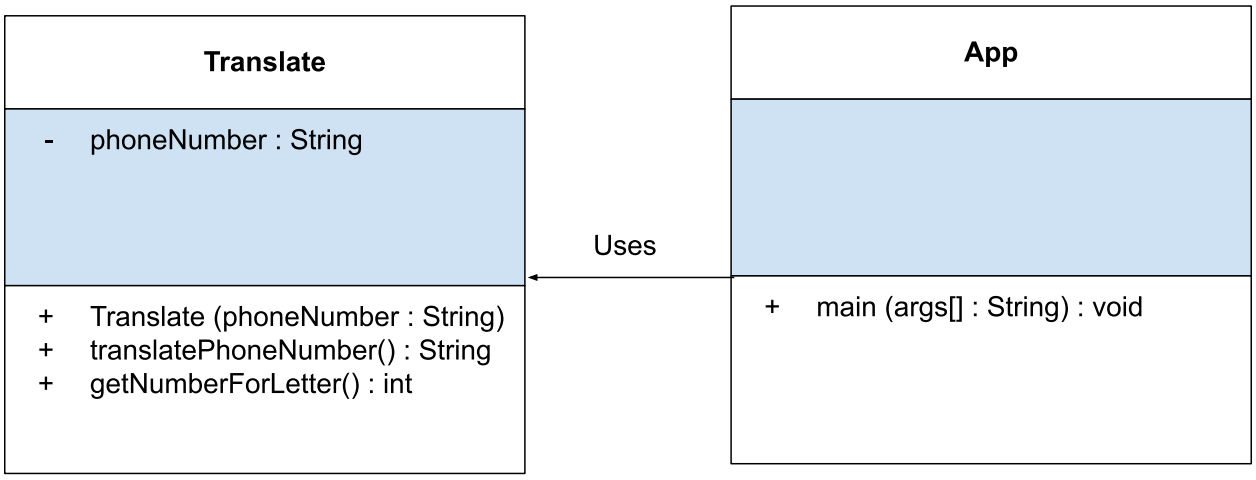
Assignment03-Q3:



Input Conversion

Main Class to Execute

App Class:
Handles user input and interacts with the Translate class, no attributes

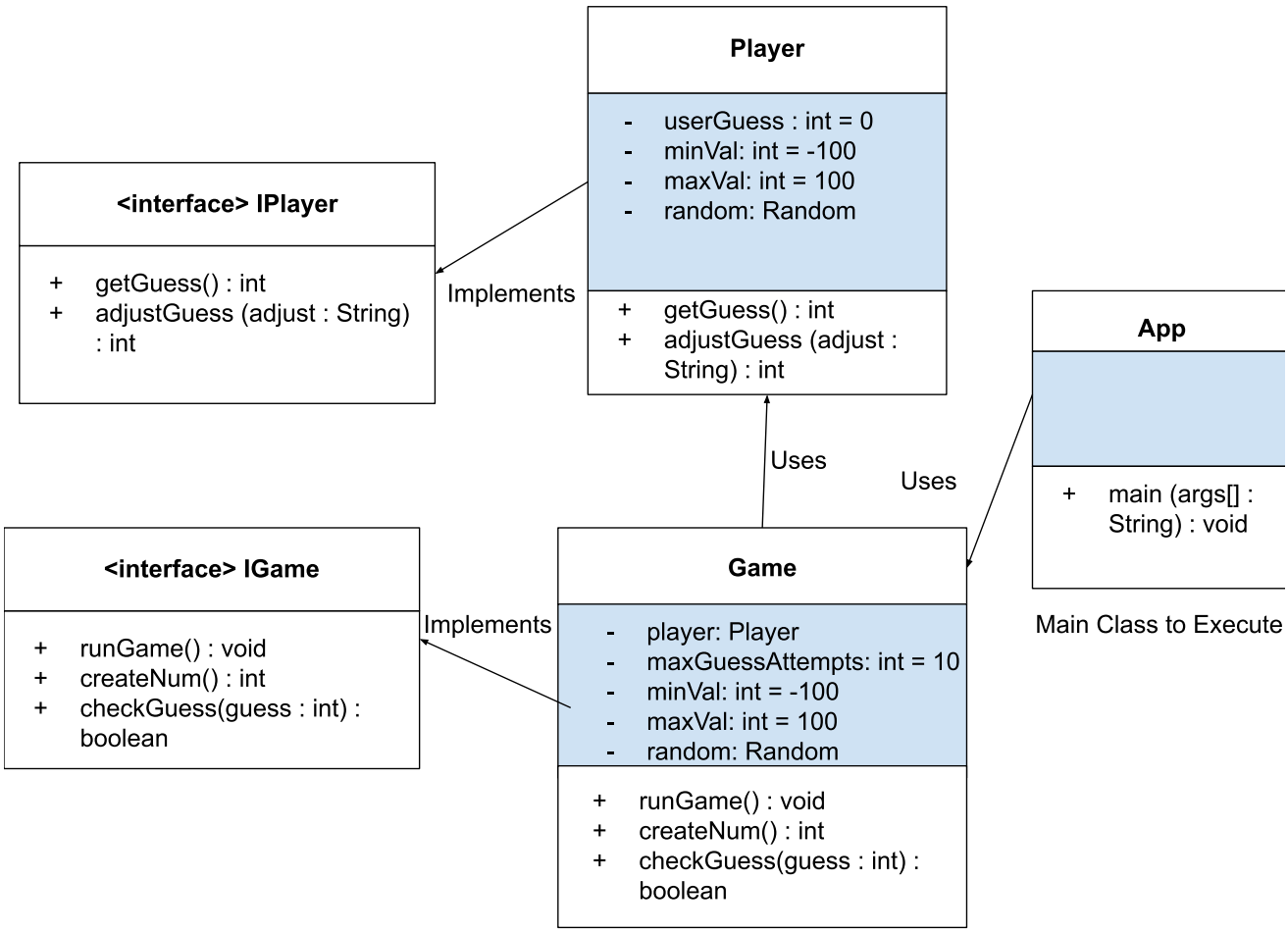
main(args[]: String) : Takes a phone number through Scanner and sends to Translate, which returns the updated phone number, which is then printed

Translate Class:
Holds a private string phoneNumber so it is not accessible from outside. The constructor Translate then takes input from App and assigns it to phoneNumber.

getNumberForLetter() is a private helper method that is used within translatePhoneNumber(). It returns an int based on the letter that is inputted, aka translates the letter into a number.

translatePhoneNumber() then uses the letters taken from getNumberForLetter() and builds a new translated phone number string that is returned.

Assignment03-Q5:



App Class:
Purpose: Executes the program

main(args[]: String) : Initializes an instance of Game and calls the runGame method to run the guessing game without user input

IPlayer Interface:
Defines the methods that the Player class will implement (i.e the contract)

getGuess(): gets the current guess of the computer player
adjustGuess(): private method to adjust the guess based on whether the number is “lower” or “higher” (which is the string input)

IGame Interface:

Defines the methods that the Game class will implement (i.e the contract)
runGame(): Contains the logic for running the game (the loop)
createNum(): Creates the random number to be guessed
checkGuess(): Checks whether the guess is equal to the created number, and returns true or false based off of that

Player Class:
Implements the methods from the IPlayer interface, simulates the computer player
userGuess is a private int used for storing the guess the computer makes, default is 0
private minVal and private maxVal define the range of numbers that can be generated (-100 to 100)
random is an instance of the random class that will be imported by Game to generate the random number, and used to adjust the guess if it is not correct

The methods are the same as in the IPlayer interface

Game Class:
private maxGuessAttempts holds the max guess attempts (10)
Player holds an instance of the player Class, so it can get the guesses.
minVal and private maxVal define the range of numbers that can be generated (-100 to 100).
Here it is used in random to create the initial number that needs to be guessed
random is an instance of the random class, and uses these minVal and maxVal to generate the number to guess

The methods are the same as in the IGame interface

Assignment03-Q6:

Encapsulation

The Player category has attributes such as userGuess. These are private and only reachable through public methods like getGuess(), and it uses private methods like adjustGuess().

The Game class wraps up the game's reasoning, including making numbers and checking guesses. It makes sure that its attributes (maxGuessAttempts) are not openly reachable or changeable outside of the class.

This allows for easier maintenance as modifications inside a class's working approach can be performed without impacting other segments of the code. It also limits access to vital data sections, avoiding unintended alterations and making the code less bug prone.

Information Hiding

Properties such as userGuess, maxGuessAttempts and the random number used in Game are private. They can be changed only through specific public methods that have been clearly defined. For example, the Game class has private functions such as createNum() and checkGuess(). These allow for specific actions but do not show what happens inside. By keeping the method details hidden, it simplifies the code and decouples information so it isn’t used in multiple places, and prevents misuse of these methods leading to easier to read code.

Interface

Interfaces such as IPlayer and IGame are made so Player and Game both stick to a specific agreement. IPlayer declares methods like getGuess() and adjustGuess() and IGame can declare methods like runGame() and createNum() for their respective classes to implement.

These interfaces give you permission to create varied versions of the same function for future implementation, like making different versions of the player class. This also contributes to separating the connection between classes. The Game class doesn't require knowledge about Player's specific execution, only that it follows the IPlayer interface rules.