# PROG 1400 - Assignment 3

# OOP and GUI

Assignment Value: *12*% of overall course mark.

Due Date: **See due dates designated on the two Assignment 3 dropboxes on Brightspace.**

Late submissions will receive the standard late submission penalty as stated in the course outline.

#### Assignment Instructions:

Use IntelliJ to create a Java Swing UI application. You will also design your solution using a UML class diagram.

#### Submissions:

**This assignment will be submitted in TWO PHASES. The first will be the initial UML class diagram. The second will be the completed program code solution.**

#### You will submit your work for this assignment via GitHub. While you will have frequent commits/pushes of your assignment code to GitHub as your work on it, the instructor needs to know which version to mark and when it was committed. So, when you have completed all assignment work, put a “Ready for Marking” comment on the last code committed into GitHub. Then submit a simple text document to the Brightspace Dropbox that contains the git Commit ID string (e.g. “b180b37”) that identifies that commit. It is this Dropbox submission that will be used to determine late penalties, so make sure to do so prior to the assignment deadline.

#### Once you have committed the code, make sure to visit the repository page on GitHub’s website to verify that the final version has been pushed to GitHub as that is where the instructor will go to get your code.

#### Evaluation:

To insure the greatest chance of success on this assignment, be sure to check the marking rubric contained at the end of this document or in Brightspace. The rubric contains the criteria your instructor will be assessing when marking your assignment.

## Program – RPG Character Creation

### Design and Coding

Prior to coding the application, you should work out a design for the objects in your application. This will be diagrammed using Visio. For example, each Character in the system (either Player or Monster) shares a common attribute of **Name**. Make sure your classes do not duplicate any code by using Java inheritance to share these common attributes and methods. Upload the Visio diagram as a JPEG or PNG along with your source code.

Some guidelines for the UML diagram:

* All classes in your program (besides Main) should be included in the class diagram.
* Your Character class hierarchy must have at least three levels of inheritance (including Character).
* At least two character types (ex. Warrior, Wizard, etc.) and at least two weapon or armour types must be included. A class for Monster should also be included.
* Your solution must include at least one example of aggregation or composition (has-a, instead of is-a). Suggested classes for aggregation: Weapon or Armour.
* All classes, attributes and methods should follow proper UML class diagram standards and conventions.
* Include constructors and over-ridden methods such as toString().
* Line indicators denoting inheritance, association or aggregation should be included wherever required.

### Program Requirements

Use Java and Java Swing to create a simple graphical role-playing game, including an opening splash screen, a simple character creation screen and a battle summary screen.

#### Screen Descriptions

The splash screen will have a title, and image and a button to progress to the creation screen.

The character creation screen will allow the user to enter a character name, choose a character type (ex. Warrior, Wizard, etc.), randomly reroll the character stats, and select a weapon (ex. Sword, Dagger, etc.). It will also have a button to progress to the battle summary screen.

The battle summary screen will display the selected character’s name, class and image, randomly select a monster, then display all stats for both the player and the selected monster.

#### Program Structure

Your program must use inherited classes for the program Frame and each of the three screens (as Panels), as well as for the different characters, character types, weapons and monsters.

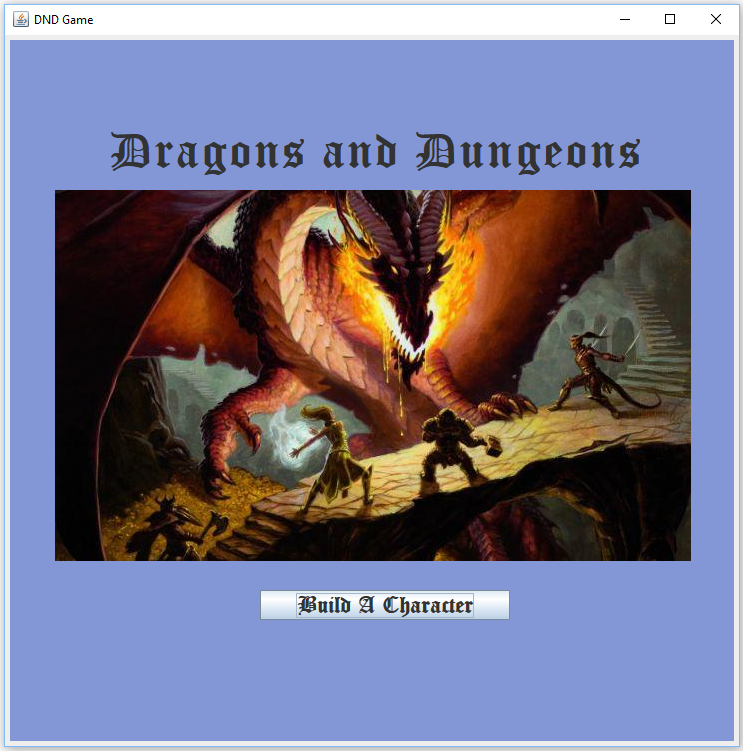
Each of your classes should make proper use of constructors, getter and setter methods, as well as any standard methods you deem necessary.

Your solution should also include at least one example of an abstract class and at least one abstract method. Examples of static properties or methods should also be included.

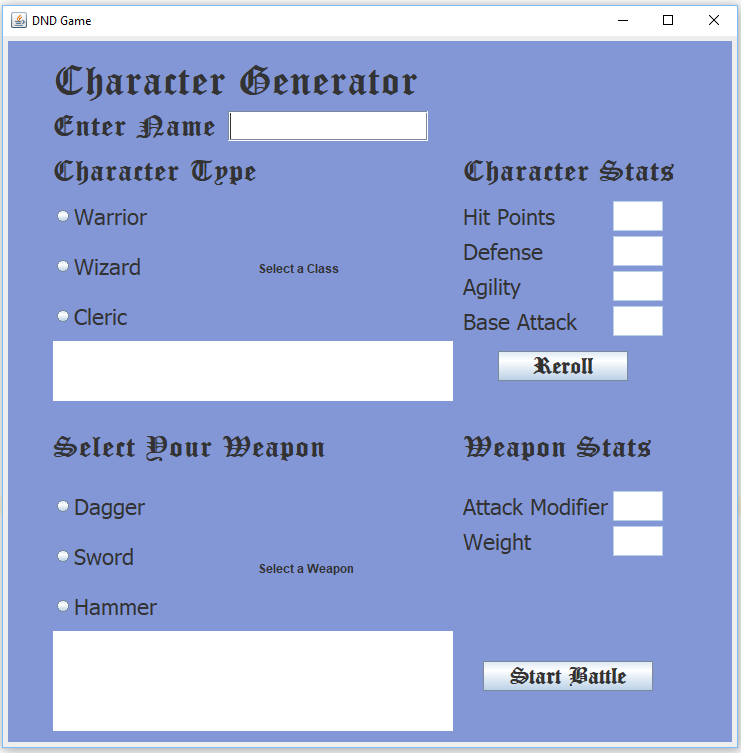
While it is good practice to validate all user-entered data in any program, it is not required in this program.

### Sample Screenshots

###### Splash Screen



###### Character Creation Screen – At Start



###### Character Creation Screen – After Creating a Character



###### Battle Summary Screen



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **RPG Character Creation** | | |  |  |  |  |
| **Criteria** | **Insufficient (0 pts)** | **Needs Improvement (1-2 pts)** | **Sufficient (3-4 pts)** | **Excellent (5 pts)** | **Mark** | **X** |
| **Class Diagram** | Submission missing entirely, or it contains too many issues and/or omissions. | An effort was made, but submission has multiple issues, major issues, or omissions. | A good effort, but submission has a few issues and/or areas for improvement. | Submission demonstrates a strong understanding of class diagram concepts, meets all requirements and fully covers the expected scope of the program. Proper UML notation is used throughout. Displays a high level of analysis, planning and attention to detail. Submission is well-formatted, proofread and presentable. |  | x2 |
| **Program Requirements** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program works as expected, with all required functionality, and runs without errors. |  |  |
| **UI** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | The user interface is properly structured, functional, appropriate and easy to use. Layout is clean and well-formatted. |  |  |
| **Inheritance** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program includes example(s) of inheritance, and implementation demonstrates a strong conceptual understanding. |  | x2 |
| **Aggregation** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program includes example(s) of aggregation, and implementation demonstrates a strong conceptual understanding. |  |  |
| **Abstraction – Classes & Methods** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program includes example(s) of abstracted classes, and implementation demonstrates a strong conceptual understanding. |  |  |
| **Constructors** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program includes a multi-parameter constructor to populate at least some of the properties of a game character object using data taken from the user interface at runtime. Use of class constructors demonstrate a strong conceptual understanding. |  |  |
| **Class Interactivity** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Classes interact using appropriate techniques, through getter/setter/standard methods, using static properties where appropriate. |  |  |
| **Code Efficiency** | Missing entirely, or it contains too many issues and/or omissions. | An effort was made, but has multiple issues, major problems, or omissions. | A good effort, but has a few issues and/or areas requiring improvement | Program demonstrates reasonable code efficiency. (e.g. large segments of code that could be shorter, unnecessary code duplication, code not encapsulated in functions, etc.) |  |  |
| **Comments & Best Coding Practices**  (At least 60% of the reqs must be complete) | Little to no effort was made or contains too many errors / omissions. | A reasonable effort was made, but there are multiple areas for improvement. | A good effort was made, but at least one error or omission exists. | Organizational or explanatory comments are used extensively, most are meaningful and easily understood. A consistent naming convention was used for most of the program and deviated very little. Source code was clean, consistently well-formatted and easy to read. |  |  |
|  |  |  | **Total:** |  | **/60** | |