Our implementation of *Simple LSRPG* is based on three layers: business, persistence and presentation.

In the presentation, we have all the messages we need to print and are static meaning that they do not depend on any variable. For example, the welcome message is “Welcome to Simple LSRPG.\n” is always the same. Another examples could be the error message when the adventure name entered is not unique, and the printed message when data is well loaded “Data was successfully loaded.\n”. Those read-only attributes together with printing non-static information functions and request information with the user are placed in the **Console Ui Manager**. In other words, everything displayed in the console that is connected with the user. In addition, there’s **two enumerations** since we have detected two menus. The main one with options like CHARACTER_CREATION or LIST_CHARACTERS, and an encounter menu with ADD/REMOVE MONSTER and CONTINUE.

Continuing with the business layer, we have considered two main **entities** that were already declared in the statement (**character** and **monster**). We also have added **adventure** with its name, number of encounters and array of encounters. That’s the reason why there’s a one way aggregation relationship. One adventure has between 1 and 4 encounters. To be able to store **encounters**, we have created another class. One encounter has multiple monsters. Finally, a **party** is played in an adventure and the party itself has multiple characters and contains important information to control the characters hit points and initiative of the monsters which are inside adventures. To control all the logic for example checking name requirements is inside the **Business Facade Implementation** and declared in its interface.

Ending with the persistence layer, there’s three Data Access Objects, one for each json file (monster, character and adventure). In every DAO we determine the “filter by” functions where a specific class is ordered by a concrete attribute. Last but not least, there’s **PersistenceException** class to solve errors when loading data from different files.

How are the three layers connected?

The UI Controller in the presentation layer uses a business facade interface (connected in turn with the functions implementation, BusinessFacadeImpl). This last class where the logic of the program is executed connects different managers which access different DAOs from their Json Class.