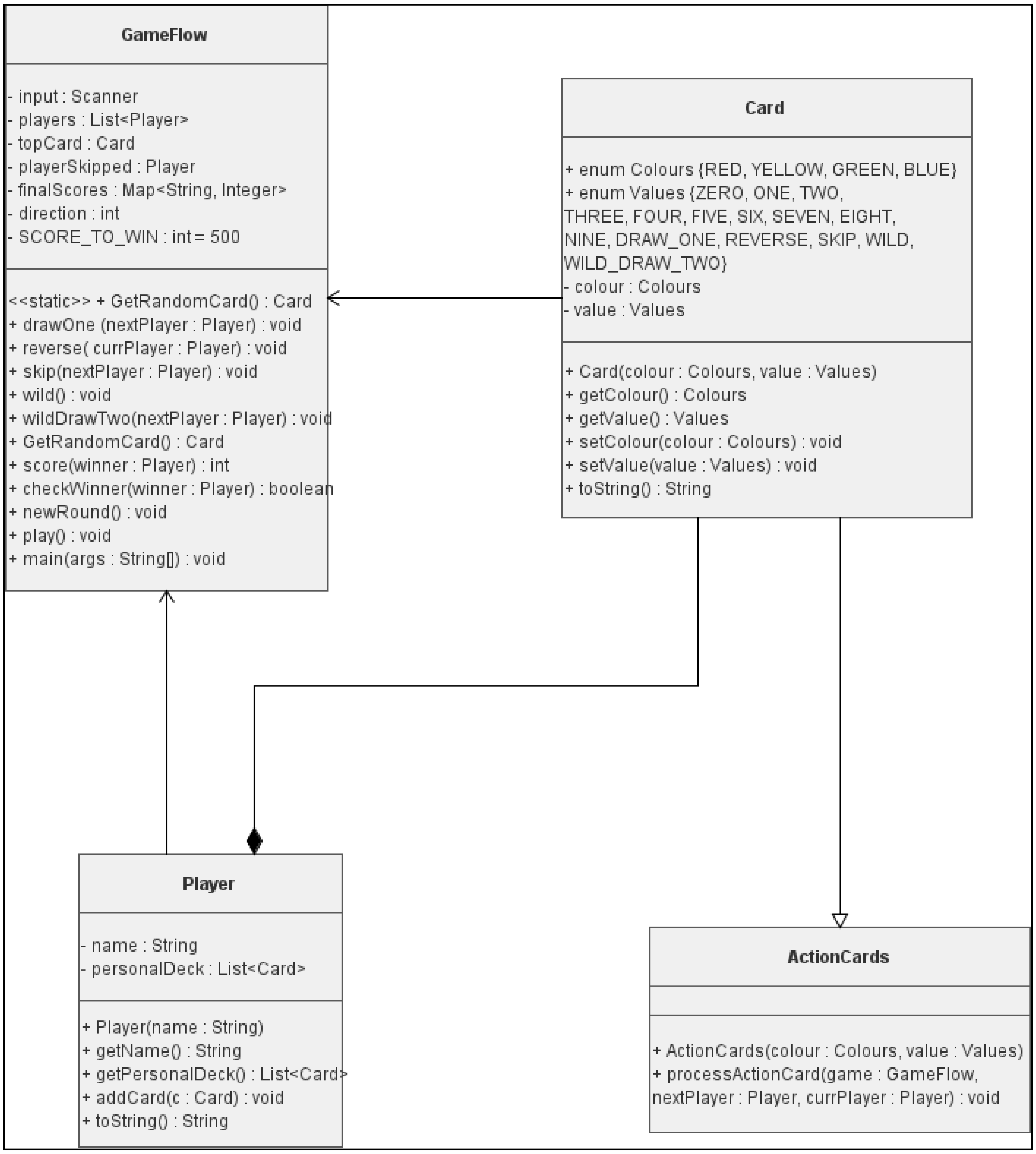
SYSC 3110 - A
Software Development Project

Milestone 1

Team 21:
Marc Aoun, 101263718
Iman Elabd, 101232751
Naima Mamun, 101276213
Amreen Shahid, 101306199

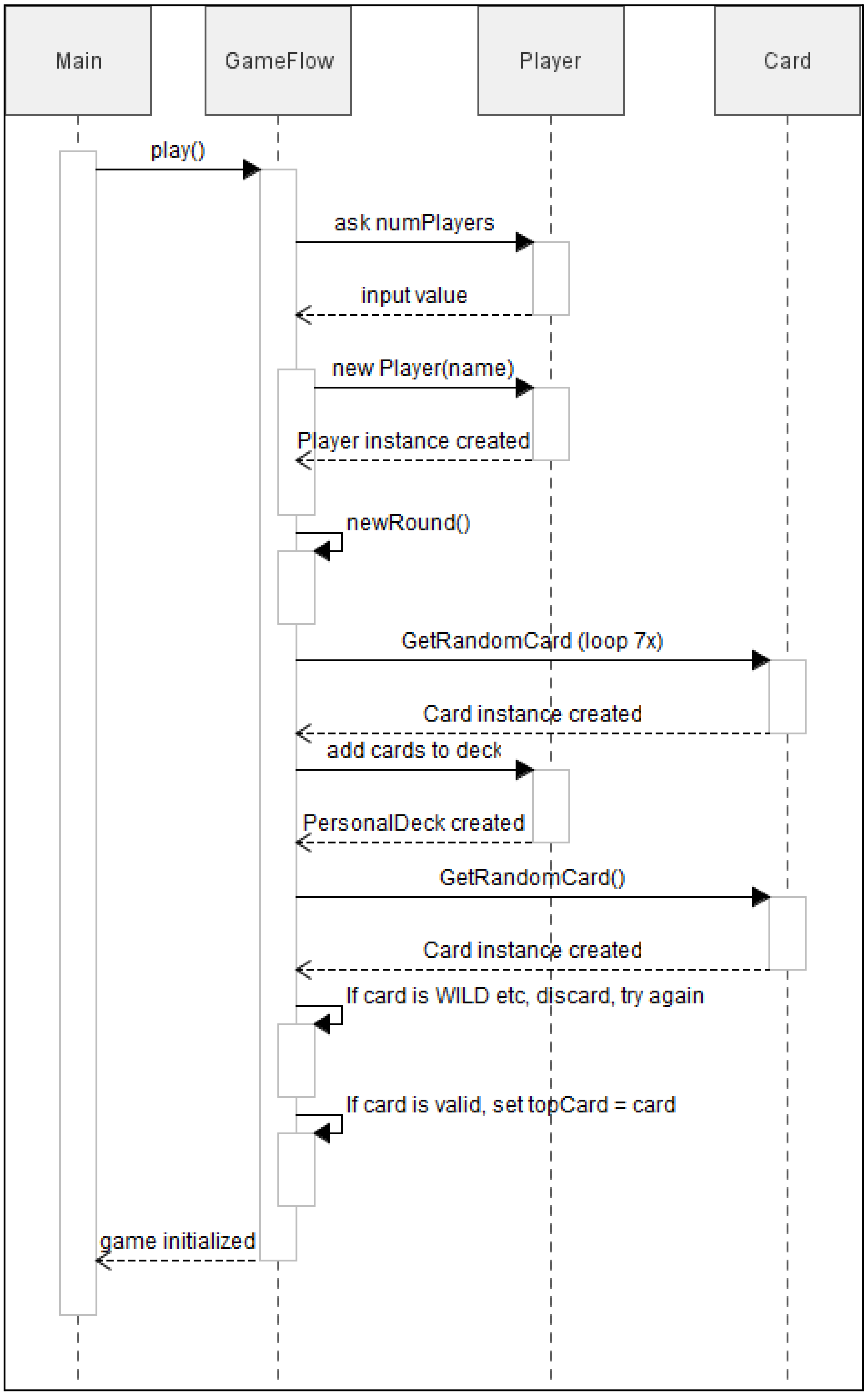
Submitted on: October 27, 2025

1. UML Class Diagram

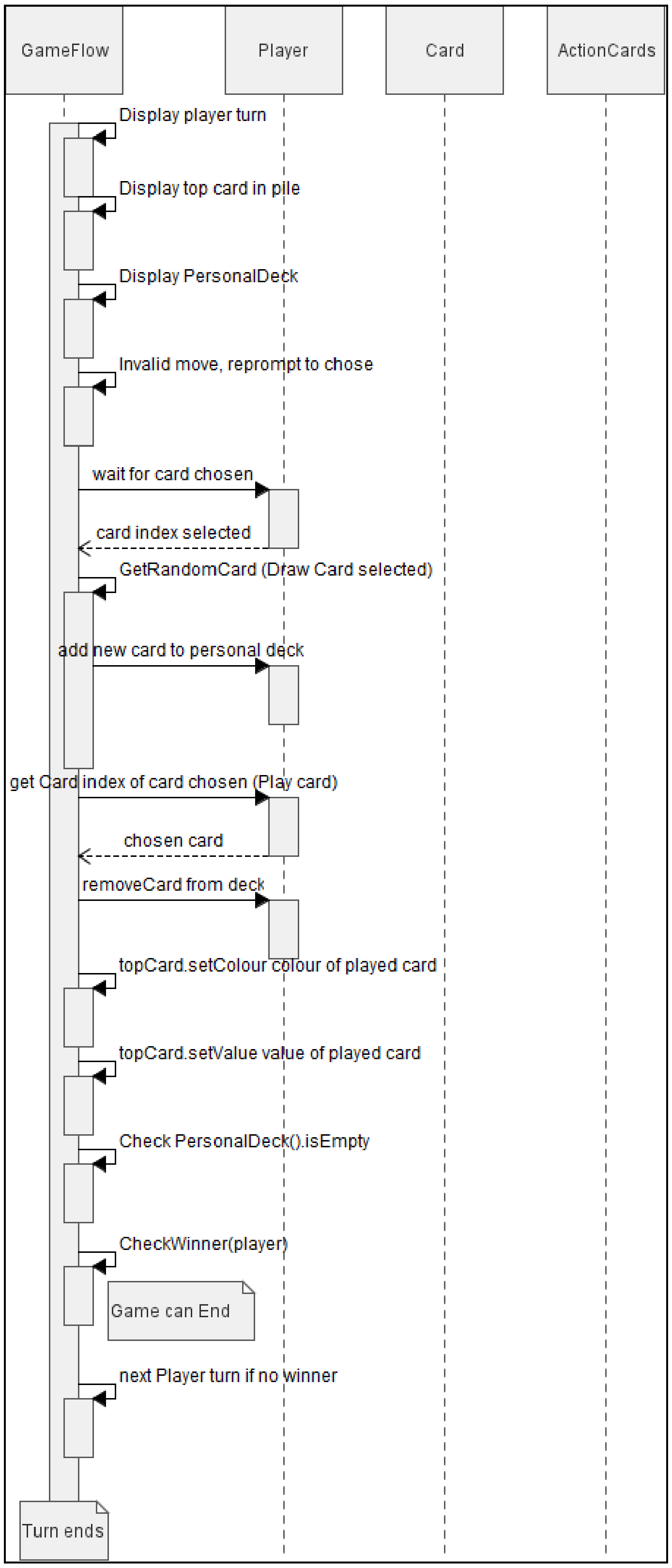

GameFlow
- input : Scanner
- players : List<Player>
- topCard : Card
- playerSkipped : Player
- finalScores : Map<String, Integer>
- direction : int
- SCORE_TO_WIN : int = 500
<<static>> + GetRandomCard() : Card
+ drawOne (nextPlayer : Player) : void
+ reverse(currPlayer : Player) : void
+ skip(nextPlayer : Player) : void
+ wild() : void
+ wildDrawTwo(nextPlayer : Player) : void
+ GetRandomCard() : Card
+ score(winner : Player) : int
+ checkWinner(winner : Player) : boolean
+ newRound() : void
+ play() : void
+ main(args : String[]) : void
Card
+ enum Colours {RED, YELLOW, GREEN, BLUE}
+ enum Values {ZERO, ONE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, DRAW_ONE, REVERSE, SKIP, WILD, WILD_DRAW_TWO}
- colour : Colours
- value : Values
+ Card(colour : Colours, value : Values)
+ getColour() : Colours
+ getValue() : Values
+ setColour(colour : Colours) : void
+ setValue(value : Values) : void
+ toString() : String
Player
- name : String
- personalDeck : List<Card>
+ Player(name : String)
+ getName() : String
+ getPersonalDeck() : List<Card>
+ addCard(c : Card) : void
+ toString() : String
ActionCards
+ ActionCards(colour : Colours, value : Values)
+ processActionCard(game : GameFlow, nextPlayer : Player, currPlayer : Player) : void

2. Sequence Diagrams

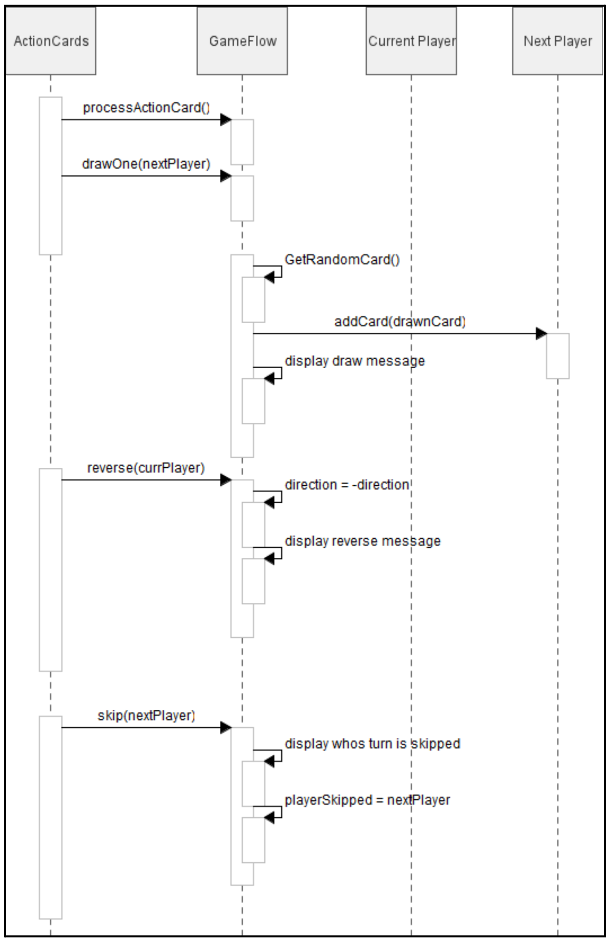
2.1 Game Initialization

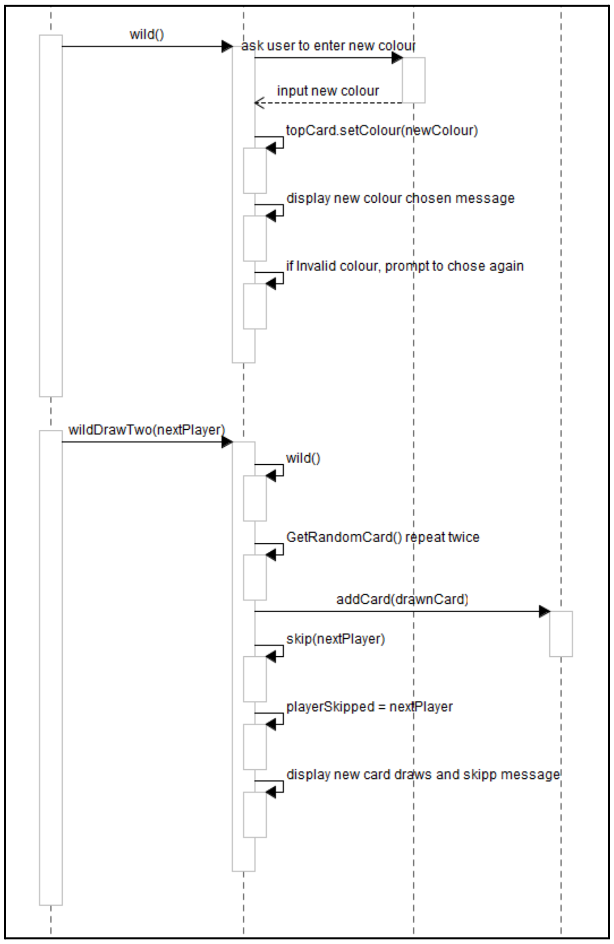


2.2 Player Turn Normal Conditions


GameFlow
Player
Card
ActionCards
Display player turn
Display top card in pile
Display PersonalDeck
Invalid move, reprompt to chose
wait for card chosen
card index selected
GetRandomCard (Draw Card selected)
add new card to personal deck
get Card index of card chosen (Play card)
chosen card
removeCard from deck
topCard.setColour colour of played card
topCard.setValue value of played card
Check PersonalDeck().isEmpty
CheckWinner(player)
Game can End
next Player turn if no winner
Turn ends

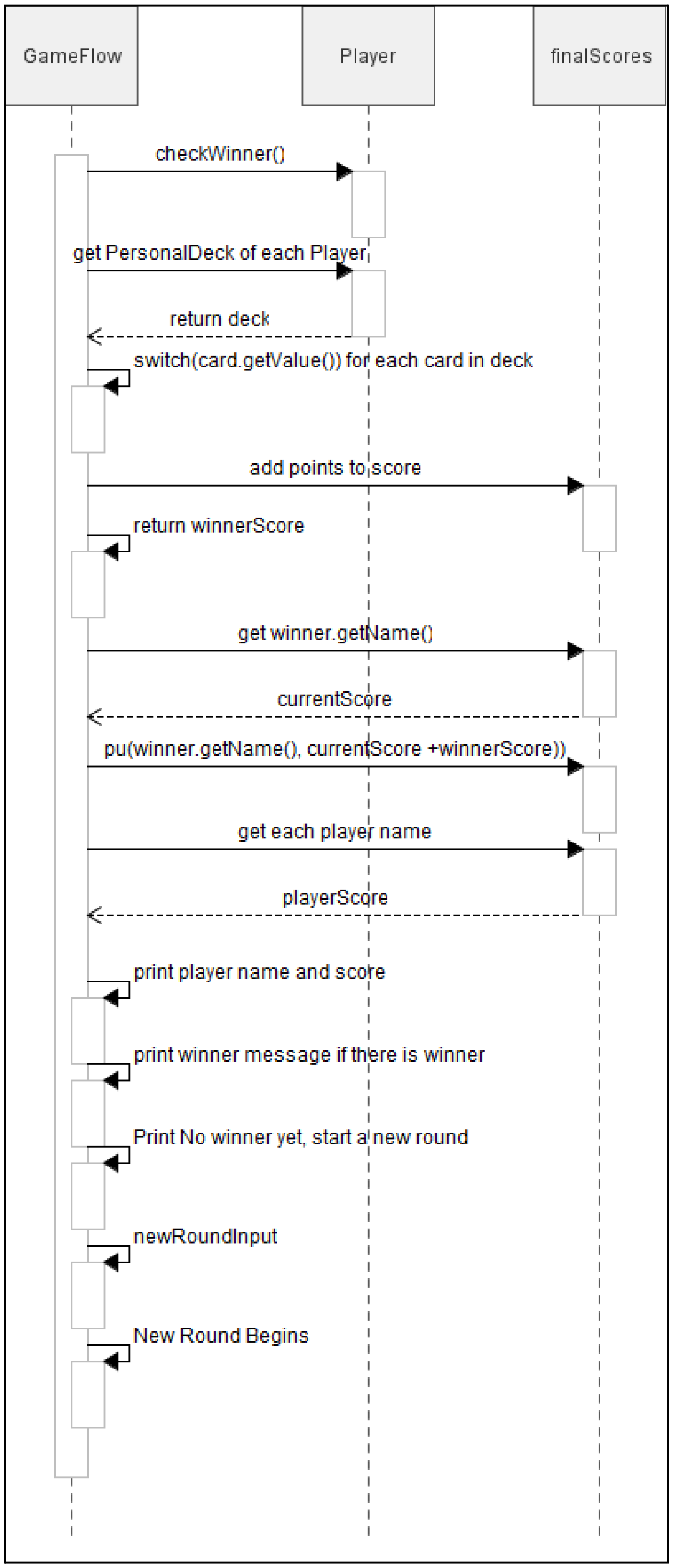
2.3 Action Cards Execution




wild()
ask user to enter new colour
input new colour
topCard.setColour(newColour)
display new colour chosen message
if Invalid colour, prompt to chose again
wildDrawTwo(nextPlayer)
wild()
GetRandomCard() repeat twice
addCard(drawnCard)
skip(nextPlayer)
playerSkipped = nextPlayer
display new card draws and skipp message

Would take place after the player has chosen a card to play.

2.4 Scoring and Round Completion


GameFlow
Player
finalScores
checkWinner()
get PersonalDeck of each Player
return deck
switch(card.getValue()) for each card in deck
add points to score
return winnerScore
get winner.getName()
currentScore
pu(winner.getName(), currentScore +winnerScore))
get each player name
playerScore
print player name and score
print winner message if there is winner
Print No winner yet, start a new round
newRoundInput
New Round Begins

3. Explanation of Data Structures and Design Used

3.1 Data Structures

We used the ArrayList data structure in the following cases including for the attribute “players” which is a list of Player class instances and for the attribute “personalDeck” which is an array list of Card class instances.

- “private List<Player> players” in the GameFlow class. This was necessary because the list of players needed to be stored in an ordered manner which helped with the player order, and was easy to access next player through indexing to determine the player’s turns.
- “private List<Card> personalDeck” in Player class. This was necessary because each player needed to have a set of cards that was not only ordered but accessible by their index, so the player can type in the index of the card they want to play. It is also easy to remove the cards from the list and to shift the cards in the list, for when the player decides to use a card which discards it from their personal deck. It’s easy method of inserting a card is also helpful for when the player decides to draw a new card, or when they have to gain two cards.

We also used HashMaps as “private Map<String, Integer> finalScores” in the GameFlow class. This map stored a string of the player’s name as the key that associates with an integer of their scores. A hash map was used because of its ease of finding specific values without having to loop through multiple other values. For our game, only one score is stored with each player so finding one player’s score is really easy by looking up their name, allowing an efficient way to store and update the players’ scores.

3.2 Design Decisions

Our game was designed using the Event/Observer Pattern. The GameFlow is the overall controller, and the player objects can be viewed as the observers. This is seen as, for example when a player plays a special card, GameFlow notifies other players, and handles the necessary shared state changes (direction of play, player’s turns, card pile’s top card) and updates the data of all players.

4. Breakdown of Project Tasks and Responsibilities of Each Member

Name:	Tasks and Responsibilities:
Marc Aoun	Responsible for implementing the following: Game Flow: - Initialize players. The ability to play with 2 – 4 players. - Display the cards held by each player. - Pass Turn Functionality: Implement functionality for a player to pass their turn and draw a card. - Card Placement Validation: Ensure that card placement is valid according to the game rules. Resultant State Observation: Display the resultant state of the game in text - Format after each turn (discard pile, and the next player's cards). - Display the resultant state of the game in text format after each turn (discard pile, and the next player's cards).
Iman Elabd	Responsible for implementing the following: Junit Testing: Thoroughly test all classes and methods. Ensure code is running smoothly. Find and debug any errors in code. Assist other team members when necessary.
Naima Mamun	Responsible for implementing the following: UML Modeling and Data Structures: - UML Modeling: class diagrams with complete variable and method signatures. - Sequence Diagrams: for important game scenarios. - Data Structure Explanation: detailed description of the choice of data structures and their relevant operations. Documentation: Present a well-documented project. - Readme File: explains the rest of the deliverables, lists the contributions of each team member for current and past milestones, and explains any known issues. - JavaDocs and Code Comments: JavaDocs are present and well-written for classes and methods.
Amreen Shahid	Responsible for implementing the following:

	Uno Action Cards: Implement functionality for the following special action cards: - Draw One Card - Reverse Card - Skip Card - Wild Card - Wild Draw Two Card Card Placement: Implement card placement via the keyboard. Scoring: Implement the ability to add and update scores for each player

Note: Distribution of tasks were distributed and agreed upon by each member of the team. Additionally, description of tasks were taken straight from the provided documentation specifically, the rubric and project description document.