**SYST 17796 TEAM PROJECT**

Team Name: Sunday Afternoon

**Group Members:**

Jinyoung Jeon

Juyoung Jung

Tamara Dang

Winston Martinez

Deliverable 1

June 24, 2021

Contents

[Team Contract 1](#_Toc75094020)

[Domain Class Diagram 4](#_Toc75094021)

[Design Document Template 5](#_Toc75094022)

# Team Contract

*Please negotiate, sign, scan and include as the first section in your Deliverable 1.*

**Please note that if cheating is discovered in a group assignment each member will be charged with a cheating offense regardless of their involvement in the offense. Each member will receive the appropriate sanction based on their individual academic honesty history.**

**Please ensure that you understand the importance of academic honesty. Each member of the group is responsible to ensure the academic integrity of all of the submitted work, not just their own part. Placing your name on a submission indicates that you take responsibility for its content.**

**For further information read Academic Honesty Policy on AccessSheridan.**

|  |  |  |
| --- | --- | --- |
| Team Member Names (Please Print) | Signatures | Student ID |
| Project Leader:  Tamara Dang | Shape  Description automatically generated with medium confidence | 991223558 |
| Winston Martinez | A picture containing text  Description automatically generated | 991602676 |
| Jinyoung Jeon | Jinyoung Jeon | 991622283 |
| Juyoung Jung | A picture containing text  Description automatically generated | 991627394 |

By signing this contract, we acknowledge having read the Sheridan Academic Honesty Policy as per the link below.

<https://policy.sheridanc.on.ca/dotNet/documents/?docid=917&mode=view>

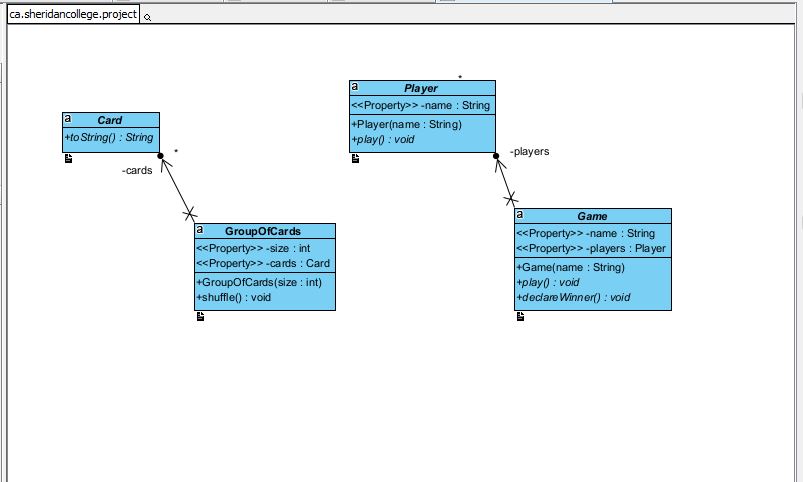
## Responsibilities of the Project Leader include:

* Assigning tasks to other team members, including self, in a fair and equitable manner.
* Ensuring work is completed with accuracy, completeness and timeliness.
* Planning for task completion to ensure timelines are met
* Any other duties as deemed necessary for project completion

## What we will do if . . .

| Scenario | Accepted initials | We agree to do the following |
| --- | --- | --- |
| Team member does not deliver component on time due to severe illness or extreme personal problem | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team absorbs workload temporarily \_\_ 2. Team seeks advice from professor \_\_ 3. Team shifts target date if possible \_\_ 4. Other: We must update one another in advance, if anything seems to come up. |
| Team member cannot deliver component on time due to lack of ability | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team reassigns component \_\_ 2. Team helps member \_\_ 3. Team member must ask professor for reference material \_\_ 4. Other: We must communicate each other to find a solution/ reach out to get the second opinion. |
| Team member does not deliver component on time due to lack of effort | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team absorbs workload \_\_ 2. Team "fires" team member by not permitting his/her name on submission \_\_ 3. Other: |
| Team member does not attend team meeting | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team proceeds without him/her and will assign work to the absent member \_\_ 2. Team doesn't proceed and records team member's absence \_\_ 3. Team proceeds for that meeting but "fires" member after \_\_ occurrences \_\_ |
| An unforeseen constraint occurs after the deliverable has been allocated and scheduled (a surprise test or assignment) | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team meets and reschedules deliverable \_\_ 2. Team will cope with constraint \_\_ 3. Other: we’ll try to have an immediate team meeting for the case |
| Team cannot achieve consensus leaving one member feeling "railroaded", "ignored", or "frustrated" with a decision which affects all parties | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team agrees to abide by majority vote \_\_ 2. Team flips coin \_\_ 3. c) Other: |
| Team members do not share expectations for grade desired | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team will elect one person as "standards-bearer" who has the right to ask that work be redone \_\_ 2. Team votes on each submission's quality \_\_   (Team members can give suggestions/ feedbacks before submission)   1. Team will ask for individual marking and will identify sections by author \_\_ 2. Other: |
| Team member behaves in an unprofessional manner by being rude or uncooperative | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team attempts to resolve the issue by airing the problem at team meeting \_\_ 2. Team requests meeting with professor to problem-solve \_\_ 3. Team ignores behaviour \_\_ 4. Team agrees to avoid use of all vocabulary inappropriate to the business setting \_\_ |
| Team member assumes or requests that his/her name be signed to a submission but has not participated in production of the deliverable | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team agrees that this is cheating and is unethical \_\_ 2. Friends are friends and should help each other \_\_ 3. c) Team will submit with signature but will advise professor who will take action \_\_ |
| There is a dominant team member who is content to make all decisions on the team's behalf leaving some team members feeling like subordinates rather than equal members | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team will actively solicit consensus on all decisions which affect project direction by asking for each member's decision and vote \_\_ 2. Team will express subordination feelings and attempt to resolve issue \_\_ 3. c) Other: |
| Team has a member who refuses to participate in decision making but complains to others that s/he wasn't consulted | TD  JJ  A picture containing night sky  Description automatically generated  WM | 1. Team forces decision sharing by routinely voting on all issues \_\_ 2. Team routinely checks with each other about perceived roles \_\_ 3. c) Team discusses the matter at team meeting \_\_ |

# Domain Class Diagram



# Design Document Template

## OVERVIEW

### Project Background and Description

#### Project goals and final vision

Our project goal is to create an interactive card game program using Java language. The program must function properly and smoothly. On the coding side, each class should be loosely coupled so that classes operate independently, and it will help us maintain the whole project more efficiently. Also, we will achieve high cohesion by describing a class as a single entity and making them logically connected to each other. Our final vision is to accomplish writing clean code through the entire project using debugging tools and code refactoring, applying OO concepts, and cooperation with team members on GitHub.

#### How to play the "War" card game

1. Each player gets dealt half the deck, 26 cards, and the cards are put face down in a stack in front of the players.
   * Both players turn their top card face up at the same time.
2. The person with the higher card wins that draw and takes both the cards.
   * Card Rank : (Highest) A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2 (Lowest)
3. If both players draw a card of the same rank, then there's a war.
   * The face up cards are left on the table and each player puts three cards face down on the table, and then puts one card face up.
   * The face up card determines who wins the war and gets all 10 cards that are on the table at this point.
   * If the face up card is again the same rank, then the war goes on, three more face down, one face up etc.
4. Once a player has all 52 cards, the person wins the game.
   * Or if a player finishes their cards during a war without having enough cards to finish the war, then the player loses immediately.

##### References

*How To Play War*. Gather Together Games. (n.d.). <https://gathertogethergames.com/war>.

*WAR Card Game*. Card Game | Play it online. (n.d.). <https://cardgames.io/war/>.

Wikimedia Foundation. (2021, April 16). *War (card game)*. Wikipedia. <https://en.wikipedia.org/wiki/War_(card_game)>.

#### Description of the base code

There are four base classes. The relationship between a concrete class GroupOfCards and an abstract class Card is association. The GroupOfCards class has two attributes; one is "size" whose type is integer and the other is "cards" whose type is ArrayList<Card>. In this class, we can define the size of "group of cards" and mix the "group of cards" using shuffle() method. The Card class represents a card that our team chose (a regular deck of playing cards). We will have to override toString() method in its child classes. Another relationship between abstract class Game and Player is also association. The Game class models our game. It also has two attributes; one is "name" whose type is String and the other is "players" whose type is ArrayList<Player>. In this class, we will define the title of the game, and get an ArrayList of players from Player class. We will have to override play() method and declareWinner() method in its child classes. The Player class has an attribute "name" and its type is String. When we extends this class, we will have to override play() method.

### Project Scope

#### Member Roles

This project comprises of four main roles, with such responsibilities as follows:

* **Programmer:** Responsible for writing code.
* **Code reviewer:** Responsible for ensuring that code conforms to scope/requirements outlined in design document.
* **QA/Bug Tester:** Responsible for testing code to try and break or exploit it.
* **Project lead:** Responsible for checking in on members and status of their tasks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Role | Jinyoung | Juyoung | Tamara | Winston |
| **Programmer** | Card.java  Main.java | Game.java  Main.java | GroupOfCards.java  Main.java | Player.java  Main.java |
| **Code Reviewer** |  |  | X | X |
| **QA/Bug Tester** | X | X |  |  |
| **Project Lead** |  |  | X |  |

#### Technical Scope

The project will be coded in Java and will be played through the console. The project will be considered complete when the requirements outlined High-Level Requirements are fulfilled with no errors during runtime.

### High-Level Requirements

The new system must include the following:

* Ability for each player to register with the game
* Ability for a player to play a card
* Ability for player to check number of cards in hand
* Ability for player to forfeit the game
* Ability for the game to determine the value of the card
* Ability for the game to determine which card is ranked higher
* Ability for the game to communicate a win or loss
* Ability for players to play another round
* Ability to display final score when players no longer want to keep playing

The new system may include the following:

* Display card face in ASCII art

The new system will not include the following:

* Ability for players to change their names

### Implementation Plan

**GitHub Repository:**

<https://github.com/WinMillz/SundayAfternoonCodingProject.git>

The expected use: each member checks in code at the end of each week as a regular routine as well as whenever we are told to have updates in code by team members. Codes that each member designs will be saved to this repository when it is pushed from a local repository.

Under the “Text files” directory, text files are stored. UML diagrams have their own folder which is “UML”. All the codes are under “src/ca/sheridancollege/project”.

Team Sunday Afternoon follows Programs industry standard as coding standards:

1. Easier to read: code that is difficult to read will take longer to decipher, even if you wrote it yourself. Proper standards like good indentation, spacing, documentation, self-documenting identifiers are required.
2. Easier to maintain: clients frequently require changes to existing programs. Whether you wrote the program or someone else did, it is difficult to modify and maintain code that violates standard practices: it can be difficult to find the module you need to change or the values that need to be updated.
3. Easier to debug and avoid coding errors.
4. More robust and reliable
5. More user-friendly and consistent
   * Standards for Java
     1. Naming Conventions
        + Primitive and Object Variables
        + Classes
        + Methods
        + Code
     2. Using Proper Indentation
        + Line length
        + Using proper spacing
        + Documentation

We expect ourselves to use NetBeans, Visual Paradigm Professional, and GitHub.

### Design Considerations

When coding our project, we intend to adhere to the object-oriented principles of encapsulation, delegation, and flexibility/maintainability.

#### Encapsulation

The principle of encapsulation is demonstrated in the GroupOfCards class. The “cards” and “size” variables are declared private to keep them hidden from other classes in the program. This allows the program to instantiate a group of cards while ensuring that the instantiated group of cards (for example, a players hand) cannot be inadvertently altered. The getters (getCards and getSize) in this class allow access to the private data members.

For the game we have chosen in particular (War), we could use this class two instantiate two groups of cards of size 26 for each player. We could use the getSize method to track when a player is down to zero cards which would be a condition for terminating the current round of play.

The Player class also makes use of encapsulation. The “name” variable is declared private. Since this is an abstract class, we would need to create subclasses to instanstiate the methods of the Player class, but once these subclasses have been implemented encapsulation is still used to keep the data member hidden.

In the game of War, we could use the getName method to print a prompt to the console indicating which player’s turn it is. This method could also be used to declare a winner at the end of the round being played.

#### Delegation

The principle of delegation is present in the Game class where one of the instance variables is an ArrayList of Player objects. In this scenario Game is the delegator and a Player object is the delegate. In this case, once the subclass for the Player object is created, the getPlayers method can be called with the particular index of the desired Player object and use the play method(currently not implemented) from the Player class can be used.

This is also demonstrated in the GroupOfCards class where an ArrayList of Card objects is listed as an instance variable. Here, GroupOfCards is the delegator, and a Card object is the delegate. If we wanted to call upon the toString method (currently not implemented because there is no subclass for Card yet) we could use the getCards method from GroupOfCards with the desired index to access the card object. The toString method in this case could be used to display the Card object’s data to the Users.

#### Flexibility/Maintainability

Ensuring loose coupling and high cohesion within our code allows us to be flexible and reduces maintenance.

Separating the Cards and GroupOfCards classes is an example of loose coupling. The Cards class is responsible for defining the total deck that we are working with. GroupOfCards, on the other hand, defines how these cards can be combined. If we want to play war with only half a deck, we just need to modify GroupOfCards and we do not have to touch Cards. If we want to create a different game using a deck of cards, we can reuse the Cards class without needing to remove any code. This reduces the amount of work needed for maintenance and lets us be more flexible with what we can create.

To maintain high cohesion in future classes, we will create classes that are responsible for only one thing. For example, when the cards played are the same value, the players go to “war.” We should separate the action playCard() and the condition required for “war” checkWar(). This allows us to reuse playCard() in another game. It would also allow us to easily find and change the conditions for “war” if we wanted to change the game to compare suits instead of card value.