Deliverable 2

Muharrem Kaya
Derya Kaya
Mustafa Soroush

Prof: Amandeep Sidhu
March 30, 2020

1. Project Background and Description

This game plays with 52 cards. These 52 cards are divided into four suits, such as clubs (♣), diamonds (♦), hearts (♥) and spades (♠). Also, each card has a number or a character with one of the suits. For example, A for an Ace, K is for a King, Q is for Queen, and J is for Jack, like ♠A, ♠2, and ♠3.

2. Use Case Diagram

If the game plays four players or less than four, seven cards are distributed to each player of the game. If the game plays more than five players, five cards are distributed to each player of the game. Remaining cards put any place where each player can access them. A player who writes his/her name first can be the first player. The first player wants a card from the next player to complete his/her suit that has the same number. For example, Can you give me a card that contains number 8? If the next player has the card that is wanted by the first player, the next player gives to the first player (how many cards with the same number he/she has, he/she has to give them to the first player) and the next player has a right to want another card from the next player.

On the other hand, if the next player does not have the card, he/she says "GO FISH," so the first player takes a card from the remaining cards. The line passes the next player. The player opens cards on the table if the player completes a series, such as ♣7, ♦7, ♥7, and ♠7. After the player begins the cards on the table, he/she takes another card from the remaining cards. The game continuous in this way until the remaining cards finish. When the game finishes, the player who has the most completed series becomes a winner.

3. Class Diagram

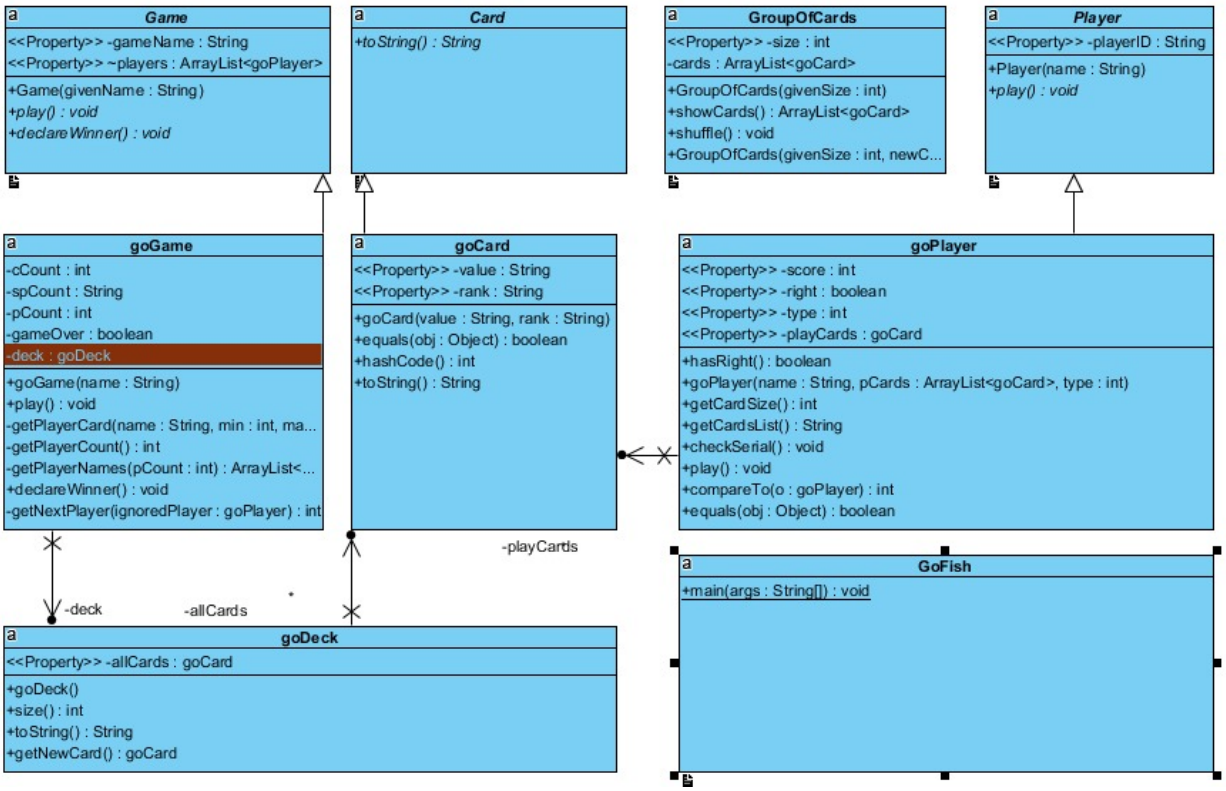

Figure 1

We have four classes at the beginning, and we have added five more classes to our application.

- Encapsulation:
  - Game.java
  - GroupOfCards.java
  - GoCard.java
- Delegation:
  - Game.java
  - GroupOfCards.java
  - Player.java
  - GoCard.java
- Cohesion:
  - Game.java
  - GroupOfCards.java
  - Player.java
- Coupling:
  - Card.java
  - Game.java
  - GoFish
  - GroupOfCards
  - Player.java
  - GoCard.java
  - GoDeck
  - GoGame
  - GoPlayer

  At the above, we have added both tight and loose coupling classes, and further explanation is provided in each class section.
- Inheritance:
  - GoGame inherited from the Game class.
  - GoCard inherited from Card class.
  - GoPlyaer inherited from Player class.

    These are the three inheritance that we have in the Diagram.
- Aggregation:
  - GoDeck to GoCard
  - GoPlayer to GoCard
- Flexibility/Maintainability:
  - Card.java
  - Player.java
  - GoCard.java

  *At above, the name of the class that has these types of functions in them is provided, and a detailed explanation can be found below.

Card.java

This class is an abstract base class in which the children classes will extend it.
This class has a default constructor, an abstract method, and the children class will implement the abstract method of this class.
This class can be maintained and reused by other classes.
This class has loose coupling, and its method is not dependent on other classes.

Game.java

In this class, we can define the name of our game.
There is also an ArrayList of players name in this class which it is of type, GoPlayer.
In this class, we can see getters and setters of the fields which indicate using encapsulation.
There is also an abstract method that will be called and implemented by the children class, and this method will declare the winner of the game.
Also, another abstract method called play. This method will be called and implemented by the children class. To implement the function of the game and define how the game must be played.
In this class, we can see delegation, encapsulation, and cohesion.
In this class, the ArrayList is of the type of GoPlayer, so we have tight coupling in here.

GoFish.java

GoFish is our view class, where the user can start the game and interact with our application.
This is the only class that the main class exists in it so we can communicate with the user. This class will make a call to the goGame class by creating an object of this class and passing the name of the game.
This class uses the object of GoGame is not loosely coupled.

GroupOfCard.java

GroupOfCard is a class for defining the size of our cards. We used ArrayList of the type GoCard in here to implement this class.
There is a method to get the size and assign the size, which can also be done by the constructor.
Encapsulation of data fields, delegation, and cohesion can be seen in this class.

Player.java

This class is design to assign players names.

This class has delegation, cohesion since the only purpose of this class is to assign the player names.

This class can be maintained and reused by other classes.

This class is loose coupling, and its method is not dependent on other classes.

GoCard.java

This class has two fields along with their getter and setter, which indicate encapsulation.

This class will extend our Card class so we can see inheritance here.

The constructor of this class will take and set value and the rank of the cards. We assign the name of the classes in here we don't create the cards; we create them the cards in GoDeck class, and this consider delegation.

This class can be used later again to assign the value and rank of our cards.

This class extends the Card class, so it is not loose coupled.

GoDeck.java

In the MVC pattern for the classes that are related to cards, this is our controller class.

This implements the functions that will help us to present our cards. This class will do this by using the GoCard class and the attributes that exist in that class.

This class will create cards and pass them to GoCard class for the names to be assigned.

These class methods and ArrayLists are of her type GoCard, this classes in not loose coupled.

GoPlayer.java

This class extends player class, so we have an inheritance.

In this class, check scores, count card size, remove a player after they lose.

It can be getter and setters have been used, and with different data fields that indicate our data encapsulation.

Also, these class methods use GoCard class in its implementation, so it is not loosely coupled.

GoGame.java

GoGame is our main controller class. Most of our logic is in this class.

In this class, we can see inheritance. This class will extend the Game class.

GoFish class will pass the name of the game to the constructor of this class, and the constructor will pass this name to its superclass, which is game.

After using various classes to assign player names, create cards, create cards size, assigning card name, now, We have brought the functions of various classes that we have implemented to here, to do the final implementation and make the logic of how our code should interact with the user.

The play method of this class is the main method in here, which will run the game.

This method will be called by goFish class to run the game.

The getPlayerCard method will make the function to ask the user to choose a card and process it, and if the value was not a valid loop for a valid value.

The getPlayerCount will help to define the max and min of player and prompt for names.

The getNextPlayer help to choose the next player in the game.

This class methods mostly use other classes methods and functions to do the implementation, so it is not loosely coupled.