Dungeons and Dragons Character Creation Helper

OOP Coursework project paper report

Written by: Anastasija Vankova, Ef23-1 group, 2024

## Contents:

1. Introduction
   1. What are the goals and functionality of the program?
   2. How to run and use the program?
2. Body/Analysis
   1. Outlined requirements for the functionality of the program
   2. How does the program meet the requirements?
3. Results
4. Conclusions

## Introduction

As a conclusion of Electronics Engineering program’s Object Oriented programming course, the goal of creating a complex program written in Python was chosen. The aim was to write a functional code on the decided topic that would cover most of the discussed OOP techniques as well as to write a report paper to explain the program functionality and structure. For this project, the topic was “Dungeons and Dragons” (“DnD”) tabletop roleplaying game character creation spreadsheet helper.

For this program to work as intended, it should be able to help the user with the creation of their fictional character creation process. This process usually consists of developing a digital character sheet, which has different attributes, such as character’s name, class, backstory, statistics (powers), possessed items and abilities. The user should then be able to not only save, but also edit and view created characters. Then, a separate file should be used for storing all data of created characters (in our case it will be a .json type file).

#### How to run and use the program?

For easier access to program’s functionality, a separate README file was created, where all features of this command based helper were listed. The features are:

\* Adding multiple characters

\* Editing characters attributes: name, class, history, stats, abilities, inventory

\* Listing characters

\* Saving characters to JSON file

\* Loading characters from JSON file

To use the program, the user shall execute the python main.py file.

## Body/Analysis

#### Outlined requirements for the functionality of the program

Some of the crucial requirements for this program by the lecturer were:

* The program should strive to implement OOP pillars: Polymorphism, Abstraction, Inheritance and Encapsulation principles among the others.
* There should be used at least 2 Python design patters, such as Factory method, Decorators, Builders, Adapter, Singleton, or others.
* There should be testings done using unit test features.
* The program should be able to read and write to a separate file (like .txt, .json)

#### The structure and core logic of the created program

The DnD helper consists of three functioning .py files: main.py, manager.py and character.py. Also, it has a test\_main.py file for performing tests, as well as a characters.json dictionary file for storing all accumulated character related data. Lets take a closer look at each of the files.

#### Main.py

The script allows the user to create, edit, list, and delete DND characters. It uses the Builder pattern to create characters with different classes (e.g., Warrior and Wizard). The CharacterManager handles the saving, loading, and updating of characters. The script provides a menu-driven interface for the user to interact with the character management functionalities.

*Data handling:*

* Character Creation: The script uses builders (WarriorBuilder and WizardBuilder) and a director (Director) to construct character objects.
* Storage: Characters are managed by the CharacterManager, which presumably handles reading from and writing to a file (although the file operations are not shown in this snippet).
* User Input: The script heavily relies on user input for creating and editing characters.

*Functionality:*

The methods of this file are typically responsible with proving the user with options for interacting with the program and creating/editing their character. They also start the initial processes of interacting with created characters. For example, in this file there are methods responsible for starting character deletion, creation, editing, or for starting to edit main characteristics of said characters. Some of the frequent functions utilized here is ‘match’ command, where the program mathes the user input with the index of supported character edit commands:

A screen shot of a computer code

Description automatically generated

‘match’ command use example

One of the important methods that provide smooth program functionality is described here. It is also used in multiple cases when the user needs to proceed with some kind of important action:

A screenshot of a computer program

Description automatically generated

Ask confirmation prompt code

*The main loop*

The is an endless While loop implemented in the main.py file. The script enters an infinite loop where it waits for user input and performs actions based on the input command. The available commands are:

* h: Display help.
* n: Create a new character.
* l: List all characters.
* e: Edit an existing character.
* q: Quit the application.

#### Manager.py

*Classes and Methods:*

**Director Class**

The Director class is responsible for constructing a Character object using a specified builder. It follows the Builder design pattern, which separates the construction of a complex object from its representation.

It has an attribute “Builder”, which holds the builder object that will be used to construct the Character.

A screen shot of a computer program

Description automatically generated

Figure 1. The Builder method implementation code snippet

The methods of this Builder are:

* set\_builder(self, builder): Sets the builder object.
* build\_character(self): Constructs a Character object using the builder. It sequentially calls the builder's methods to set the character's category, items, abilities, max health, and stats.

**CharacterManager Class**

The CharacterManager class has a Singleton design pattern responsible for managing the storage and retrieval of character data. It handles the file operations for saving, loading, updating, and deleting characters.

It has such attributes:

* \_instance: Holds the singleton instance of the class.
* character\_storage\_file\_path: The file path where character data is stored.

A screen shot of a computer program

Description automatically generated A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

There are quite a lot of methods in this class, most of them are for managing (saving, editing, deleting characters or editing their attributes.

The Singleton pattern requires these methods to function properly here:

* \_\_new\_\_(cls): Ensures only one instance of CharacterManager is created (singleton pattern).
* \_\_init\_\_(self): Initializes the file path for character storage.

Here is an overview with comments of a more interesting method of this class: A method responsible for saving characters. It also shows how we utilize json.dump and writing to .json file.

A screen shot of a computer code

Description automatically generated

In summary, the manager.py file defines two key classes, Director and CharacterManager, which together handle the creation and management of DND characters. The Director uses the Builder pattern to construct characters, while the CharacterManager provides methods for storing and retrieving character data from a JSON file. This structure supports the main functionalities in main.py, allowing users to create, edit, list, and delete characters through a command-line interface.

#### character.py

The character.py file defines the Character class and its builders for creating specific types of characters (*WarriorBuilder* and *WizardBuilder*). Here's a detailed breakdown of the file, its main logic, data handling, and the functionality of each class and method:

**“Character” class**

This class has several attributes important for any DnD character**.**

* name: The name of the character.
* backstory: The backstory of the character.
* category: The class/category of the character (e.g., Warrior, Wizard).
* items: The items the character possesses.
* abilities: The abilities the character has.
* max\_health: The maximum health points of the character.
* stats: The stats of the character (e.g., strength, dexterity, intelligence, wisdom).

**CharacterBuilder Class**

This is an abstract base class for creating character builders. It defines methods that should be implemented by concrete builders to provide the character's attributes.

**WarriorBuilder Class**

This is a concrete builder class that implements the CharacterBuilder interface to create a Warrior character with predefined attributes. It has a lot of ‘get’ functions.

A screen shot of a computer program

Description automatically generated

WarriorBuilder class

**WizardBuilder Class**

This is a concrete builder class that implements the CharacterBuilder interface to create a Wizard character with predefined attributes. It functions in a similar way to a WarriorBuilder, but has a different set of pre-defined attributes for the characters of the wizard class.

A screen shot of a computer program

Description automatically generated

WizardBuilder class

Then, in main.py, these classes are used to create and manage DND characters.

#### Tets\_main.py

This file was created specifically for running tests with Unit test. At the moment, it only tests a small fraction of the code, the WizardBuilder class of the character.py file.

This code requires importing

#### All in all

*Character Creation:*

* The Director class sets a builder (either WarriorBuilder or WizardBuilder) and constructs a Character object by calling the builder's methods to set the character's attributes.
* The newly created character's name and backstory are set through user input.
* The character's information is displayed, and the user is asked to confirm if they want to save the character.

*Character Management:*

* The CharacterManager class interacts with Character objects to save, load, and update character data in a JSON file.

### Results

* A properly functioning DnD character creating software with command-based user interface was created.
* The ways for allowing the user to create, view, edit and delete characters and their attributes were provided for the user.
* Different OOP principles and design patterns (Builder, Singleton) were used.
* Some additional functions such as attributes editing are still to be implemented, however, the outline and logic for this functionality is already present.

### Results

This coursework has provided a much needed cause for the actual practical usage of all skills and knowledge the OOP lectures were giving the students. In this case, developing a Dungeons and Dragons character creating helper from scratch required learing and implementing OOP techniques, running tests with Unit Tests, handing program interaction with a separate .json file as well as fixing a lot of errors along the way.