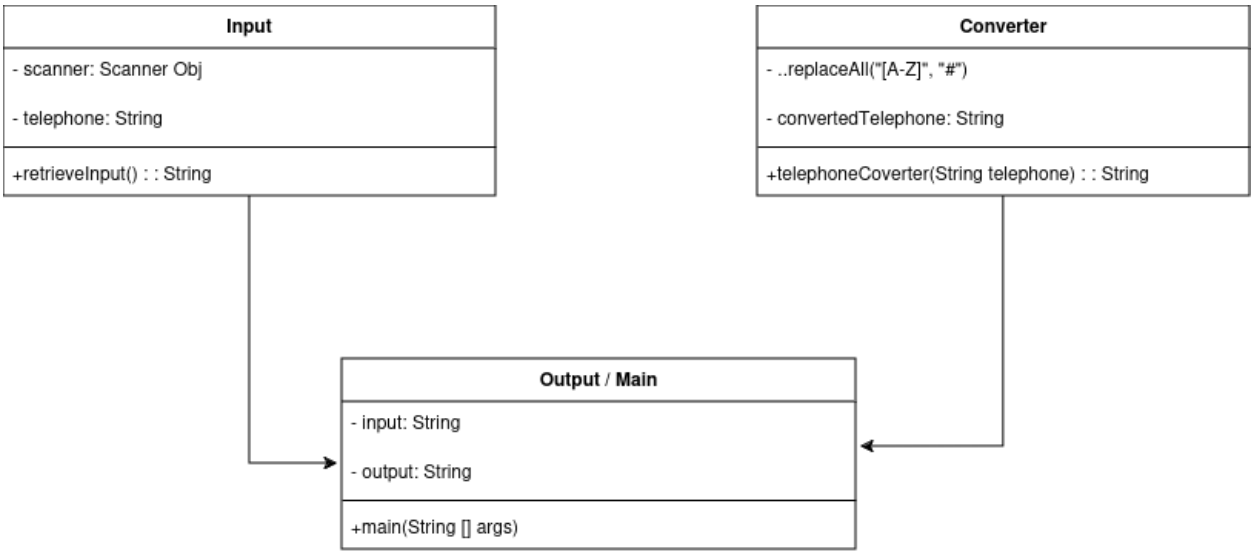
Assignment03-text

Question 3:



Full detail of Methods and Variables:

1. **Method:** `retrieveInput()`
    - **Purpose**: This method is used to gather input from the user via the console. It prompts the user to enter a 10-character telephone number in the format `XXX-XXX-XXX`. It also ensures that the input is converted to uppercase.
    - **Return Type**: `String` – the telephone number provided by the user, converted to uppercase.
    - **Variable Used**:
        - `scanner`: An instance of the `Scanner` class that reads input from the console.
2. **Method:** `telephoneConverter(telephone: String)`
    - **Purpose**: This method takes a string representing a telephone number as input and replaces any letter groups (e.g., `ABC`) in the string with the

corresponding digits (like `2`). It essentially converts an alphanumeric telephone number into a numerical one.

- **Return Type**: `String` – the converted telephone number with letters replaced by numbers.
- **Variable Used**:
  - `convertedTelephone`: A string variable that holds the modified telephone number, starting as the original `telephone` input and then being progressively updated as each letter group is replaced with a corresponding digit.

3. **Method:** `main(args: String[])`
   - **Purpose**: The entry point of the program, which coordinates the execution of the other methods. It retrieves the telephone number from the user, converts it using `telephoneConverter()`, and prints both the original and the converted telephone numbers to the console.
   - **Return Type**: `void`
   - **Variables Used**:
     - `input`: Stores the telephone number provided by the user (after calling `retrieveInput()`).
     - `output`: Stores the numerical telephone number (after calling `telephoneConverter()`).

Question 5:

Class: `GameHost`

- **Purpose**: This class is responsible for setting up and managing the game. It generates a target number and evaluates the guesses made by the player program. The game ends when the player either guesses the number correctly or uses up all attempts.
- **Attributes**:
  - `target`: The number the player needs to guess, randomly generated within a range.

- `maxAttempts`: The maximum number of guesses allowed (e.g., 10).
- `rangeMin`: The lower bound of the range within which the target number falls (e.g., -100).
- `rangeMax`: The upper bound of the range (e.g., 100).

- **Methods**:
  - `startGame()`: This method initializes the game, generates the target number, and sets the game environment.
  - `evaluateGuess(guess: int): String`: Compares the player's guess with the target number and returns feedback such as "too high", "too low", or "correct".
  - `isGameOver()`: Checks if the game has ended, either due to a correct guess or after reaching the maximum number of attempts.

2. Class: `PlayerProgram`

- **Purpose**: The player program acts as the one trying to guess the target number.
  It generates guesses based on feedback from the host (e.g., narrowing down the range after each guess).
- **Attributes**:
  - `currentGuess`: The most recent guess made by the program.
  - `attemptsMade`: Keeps track of how many guesses the player program has attempted.
  - `minRange`: The minimum range for the current guessing logic, adjusted based on feedback (e.g., when the guess is "too high").
  - `maxRange`: The maximum range for the guessing logic.
- **Methods**:
  - `makeGuess()`: This method generates a new guess by picking a random number within the current range.
  - `receiveFeedback(feedback: String)`: Based on the feedback from the `GameHost`, this method adjusts the guessing range (like if the guess was "too high",

the program will adjust `maxRange` to be below the guess).

3. Class: `Game`

- **Purpose**: This class brings together both the `GameHost` and `PlayerProgram`. It coordinates the game by letting the player program make guesses and passing those guesses to the game host for evaluation. It handles the game loop and ensures the game progresses until the game ends.
- **Attributes**:
  - `host`: An instance of the `GameHost` class.
  - `player`: An instance of the `PlayerProgram` class.
- **Methods**:
  - `play()`: This method manages the interaction between the host and the player program. It allows the player to make a guess, sends the guess to the host for evaluation, and processes the feedback. The game continues until the host signals that the game is over.

High-level Overview of Game Flow:

1. **Game Setup (`GameHost.startGame()`)**:
   - The game host generates a random target number within a defined range (-100 to 100).
   - The host is ready to evaluate guesses.
2. **Player Makes a Guess (`PlayerProgram.makeGuess()`)**:
   - The player program generates a guess, starting with the full range (-100 to 100).
   - The player sends the guess to the host for evaluation.
3. **Host Evaluates Guess (`GameHost.evaluateGuess()`)**:
   - The host checks if the guess matches the target number. If not, the host provides feedback (too high or too low).
4. **Player Adjusts Based on Feedback (`PlayerProgram.receiveFeedback()`)**:

- The player program adjusts its guessing range based on feedback. If the guess was too high, the player narrows the guessing range to below the guessed number. If the guess was too low, it narrows the guessing range to above the guessed number.

5. **Repeat**: This process continues until either the player program correctly guesses the number or the maximum number of attempts is reached.
6. **End of Game**:
    - If the player guesses correctly, the game ends with success.
    - If the player runs out of attempts, the game ends with failure, revealing the correct number.

Question 6:

1. Encapsulation Principle

- The `GameHost` class encapsulates the target number, attempt limits, and range boundaries.
- The `PlayerProgram` class encapsulates the current guess, attempts made, and guessing range.
- Each class contains methods that operate on its own data, maintaining a clear separation of concerns.

2. Information Hiding Principle

Information hiding is applied to restrict access to internal details of each class:

- The `GameHost` class hides the target number, preventing direct access from outside the class.
- The `PlayerProgram` class conceals its guessing strategy and internal state.
- Methods like `evaluateGuess()` in `GameHost` and `makeGuess()` in `PlayerProgram` provide controlled access to internal functionality without exposing implementation details.

3. Interface Principle

- The `evaluateGuess()` method in `GameHost` provides a clear interface for receiving guesses and returning feedback.
- The `receiveFeedback()` method in `PlayerProgram` offers an interface for the player to process game host responses.
- The `Game` class acts as a facade, providing a simplified interface (`play()` method) that coordinates interactions between `GameHost` and `PlayerProgram`.