Exercise 1 - Design patterns

Design Pattern 1: Observer

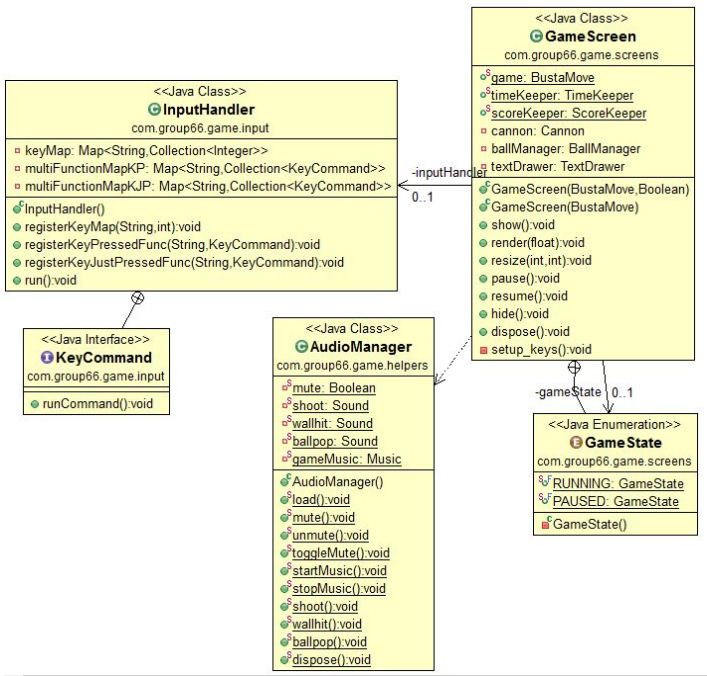
1. Write a natural language description of why and how the pattern is implemented in your code

- How: When a key is pressed, an input handler checks the pressed key and notifies the observers subscribed to that key that the state has changed. The notification comes in the form of calling methods attached to specific objects i.e. when “P” is pressed, a GameState method toggles the state of the game (running/paused), and a AudioManager method toggles the audio of the game accordingly.
- Why: Because that’s the best way to capture keyboard input. The only other way is for each objects to always keep checking the condition of its corresponding key in the keyboard; which is kind of stupid, causes a lot of lag, and very time/resource consuming. Using this design pattern is one of the most conventional ways to keep track of keyboard input.

2. Make a class diagram of how the pattern is structured statically in your code

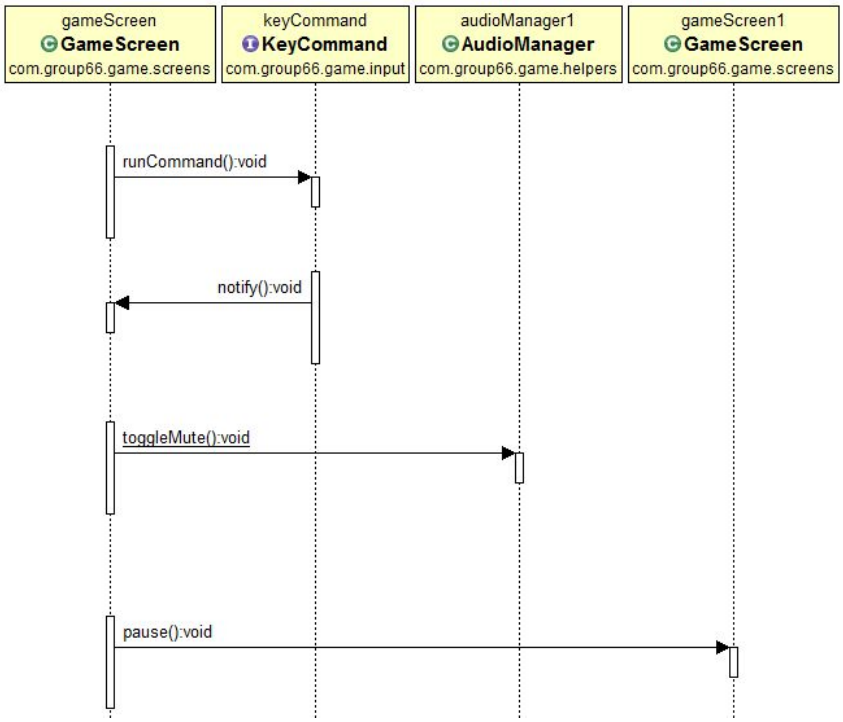
Taking the example of the case mentioned above where pressing “P” execute game toggling and audio toggling by notifying the corresponding class methods; the class diagram would be the following figure.

The Input handler class is not a part of the subject/observer group in the pattern, but it have been viewed for its relation with the GameClass screen as it sets up the key commands and manages the listener. This will also help clear its role in the sequence diagram.



3. Make a sequence diagram of how the pattern works dynamically in your code

This sequence diagram makes it clear that after registering the possible key presses, whenever that key is pressed, the AudioManager and GameScreen objects are notified. The notification alone is enough for the objects to act accordingly, as they know their current state and what's required is a toggle operation.



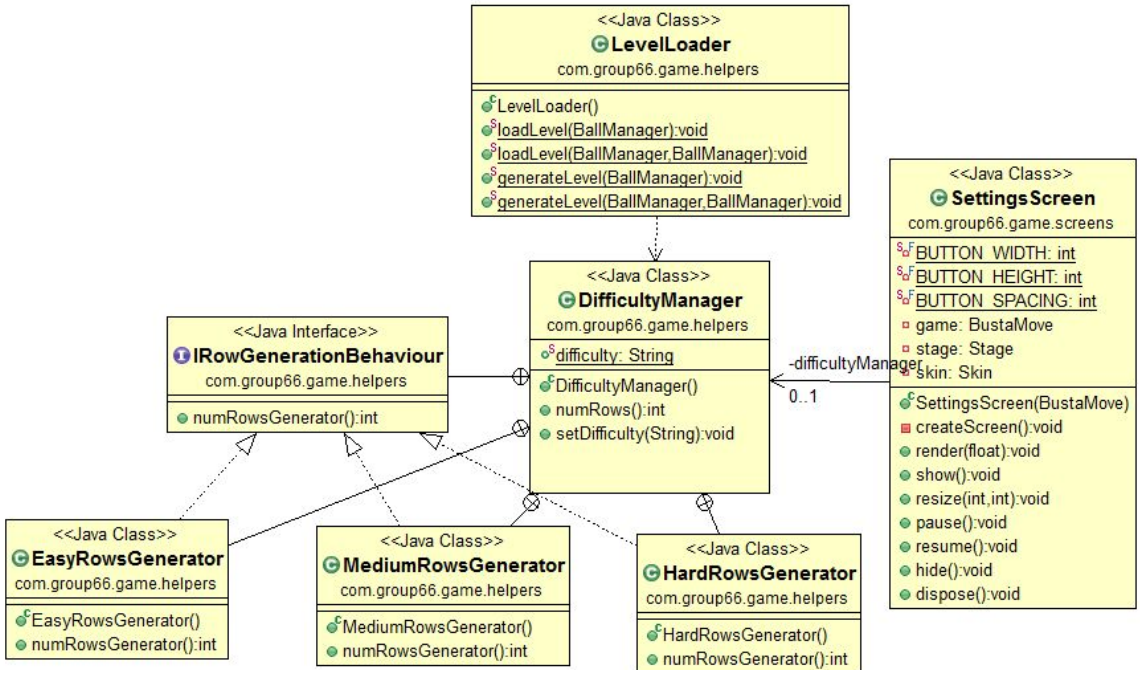
Design Pattern 2: Strategy

1. Write a natural language description of why and how the pattern is implemented in your code

- How: When the player chooses to play a random game, the game is generated based on the difficulty he selected in the settings menu. The difficulty is largely based on the number of rows generated when a game is started for now, but other attributes to each difficulty will be improved later as a part of the progress being made with the game.
- Why: The strategy pattern by definition defines a family of algorithms, encapsulates them, and makes them interchangeable. The design pattern usage here focuses on encapsulating the random game generation algorithms so each can be used interchangeably with the other based on player preferences. For exactly our case right now -which is only changing how the number of rows in the auto-generated game is defined-, normal code writing methods like case switches may work; but for our plan to actively improve how game difficulty generation works, it is best to follow this design pattern.

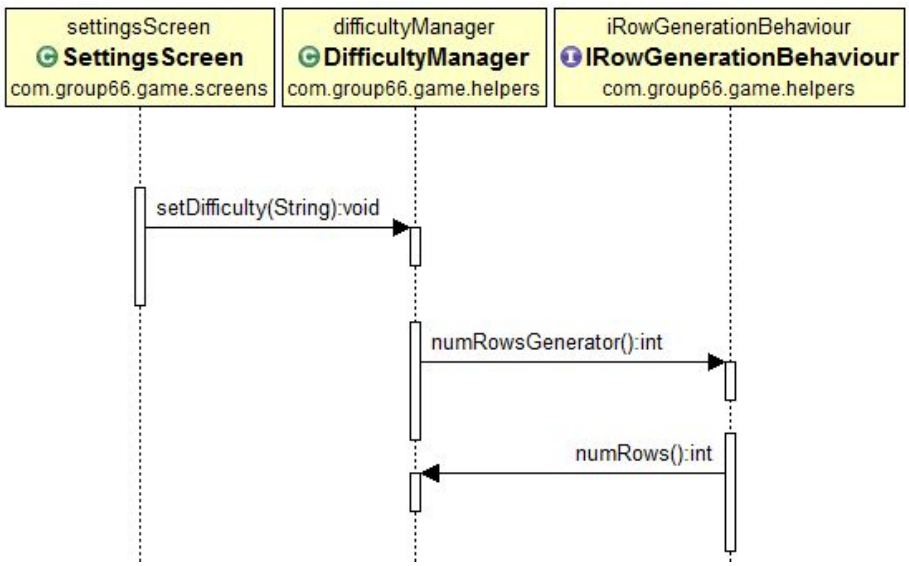
2. Make a class diagram of how the pattern is structured statically in your code

The DifficultyManager class contains an interface named IRowGenerationBehaviour. Three instances of this interface generate the easy, medium, and hard difficulties. These are the algorithms encapsulated, and can be changed on runtime based on the player preference.



3. Make a sequence diagram of how the pattern works dynamically in your code

The SettingsScreen includes the option that gives the player the opportunity to select a difficulty. Upon selection using the DifficultyManager object in that screen, different algorithms of generation (easy, medium, hard) are selected based on the player selection. Algorithms are encapsulated in a class that implements the IRowGenerationBehaviour interface.



Exercise 2 - Your wish is my command

1. Each project and group is heading to a specific direction and there are specific things that should be implemented and improved. Who knows it better than your TA?

In this exercise, you have to talk to your TA (during the group meeting on Monday) and (s)he will give you what to do next to your game.

After you receive the task from your TA, write a requirements document, which will be evaluated in the same way as for the requirements document of the initial version. Afterwards you must implement the requirements.

Special ball related requirements can be found within the Your wish is my command directory.

2. During the analysis and design phases of this extension use responsibility driven design and UML (push to the repository the single PDF file including all the documents produced)

Special ball related design can be found within the Your wish is my command directory.

Exercise 3 - 20-time

1. Google asks its employees to spend 20% of their time at Google to a project that their job description does not cover. As a result of the 20% Project at Google, we now have Gmail, AdSense, and Google News, among the others.
This is your occasion to have similar freedom. You can decide what to do next to your game: It can be an extension/improvement from any perspective, such as improved code quality, or novel features.
Define your own requirements and get them approved by your teaching assistant. Afterwards you must implement the requirements.

Splitscreen related design can be found within the 20-time folder.

2. During the analysis and design phases of this extension use responsibility driven design and UML (push to the repository the single PDF file including all the documents produced)

Splitscreen related design can be found within the 20-time folder.