# Application Design, Decisions, and Methods (15 points)

Design and implement C# application ***Decisions***. Include a *custom method* that contains at least one *switch* statement. Include a *custom class*. The custom method may be inside the custom class or in any form – your choice.

This is a very flexible problem! Use your creativity – but please design an application that does something meaningful. Make sure your application is significantly different from any other application you have seen in your textbook, in our classroom, or in a homework assignment. Do not copy code from the Internet for this application.

Complete the CASR excerpt below and implement your solution. When you are done, upload your .zipped answer including your C# application and this sheet to Moodle’s “*Final exam: take-home portion*” link.

*Ideas:*

1. *Custom class represents a person; custom method determines age group (adult, teen, etc.) based on age;*
2. *Custom class represents a set of data collected on a data entry form; custom method determines whether data is valid;*
3. *Custom class represents a calendar event; custom method determines season of the event date.*

## Overview

|  |  |
| --- | --- |
| Application title: | ***Dungeons & Dragons Quick Character Creator*** |
| Purpose: | If you have ever played D&D, then you know about the hours upon hours wasted creating player characters. So much time is wasted in fact, you rarely get to begin the campaign before its time to leave or pack up. This is only the first phase for this project, which I am submitting for the final, but the program still needs about a week worth of programming to be fully operational for the game. This Program makes several decisions based on your choices. |

## Design

|  |  |
| --- | --- |
| Sketch: | C:\Users\The Tacoman.TheTacoman-PC\Desktop\scan0001.jpg |
| Feature list: | There is a random number generator with a strange increment variable to keep the seeds different and not generate the same numbers, this however, is not full-proof and starts generating identical numbers after about the hundredth roll. There is a switch statement to track the modifiers associated with each skill. Since there is very little user input, only the name textboxes have data validation or exception handling. There is also dynamically changing text in the rtbSkills, which is changed when you change character class or race. When You hit Save, you are prompted to if you want to save the file, click ok if you do and the file goes to the xml folder in the file. The file is named for your character, which will also become the retrieval method in the future for when I write a reader for the program. |

## Test *(add or delete rows as needed)*

|  |  |  |
| --- | --- | --- |
| Aspect to be checked | Expected result | Observed result |
| Empty Text boxes | Get message telling me to fill either textbox | works |
| Roll button | Shows stats and corresponding stat modifiers | Works |
| Check all Race Buttons | Changes racial modifiers Accordingly | Works |
| Druid | If druid is selected, Alignment is Neutral automatically | works |
| Paladin | If Paladin is Selected, Alignment is automatically Lawful Good | works |
| Paladin on Load | Same as above only on Load instead of selection | works |
| Class Radio Buttons | Change text according to character Class | works |
| Male/female | Sets string sex as male or female | Works |
| Xml writer | Creates new file with character Name | Works |
| Xml writer | Writes all 25 variables in constructor to new xml document | works |

# Rubric *(standard CIT 195 programming homework rubric scaled to 15 points)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trait | 0 points | 0.5 points | 1 point | 1.5 points | 2 points |
| Application plan (programmer declares what the app should do) | CASR does not describe a solution that answers basic requirements. |  | CASR identifies a few major application requirements and specifications. | CASR identifies all important application requirements and specs. |  |
| User interface sketch (programmer designs user interface before building application) | User interface sketch is missing. | Windows snapshot of application shows result, not plan. |  | Clear user interface sketch represents original design plan. |  |
| Test plan and results (tests check application features; results prove success) | Test plan missing or not applicable. | Test plan and/or results incomplete. |  | Test plan and results are solid and complete. |  |
| Function (application does what it should do) | Application does not fulfill major lesson objectives. | Application fulfills one or a few major lesson objectives. | Application fulfills most major lesson objectives. |  | Application fulfills all major lesson objectives. |
| Results (application produces correct outcomes) | Application produces incorrect results. |  | Application delivers results ineffectively or incompletely. |  | Solution works effectively and produces and displays correct results formatted appropriately. |
| Code quality (the program is written well) | Code is chaotic, poorly structured, or easily broken. | Solution requires data to be clean. Code may be inefficient (ex: brute force approach). | Solution avoids common errors. Code is fairly efficient. | Solution is robust, handles most Exceptions. Code is efficient and well-structured. |  |
| Code aesthetics (programmers can use the app effectively) | Varied spacing and/or inconsistent alignment make code hard to read. | Code is legible but not quite consistent. | Code alignment and white space are consistent and very easy to read. |  |  |
| Naming conventions (programmers can understand the code) | Class member, variable, and constant names do not follow conventions. | Class member, variable, and constant names are confusing or misleading. | Class member, variable names follow conventions, clearly identify purposes. |  |  |
| Design aesthetics (users can use the app effectively) | Control selection, positioning or appearance is haphazard; spelling issues may appear. GUI is difficult to use. | Controls perform correctly but are misaligned, poorly anchored, inconsistent, or counter-intuitive (ex: unexpected tab order). Form may contain minor spelling errors or capitalization inconsistencies. | GUI controls and layout are appropriate to app’s purpose and are intuitive to use; design is pleasant. |  |  |
| Internal documentation (the application is easy to diagnose and support) | Minimal comments do not provide help or are confusing (ex: significant spelling or grammar errors; poorly located). | Comments not matched well with need, insufficient or too basic to provide enough clarity. | Comments effectively explain algorithms and complexities. |  |  |
| Identity (programmers can determine author, date, and purpose) | Author / date / purpose not documented. | Comments in one or some but not all applicable solution files identify author, date, and purpose. | Comments in all forms and custom classes, as applicable, identify author, date, and purpose. |  |  |

**Type your name below** to attest that you did not interact with other students or experts via any means during this exam problem, that you did not borrow solutions or hints from another student or from any prior student(s), and that you did not and will not share your solutions or any hints with other students – current or future:

|  |
| --- |
|  |