Go Fish Project Documentation

***Team Members and Roles***

* ***Harrison Daniel Dsouza:*** *Project Manager and Lead Developer*
* ***Misha Punj****: Developer and Tester*
* ***Neil Patrick Saldanha*** *: Developer and Documentation Specialist*
* ***Sukhmanpreet Singh Aulakh****: Developer and Documentation Specialist (2)*

Table of Contents

[Introduction 2](#_Toc174552544)

[Updated UML Diagram 3](#_Toc174552545)

[**Source Code Overview** 3](#_Toc174552546)

[Class Summaries 3](#_Toc174552547)

[Game Description 4](#_Toc174552548)

[Game Functionality 4](#_Toc174552549)

[SOLID Principles 5](#_Toc174552550)

[Object-Oriented Programming (OOP) Principles 5](#_Toc174552551)

## Introduction

The objective of the Go Fish game project is to design and implement a digital version of the classic card game "Go Fish." Deliverable 3 includes updates based on the feedback received on our initial design. The document outlines the finalized UML class diagram, source code structure, game playability, and testing outcomes.

***Git Repository URL:***

* [***https://github.com/HarrisonDsouza/Project-GoFish.git***](https://github.com/HarrisonDsouza/Project-GoFish.git)

# Updated UML Diagram

A screenshot of a computer program

Description automatically generated

The UML class diagram has been updated to reflect the feedback from Deliverable 2. It includes the attributes and methods of each class, as well as the relationships between them.

## Source Code Overview

The Go Fish game is implemented using a well-structured and object-oriented design approach. The main classes include Card, Deck, Player, and GoFishGame.

## Class Summaries

* **Card**: Represents a single playing card with a suit and rank.
  + **Attributes**: suit, rank
  + **Methods**: Constructor, getSuit(), getRank(), toString()
* **Deck**: Represents a deck of 52 playing cards.
  + **Attributes**: cards
  + **Methods**: Constructor, drawCard(), getSize()
* **Player**: Represents a player in the game.
  + **Attributes**: name, hand, books
  + **Methods**: Constructor, getName(), getHand(), addCard(), hasCardOfRank(), transferCards(), checkAndBook(), getBookCount()
* **GoFishGame**: Manages the game flow and logic.
  + **Attributes**: players, deck
  + **Methods**: Constructor, dealCards(), playRound(), isOver(), announceWinner()

## Game Description

The Go Fish game is implemented using several classes, each with a distinct role. The Card class represents an individual playing card with attributes for suit and rank. This class includes methods for retrieving the card's suit and rank, as well as a string representation method that makes it easier to display and log the card's details. Moving on, the Deck class simulates a standard deck of 52 playing cards. It contains a list of Card objects and provides functionalities such as shuffling the deck and drawing a card, which are crucial for the initial card distribution and the "Go Fish" mechanic when players need to draw cards.

The Player class models a game player, managing their name, hand of cards, and collection of books (pairs of cards). Player class provides methods to add cards to the player's hand, and to check if a player has a card of a certain rank, and it also has methods for transferring cards between players. Additionally, the player can check and form books from their hand, moving pairs of cards to their book collection, which is central to the game’s objective. Each player’s current hand and books are managed and updated using these methods.

The game’s logic and flow are encapsulated in the GoFishGame class. This class initializes the game with a list of players and a new deck of cards. It deals the initial set of cards to each player and then manages the sequence of turns. During each turn, players can ask for cards of a specific rank from their opponents. If the requested player has the card, they hand it over; otherwise, the asking player must "Go Fish" by drawing a card from the deck. This class also handles the end-game conditions by checking if all books have been formed or if the deck is empty. When the game ends, it announces the winner based on the number of books each player has collected.

Overall, the design follows object-oriented principles, where each class has a clear responsibility, and the interactions between classes are well-defined to facilitate a seamless and logical game flow.

The codebase is organized into separate files for each class, with clear documentation and comments for each method and attribute. This organization promotes maintainability and scalability.

## Game Functionality

* Players can ask for cards by rank.
* If the requested card is available, it is transferred; otherwise, the player "fishes."
* The game continues until all books are formed.

## SOLID Principles

In the Go Fish game project, the SOLID principles used are:

1. **Single Responsibility Principle (SRP)** - Each class has a specific responsibility (e.g., Card for card properties, Deck for managing the deck, Player for managing hands and books, and GoFishGame for game flow).
2. **Open/Closed Principle (OCP)** - The design allows for extending functionality (e.g., adding new card types or game rules) without modifying existing code

## Object-Oriented Programming (OOP) Principles

1. **Encapsulation**
   * **Implementation**: Each class encapsulates its data and behavior:
     + **Card**: Encapsulates the suit and rank attributes, providing methods to access and manipulate these attributes.
     + **Deck**: Encapsulates the collection of cards and methods to manage them.
     + **Player**: Encapsulates the player's hand and books, along with methods for interacting with other players.
     + **GoFishGame**: Encapsulates the game state and flow, managing interactions between players and the deck.
2. **Abstraction**
   * **Implementation**: The classes abstract away the complexities of their operations:
     + **Card** abstracts the details of a playing card's suit and rank.
     + **Deck** abstracts the operations of card management, such as drawing and shuffling.
     + **Player** abstracts the interactions a player can have, such as asking for cards and forming books.
     + **GoFishGame** abstracts the entire game flow, making the game logic easier to understand and manage.