: Functionality).
    e. Create a method called **LoadSkills** which loads the entire **skills.txt** file into the **SkillsList** (4 Marks Functionality)
    f. In the **CharacterGenerationForm's Load** event handler. Call the **LoadSkills** method (1 Mark: Functionality).
    g. Create a **GenerateRandomSkills** method that picks 4 random skills for the **SkillsList** and assigns them to the text properties of the Labels that you created above (4 Marks: Functionality).
    h. In the GenerateSkills Button's click event handler, call the **GenerateRandomSkills** method (1 Mark: Functionality).
4. **CharacterSheetPage: This form is partially complete (6 Marks: GUI, 21 Marks: Functionality)**
    a. Ensure that you include a TableLayoutPanel with at least 4 rows and 4 columns on this page (1 Mark: GUI).
    b. Create a series of labels on this page that can display the character's generated Name (FirstName and LastName), their abilities (Strength, Dexterity, Endurance, Intellect, Education and Social Standing) and their 4 randomly generated skills (5 Marks: GUI).
    c. When this page is loaded (i.e. when **the MainTabControl.SelectedIndex** is equal to 3) then fill the text properties of the Label controls you included above to display the "current" values stored in the Program.character container class (6 Marks: Functionality).
    d. When the user Selects the **Save option** on the menu or clicks the **save button**, a **SaveFileDialog** is displayed that allows the user to save the current state of the Program.character container class data to a text file (6 Marks: Functionality).
    e. When the user Selects the **Open menu option** on the menu or clicks the **open button**, an **OpenFileDialog** is displayed that allows the user to open a previously saved character. The character's properties will be displayed in the Labels you created above (6 Marks: Functionality).
    f. Ensure that when the user selects the **Exit menu** option that the application exits (1 Mark: Functionality).
    g. Ensure that when the user selects the About option or clicks the info button that an AboutBox is displayed (2 Marks: Functionality).
5. Include **Internal Documentation** for your Application **(5 Marks: Internal Documentation):**
    a. Ensure you include a **comment header** for each of your **C# files** that indicate: your **Student Name**, your **Student ID**, and **a brief description of the file** (2 Marks: Internal Documentation).

b. Ensure you include a **section headers** for all of your **Event Handlers, Classes,** and any **functions** (1 Marks: Internal Documentation)
c. Ensure all your code uses **contextual variable names** that help make the files human-readable (1 Marks: Internal Documentation).
d. Ensure you include **inline comments** that describe your GUI Design and Functionality. (1 Mark: Internal Documentation).

6. Share your files on **GitHub** to demonstrate Version Control Best Practices **(4 Marks: Version Control).**
    a. Your repository must include **your code** and be well structured (2 Marks: Version Control).
    b. Your repository must include **commits** that demonstrate the project being updated at different stages of development – each time a major change is implemented (2 Marks: Version Control).

SUBMITTING YOUR WORK

Your submission should include:

1. A zip archive of your project files
2. A link to your project files on GitHub.

Feature	Description	Marks
GUI / Interface Design	UI Controls meet the application requirements. Display elements are deployed in an attractive manner. Appropriate contrast is applied to application UI Controls and any background colours applied so that all text is legible.	11
Functionality	The program's deliverables are all met and the program functions as it should. No errors appear as a result of execution. User Input does not crash the program.	60
Internal Documentation	A program header is present and includes the name of the program, the name of the student, student number, date last modified, a short revision history and a short description of the program. All methods and classes include headers that describe their functionality and scope and follow commenting best practices. Inline comments are used to indicate code function where appropriate. Variable names are contextual wherever possible.	5
Version Control	GitHub commit history demonstrating regular updates.	4
Total		**80**

EVALUATION

This exam is weighted **25%** of your total mark for this course.