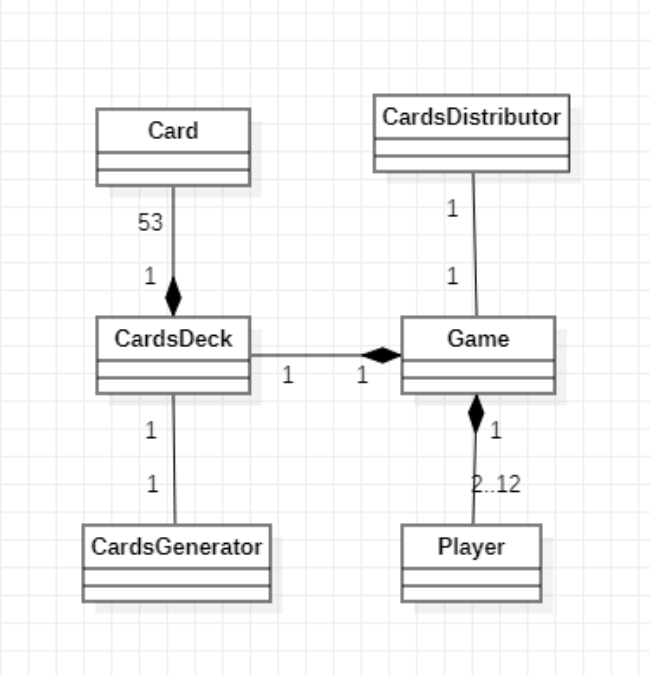
Atypon

Old maid cards game assignment report

Rayyan Al-Hourani

A-Object-Oriented design



1. Identify the Classes:
    a. Game: Represents the game and contains players and the card deck.
    b. Player: Represents individual players participating in the game.
    c. Card Deck: Represents a collection of cards.
    d. Card: Represents an individual card in the deck.
    e. Cards Generator: Represents a class responsible for generating a set of cards.
    f. CardsDistributor: Represents a class responsible for distributing cards between players concurrently.
2. Define the Relationships:
    a. Game contains multiple instances of the Player class.
    b. Game contains an instance of the Card Deck class.
    c. Card Deck contains multiple instances of the Card class.
    d. Cards Generator interacts with the Card Deck to generate the cards.
    e. CardsDistributor interacts with the Game and Card Deck classes to distribute cards.

3. Define Relationships:
    a. The Game class should have a composition relationship with the Player class, as it contains multiple instances of Player objects.
    b. The Game class should have a composition relationship with the CardDeck class, as it contains a single instance of the CardDeck.
    c. The CardDeck class should have a composition relationship with the Card class, as it contains multiple instances of Card objects.
    d. The Cards Generator class should have an association relationship with the CardDeck class, as it interacts with the CardDeck to generate and manage the cards.
    e. The CardsDistributor class should have an association relationship with the Game class, as it interacts with the Game to distribute cards.

B-thread synchronization mechanisms

After analyzing the old main cards game, I found that we can utilize multithreading in several parts of the program. The sequence of the program is as follows:

(I use multithreading in the red points and the part when the thread start work)

1. Create an instance of the Game class.
2. Instantiate the CardGenerator class and generate each type of cards separately, then add them to the original card deck.
3. Ask the user to enter the number of players (between 2 and 12).
4. Create each player and put them in an array of players for role regulation.
5. Distribute the cards in a circular (Round-Robin) fashion using the CardsDistributor class, utilizing multithreading for efficient distribution.
6. Assign each player to a separate thread and start all threads.

   //////// Here the thread starts its work //////////////
7. When the threads start, all players begin discarding their cards using the discardCard function.
8. After finishing the discarding, each player waits for a few seconds to ensure that all players have discarded their cards.

9. While players have not discarded their cards and the number of players in the game is more than one, the threads will remain active.
10. While it is not a player's turn in the game, the player waits until their turn arrives.
11. The turn depends on the players' array in the Game class. The game walks through them sequentially to determine whose turn it is.
12. Then, it checks if the player ID matches with the current thread's player ID to determine if it is their turn.
13. If it's not their turn, the player waits until other players notify all threads to wake up and check if their turn has come yet. If not, they go back to sleep.
14. If it's the player's turn, they first choose a random number of cards to take from the next player.
15. They draw a card from the next player as per the chosen index.
16. The player then tries to discard their cards if possible.
17. The game checks if the next player doesn't have any cards left because the current player took their last card. In that case, the next player goes out of the game, and the game removes them from the players array to ensure they won't get a turn again.
18. After that, the game moves the turn to the next player and notifies all threads to wake up.
19. The game continues walking through the players' array until there is only one player left.

//////// Here the thread finished its work or back to wait state //////////////

20. After the game finishes, it prints the player who lost the game.

C-Clean code

1. Use meaningful variable names and avoid single-letter variables.
2. Each function should have a name that explains what it does.
3. Add comments inside functions to provide additional explanations where necessary.
4. Organize classes into packages to maintain code organization.
5. Format the code to enhance readability.
6. Avoid using negative conditions, such as double negatives, in code logic to enhance clarity.
7. Aim to keep methods with no more than 2 parameters to avoid excessive complexity and maintain simplicity.