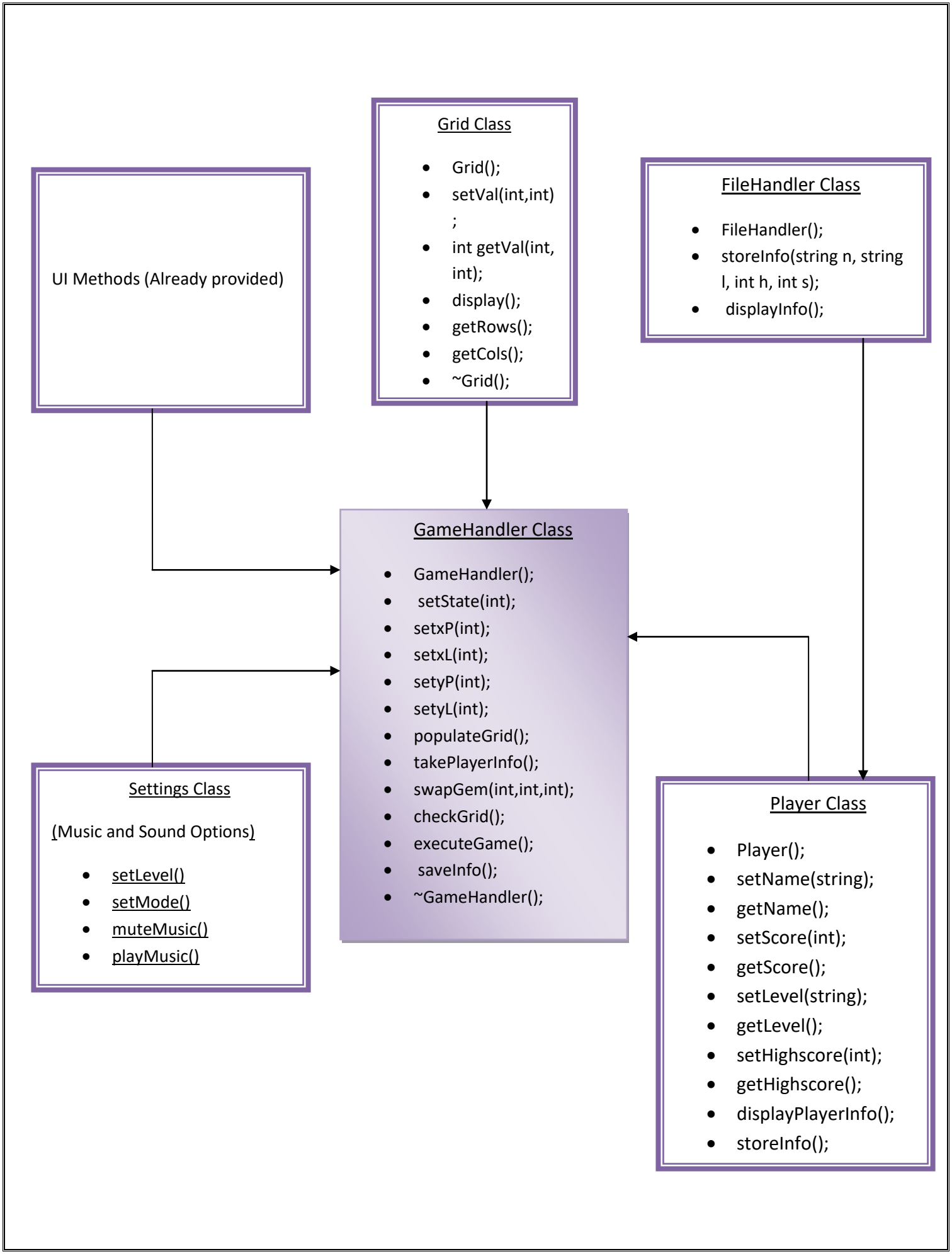
UI Methods (Already provided)
Grid Class
Grid();
setVal(int,int);
int getVal(int, int);
display();
getRows();
getCols();
~Grid();
FileHandler Class
FileHandler();
storeInfo(string n, string l, int h, int s);
displayInfo();
GameHandler Class
GameHandler();
setState(int);
setxP(int);
setxL(int);
setyP(int);
setyL(int);
populateGrid();
takePlayerInfo();
swapGem(int,int,int);
checkGrid();
executeGame();
saveInfo();
~GameHandler();
Settings Class
(Music and Sound Options)
setLevel()
setMode()
muteMusic()
playMusic()
Player Class
Player();
setName(string);
getName();
setScore(int);
getScore();
setLevel(string);
getLevel();
setHighscore(int);
getHighscore();
displayPlayerInfo();
storeInfo();

The Class Diagram

OOP Final Project

The primary class which is responsible for all the methods of all the objects, whose object will be a global variable in main.cpp file. It will start by creating an object of the Grid class which will make a dynamic 2D array and store its address in 2D pointer in Grid class.

The populateGrid function will randomly fill the array with gems (different shapes) which will be made by the 2D integer array filled with random numbers. Depending on the random number stored in the array, a gem will be made.

The next step will be the input of the player information and it will be stored in the Player object which will further be stored in a file using the FileHandler class.

The player will then enter the specifications of the game such as level, mode, highest score etc and will be storing the information in the same player object as well as appending it in the same file too.

All these specifications will be passed to the GameHandler object which is responsible for the actual execution of the game. This class will also be using the given functions in the util files. It will show buttons like ‘Hint’ and ‘Setting’.

Lastly, at the end of every iteration, the entire player’s current information (score, level, etc) will be updated in the player object as well as in the same file by the File Handler class.

Every class will interact via the GameHandler class using the principles of composition to ensure the concept of encapsulation as every class has its own functionality with no repetition. All the problems are broken down in smallest possible manner to deal with them efficiently.

The concept of abstraction is also ensured using access modifiers as no class interacts with the main() function as well as between themselves directly but are used with the GameHandler class acting as a manager between the classes.