

COP3538
Project 2 – Go Fish!

Submission Requirements

- Submit your project folder via the FileUploader tool provided on the website
 - Follow the project submission guidelines for the class

Input File Requirements

- Cards.Input.txt (The card data file to load into the program)
- CardData.java (Class file that provides important functions to the Card class)
- Queueable.java (Interface file that dictates the methods to use in the PriorityQueue and Queue classes)
- Stackable.java (Interface file that dictates the methods to use in the Stack class)

Output File Requirements

- A screenshot image of the complete program output

Design Specification Requirements

Note: Refer to the sample output in the **Example Output** section below.

1. Create 8 classes (project, Card, Deck, Game, Player, PriorityQueue, Queue, Stack)
 - A. The project class (the name will be whatever the project is called)
 1. This is the class that contains the **public static void main(String[] args)** method
 2. This class should only do three things
 - a. Display programmer name(s) and project title on separate lines, followed by a blank line
 - b. Create an instance of the Game class
 - c. Call the Game.execute method
 - B. The **Game** class (the Driver class)
 1. This class controls the operation of the program starting at the **execute** method
 2. This class must perform the following operations
 - a. Prepare the game
 1. Create a card collection (the Deck class) with a size of 52
 2. Create 4 players (the Player class) each with a size of 15

Hint: Store the Player objects in an array.

Hint: Pass a reference of the Game object to each Player object (use the “this” keyword).

 - a. Give each player a name like “Player x” where x is a number between 1 and 4
 - b. Assign each player a number between 0 and 3
 - b. Read the card data from the Cards.Input.txt input file and store it in the card collection
 - c. Deal 7 cards to each player
 - Hint:** Give each player one card at a time.
 - d. Play the game (see the **Game Rules** section for additional details on game play)
 - e. Display the results of the game (see the **Example Output** section)
 3. The class provides the following **public** methods
 - a. execute() – Performs the operations of the program (step 1.B.2.b – 1.B.2.e)
 - b. getDeck() – Returns a reference to the **Deck** object
 - c. getPlayer(int player) – Returns a reference to the specified player
 4. The class provides the following **private** methods
 - a. dealCards(int numCards) – Deals the specified number of cards to each player
 - b. display() – Displays the results of the game
 - c. playGame() – Plays the game according the rules (see the **Game Rules** section)
 - d. readFile() – Reads the data from the input file into the program

- C. The **Deck** class (a Collection class)
- Note 1: The Deck class **MUST** inherit the **Stack** class.
- Note 2: The Deck class requires a single constructor, but no other methods are required.
- The class constructor should accept a size parameter, which is passed to the **Stack** class
Hint: Pass the size value using the **super** keyword.
- D. The **Player** class (a different Collection class)
- Note: The Player class **MUST** inherit the **PriorityQueue** class.
- Initialize the object
 - The class constructor should accept a size parameter, which is passed to the **PriorityQueue** class
Hint: Pass the size value using the **super** keyword.
 Note: The priority queue will serve as the player's Hand (the cards he/she is holding).
 - Create an instance of the **Queue** class.
 Note: The queue will serve as the player's Discard Pile.
 - The class provides the following **public** methods
 - display()** – Displays the player's name, as well as the cards in his/her hand and discard pile
 - doYouHaveAny(int card)** – Determines if the player is holding the specified type of card in his/her hand. If so, the card (only one) is returned; otherwise, **null** is returned.
 - getPosition()** – Returns the player's position (a number between 0 and 3)
 - playHand()** – Performs the operations required to play a round (see the **Game Rules** section)
 - The class provides the following **private** methods
 - discard(Card card)** – Stores the specified card into the player's discard pile
 - displayHand()** – Displays the cards in the player's hand
 - displayPile()** – Displays the cards in the player's discard pile
 - drawCard()** – Draws a card from the deck of cards (the **Deck** object)
Hint: Use the **Game** object reference provided in step 1.B.2.a.2.
 - getNextPlayer(Player currentPlayer)** – Determines the player in the next higher position
 - matchCards()** – Searches the player's hand for matching pairs (2 cards) of cards, and if a match is found, the pair of cards are discarded to the discard pile
- E. The **Card** class (a different Collection class)
- Note: The Card class **MUST** inherit the **CardData** class (provided in the **Required Files** folder).
- Initialize the object
 - The class constructor **MUST** call the **generateFace()** and **generateSuit()** methods contained in the inherited **CardData** class
 - The class provides the following **public** methods
 - getFace()** – Returns the card's face (2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A)
 - getValue()** – Returns the card's value (number between 0 and 12)
 - toString()** – Returns a String containing the card's face and suit values
- F. The **PriorityQueue** class (start with the **PriorityQueue** class provided in the **Required Files** folder)
- Note 1: The **PriorityQueue** class **MUST** implement the **Queueable** interface.
- Note 2: The **PriorityQueue** class represents a player's hand.
- Initialize the object
 - Create a **Card** array using the specified size to store **Card** objects
 - Implement all of the methods defined in the **Queueable** interface
- G. The **Queue** class (start with the **Queue** class provided in the **Required Files** folder)
- Note 1: The **Queue** class **MUST** implement the **Queueable** interface.
- Note 2: The **Queue** class represents a player's discard pile.
- Initialize the object
 - Create a **Card** array using the specified size to store **Card** objects
 - Implement all of the methods defined in the **Queueable** interface

H. The **Stack** class (start with the Stack class provided in the **Required Files** folder)

Note 1: The Stack class **MUST** implement the **Stackable** interface.

Note 2: The Stack class represents the deck.

1. Initialize the object
 - a. Create a **Card** array using the specified size to store Card objects
2. Implement all of the methods defined in the Stackable interface





Additional Notes

- Refer to chapter 4 of the textbook for assistance with the stack and queue algorithms
- **NO** additional methods or classes are allowed
- Do **NOT** modify any methods, classes or interfaces
- Ensure the source code conforms to the coding standards for the class
- Format the output using the **String.format** method
- Remove as much **redundant (duplicate) code** as possible. If a process must be completed multiple times, try to find a way to write the code only once, but still perform the code multiple times.

Game Rules (specifically for this project)

1. There are 4 players, and each player receives 7 cards
2. Cards can **ONLY** exist in the Deck (Stack), a player's Hand (PriorityQueue) or a player's Discard pile (Queue)
3. Game play
 - A. Starting with the player at position 0 (Player 1)
 1. The player checks his/her Hand for pairs (2 cards) of matching cards
 - a. If a pair exists, the 2 cards are moved to the player's Discard pile
 2. Next, the player tries to make a match with another player's cards
 - a. The player selects the smallest card in his/her Hand
 - b. Starting with the player in the next-higher position (NP)
 1. The player asks the NP "Do You Have Any" and states the card value
 2. If the NP has a card with the requested card value
 - a. The card is removed from the NP's Hand
 - b. The card is inserted into the player's Hand
 - c. Perform steps 3.A.1 and 3.A.1.a
 - d. The player's turn is over, and the next player's turn starts (go to step 3.A)
 3. If the NP does not have the requested card
 - a. The player moves to the next NP (3.A.2.b.1)
 3. If none of the NPs had the requested card
 - a. The player draws a card from the Deck and inserts the card into his/her Hand
 - b. Perform steps 3.A.1 and 3.A.1.a
 - c. Play continues to the next player (go to step 3.A)
 4. Game play continues until one of the following two conditions occur
 - A. Any player discards all of the cards in his/her Hand (no cards left in his/her Hand)
 - B. No more cards exist in the Deck

Example Output

Important Note: The suit images     ONLY work correctly in the NetBeans environment. If you are using an environment other than NetBeans, be sure to pass **false** as the second parameter of the CardData.generateSuit method to display H S D C in place of the suit images.

```
run:
Ima Java Programmer
Project 2: Go Fish!

After 4 rounds:

Player 1
  Discard: 2♥ 2♣ 5♣ 5♥ 6♣ 6♥ 9♦ 9♣
  Hand: J♦
Player 2
  Discard: J♣ J♣ 3♥ 3♣ 7♣ 7♣ Q♣ Q♦
  Hand:
Player 3
  Discard: 5♦ 5♣ 6♦ 6♣ 8♦ 8♥
  Hand: A♣
Player 4
  Discard: 7♦ 7♥ Q♥ Q♣
  Hand: 2♣ 3♦ 9♣ 10♣

Deck: 4♥ 10♦ K♦ 4♣ 8♣ 10♥ K♥ A♣ 2♦ 8♣ 3♣ A♦ 4♣ J♥ K♣ 10♣ A♥ 9♥ 4♦ K♣
BUILD SUCCESSFUL (total time: 0 seconds)
```