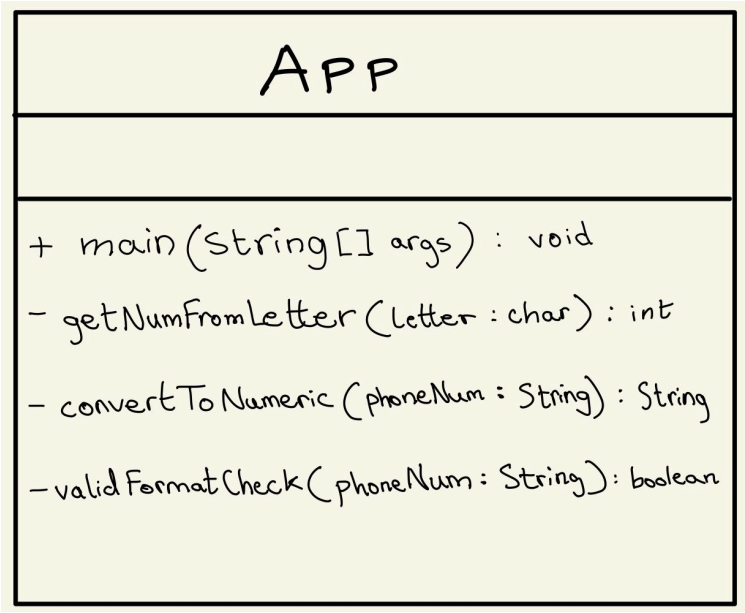
Assignment03-Q3:



Class:
The “App” class is responsible for handling the overall functionality of converting the alphanumeric telephone number that the user inputs into a numeric telephone number.

Attributes:
There are no class-level attributes in the ‘App’ class in the UML diagram

Methods:
(+ main(String[] args) : void): The main method prompts the user to input a 10-character telephone number in the format XXX-XXX-XXXX. After the program receives the user’s input, it checks whether the format of the input is valid using the validFormatCheck() method. If the format is valid, the phone number is converted to its numeric equivalent using the convertToNumeric() method, and the result is printed to the user. If the input is invalid, an error message is printed and the program terminates.

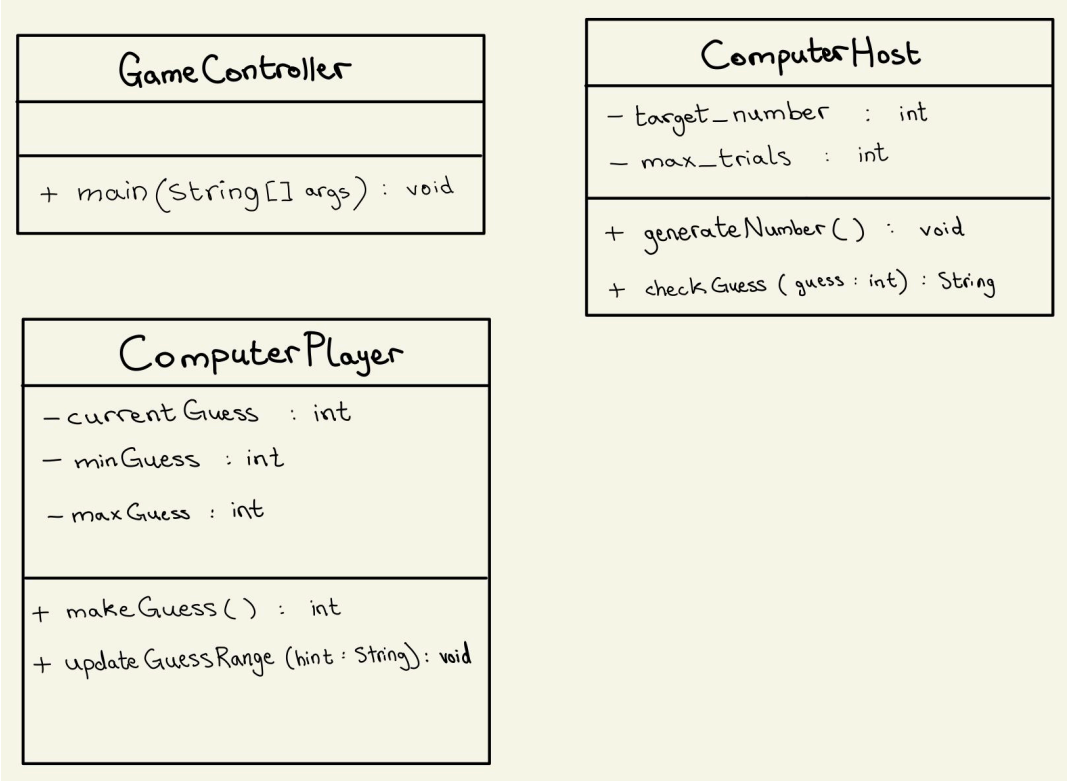
(- getNumFromLetter(char letter) : int): This method of ‘integer’ type takes a single character as a parameter and returns the corresponding numeric value of the letter based on how it is mapped on a telephone keypad. The corresponding numeric value is returned; if the input is not a letter, it returns -1, indicating the input is invalid.

(- convertToNumeric(String phoneNum) : String): This method of ‘String’ type converts a phone number (the method’s parameter) that may contain alphabetical characters into its numeric equivalent. It iterates through each character of the user’s 10-character telephone number, checking if it is a letter using the getNumFromLetter() method and converting it to its corresponding number. Once the conversion is made, then it is appended to the result. If it is not

a letter, the character is appended without any conversions or changes made. Once the entire is done being iterated through, the final result is returned as a string.

(- validFormatCheck(String phoneNum) : boolean): This method of 'boolean' type validates whether the phone number inputted from the user is in the correct format (XXX-XXX-XXXX). It basically checks that the phone number is exactly 12 characters in length and ensures that there are hyphens at the 3rd and 7th indexes. It returns true if the format is valid and returns false if it is invalid.

Assignment03-Q5:



Class:
The "GameController" class is responsible for managing the flow of the guessing game. It coordinates the interaction between the computer host and the computer player.

Attributes:
No class-level attributes in "GameController" class.

Methods:
(+ main(String[] args) : void): The public main method initializes the game by creating instances of the ComputerHost and ComputerPlayer classes. This enables the computer to generate a target number and control the guessing process throughout the game.

Class:
The “ComputerHost” class is responsible for generating the target number and checking the guesses made by the player throughout the game.

Attributes:
(- target_number : int): This attribute stores the randomly generated number that the player needs to guess. Using the principle of encapsulation, the variable is private to ensure that it cannot be accessed or modified directly outside of the “ComputerHost” class.

(- max_trials : int): This variable is a constant that defines the maximum number of attempts the player has to guess the target number. It is marked as static final to indicate that it is a constant value that is shared across all instances.

Methods:
(+ generateNumber() : void): This method generates a random value between -100 and 100 and assigns it to target_number. The method is public so it can be called in the main method of the GameController class to start the game. The method also encapsulates the logic for number generation.

(+ checkGuess(int guess) : String): This method compares the player's guess with the target number and prints out hints based on the comparison, whether the guess is higher, lower, or correct.

Class:
The “ComputerPlayer” class is responsible for acting as a player in the game, making guesses based on the hints provided by the host.

Attributes:
(- currentGuess : int): This variable stores the current guess made by the computer player. It’s private to prevent external modification and protects the integrity of the guessing game.

(- minGuess : int): This variable keeps track of the lower bound of the player’s guessing range, which is initially set to the minimum possible value of -100. It’s private to prevent external modification and encapsulates the state.

(- maxGuess : int): This variable keeps track of the upper bound of the player’s guessing range, which is initially set to the maximum possible value of 100. It’s also private to prevent external modification and encapsulates the state.

Methods:
(+ makeGuess() : int): This public method implements the logic for making a guess. The player is requested to make a guess by the game within the range.

(+ updateGuessRange(String hint) : void): This method updates the guessing range based on the feedback from the host and informs the player the adjustments they need to make to guess the target number. Specifically, if the guess is too high, it updates maxGuess. If the guess is too low, it updates minGuess.

Assignment03-Q6:

Encapsulation Principle:

Application: This principle is applied by using private attributes in the ComputerHost and ComputerPlayer classes, ensuring that private states like target_number and currentGuess are protected from external modification or accessibility. Public methods such as generateNumber() and makeGuess(), allow for controlled access to these private attributes, maintaining a clear boundary between class responsibilities. The GameController class further manages the flow of the game without exposing any inner workings.

Benefits: The benefits of this principle includes enhanced data integrity as it prevents unauthorized access to private states. Moreover, it has improved maintainability and modularity since the changes to one class do not affect the others in any way that would harm the execution of the program, further making the code easier to understand.

Information Hiding Principle:

Application: This principle is applied by restricting access to the internal workings of the “ComputerHost” and “ComputerPlayer” classes through the utilization of private attributes and methods. For instance, target_number in ComputerHost and currentGuess in ComputerPlayer are attributes that are kept private, in which they allow other classes to interact with them only through public methods like generateNumber() and makeGuess(). Ultimately, this logic is hidden from the outside, limiting the exposure of implementation details and only the public methods are exposed for interaction.

Benefits: The benefits of this principle includes a simplified interface and increased security, where by exposing only key methods, users of the class are protected from changes in its implementation that could break their code. Moreover, they can interact with it without needing to understand its internal workings, making the code user-friendly and easy to maintain.

Interface Principle:

Application: This principle is applied by defining public methods in each class, allowing interaction without exposing any internal details. For example, the GameController class calls the generateNumber() method from the ComputerHost class and the makeGuess()

method from the ComputerPlayer class, facilitating clear communication between classes.

Benefits: The benefits of this principle includes how the interface defines a clear contract for how classes communicate with one another. This essentially promotes improved modularity, easier maintenance, and flexibility as changes to a class do not affect other parts that use the same public methods within the code.