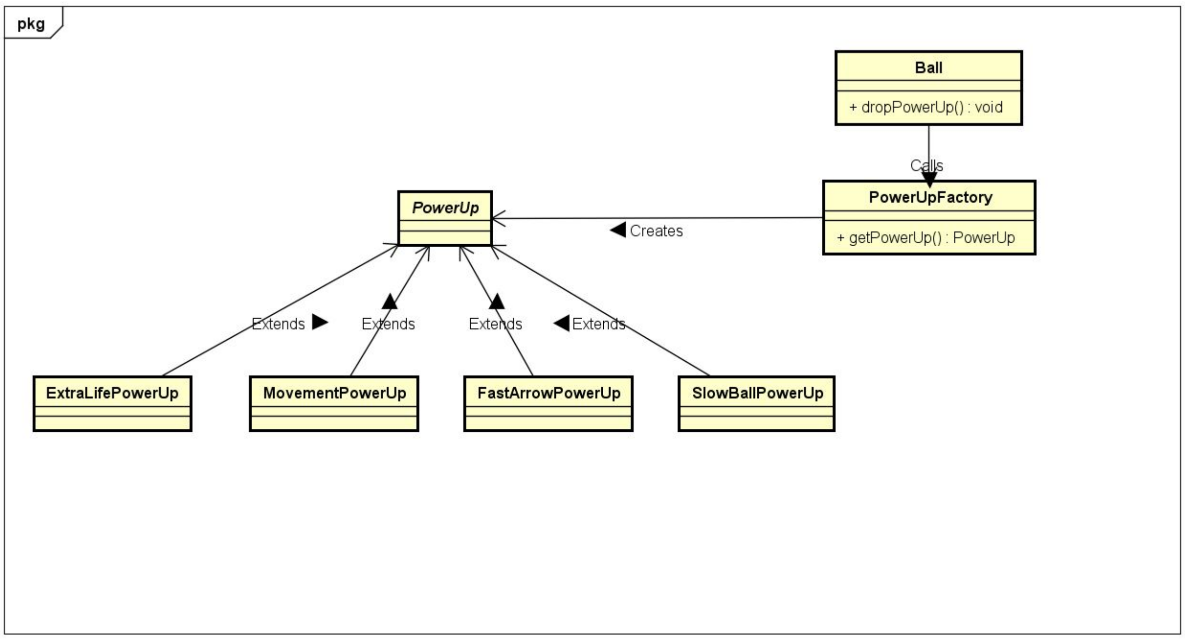
Exercise 2 - Design patterns

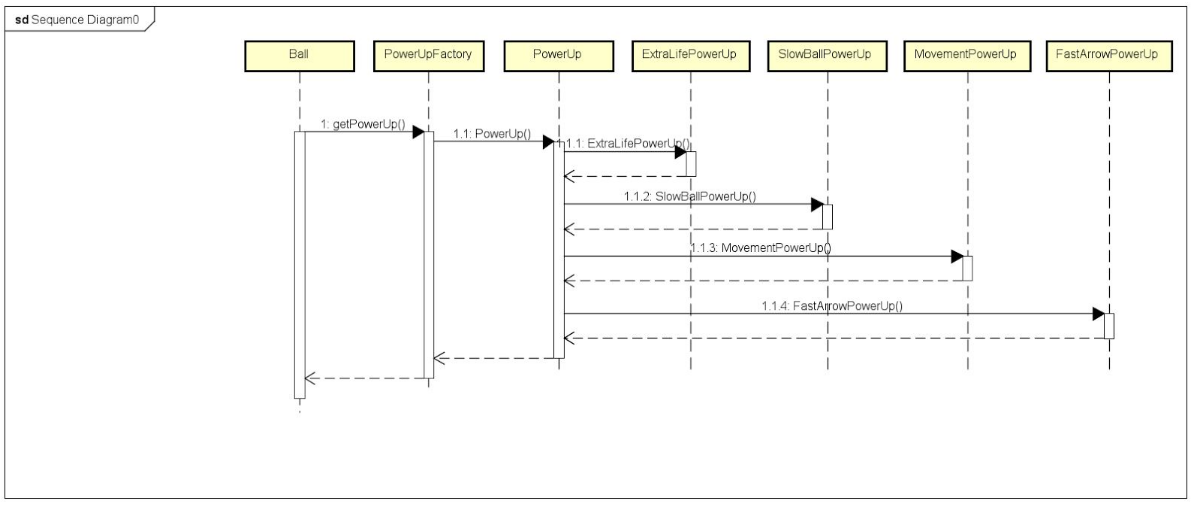
Factory Pattern

We decided to implement the factory pattern for this project. We implemented a factory for the powerups. As there are several implementations and types of powerups, implementing a powerup factory was an easy way to make the process of creating a powerup more organized.
All powerups implementations are abstractions from the abstract class PowerUp. The class PowerUpFactory chooses randomly which powerup type is created and return this to the caller. The ball class uses the PowerUpFactory to spawn powerups.

Class Diagram



Sequence Diagram



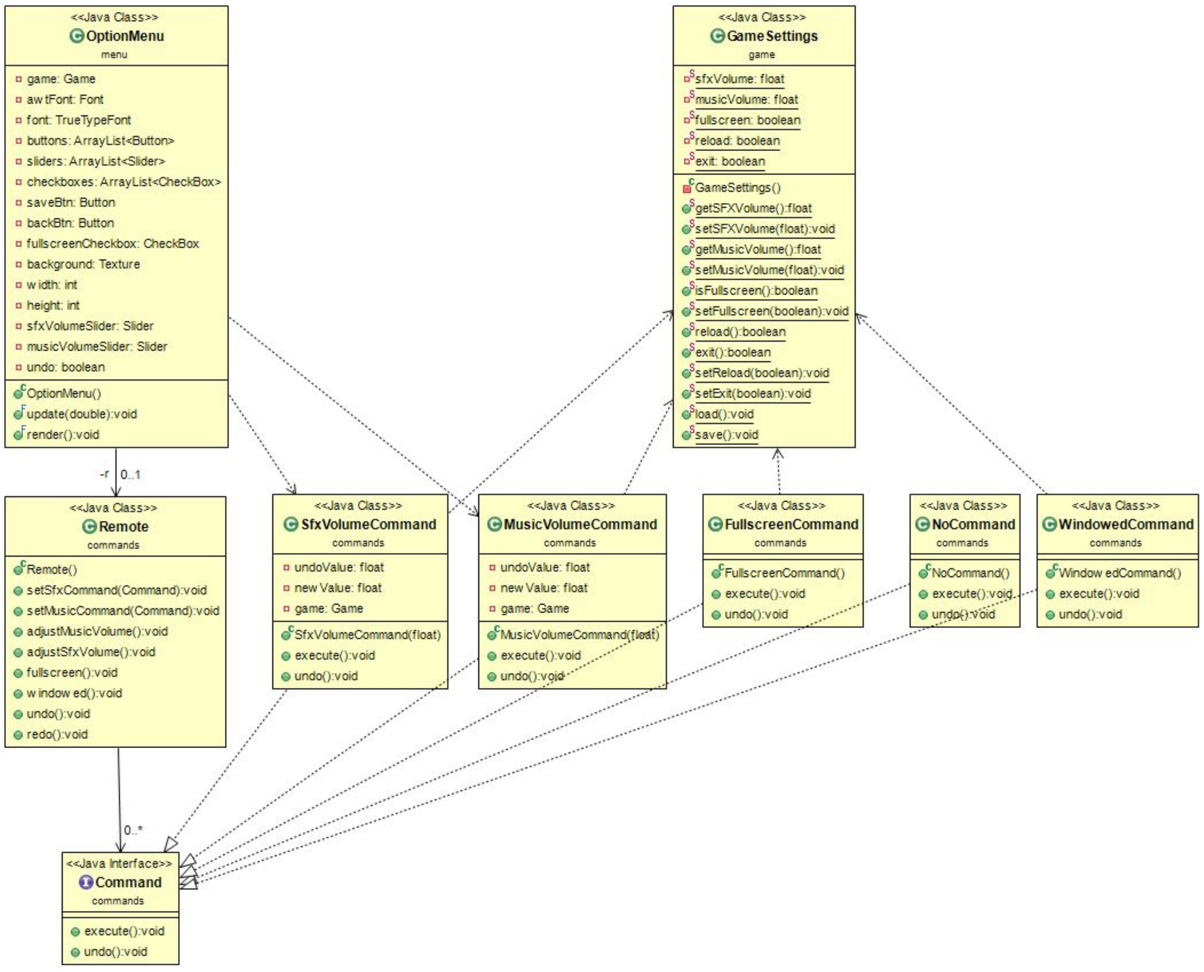
Command Pattern

To make the options menu more user friendly we decided to implement the command pattern. We use implemented Remote class to execute different types of commands. These commands are all individual Classes that implement the interface Command to ensure they have the basic functionality to execute and undo the command.

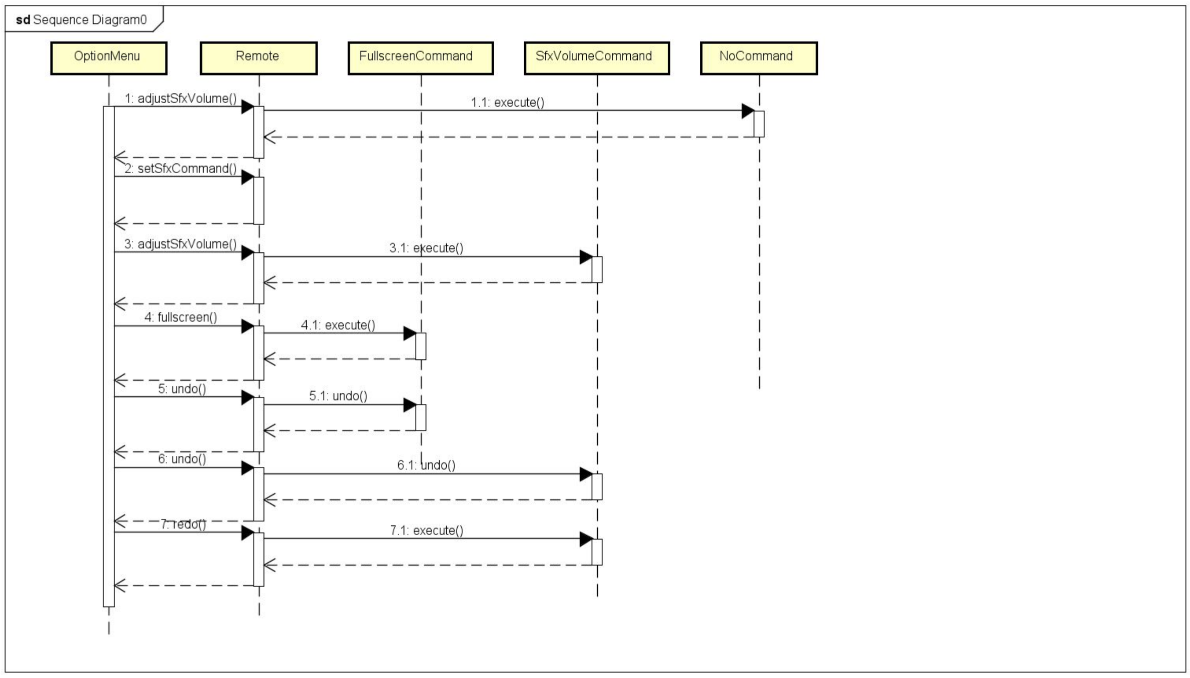
The remote uses two stacks (the history stack and the future stack) to keep track of the Commands that have been executed. To undo the last command the remote only needs to pop the last Command from the history stack and call the undo method. To redo the next command the remote only needs to pop the next Command from the future stack and call the execute method.

By implementing the Command pattern in the OptionsMenu we can allow the user to undo previous changes by pressing Ctrl+Z and redo changes by pressing Ctrl+Y.

Class Diagram



Sequence Diagram


sd Sequence Diagram0
OptionMenu
Remote
FullscreenCommand
SfxVolumeCommand
NoCommand
1: adjustSfxVolume()
1.1: execute()
2: setSfxCommand()
3: adjustSfxVolume()
3.1: execute()
4: fullscreen()
4.1: execute()
5: undo()
5.1: undo()
6: undo()
6.1: undo()
7: redo()
7.1: execute()