Design

Table of contents

Use Cases 1

Proposed User Interface 2

List of Objects 4

Properties of main objects used 4

Class Responsibility Table 5

Class Relationships Diagram 6

UML Diagrams 7

Flowcharts 10

Key Algorithms 19

Use Cases

Figure 1 illustrates the user surrounded by the possible user-program interactions (ordered clock-wise).

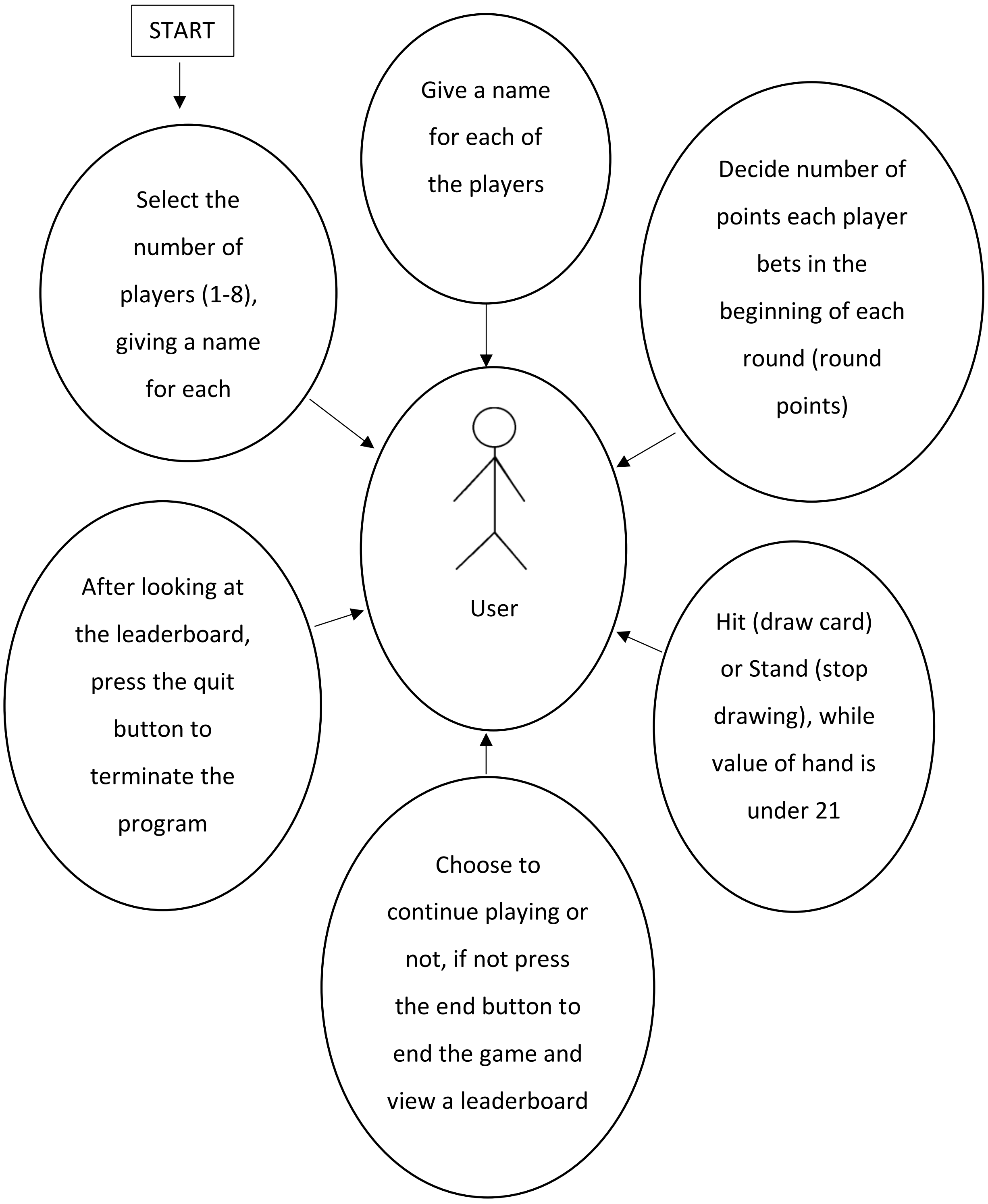

START
Select the number of players (1-8), giving a name for each
Give a name for each of the players
Decide number of points each player bets in the beginning of each round (round points)
User
Hit (draw card) or Stand (stop drawing), while value of hand is under 21
Choose to continue playing or not, if not press the end button to end the game and view a leaderboard
After looking at the leaderboard, press the quit button to terminate the program

Figure 1: Use Cases Diagram

Proposed User Interface

Figure 2 and **3** depict my user interface proposal. User's comments are in red as taken from interviews.

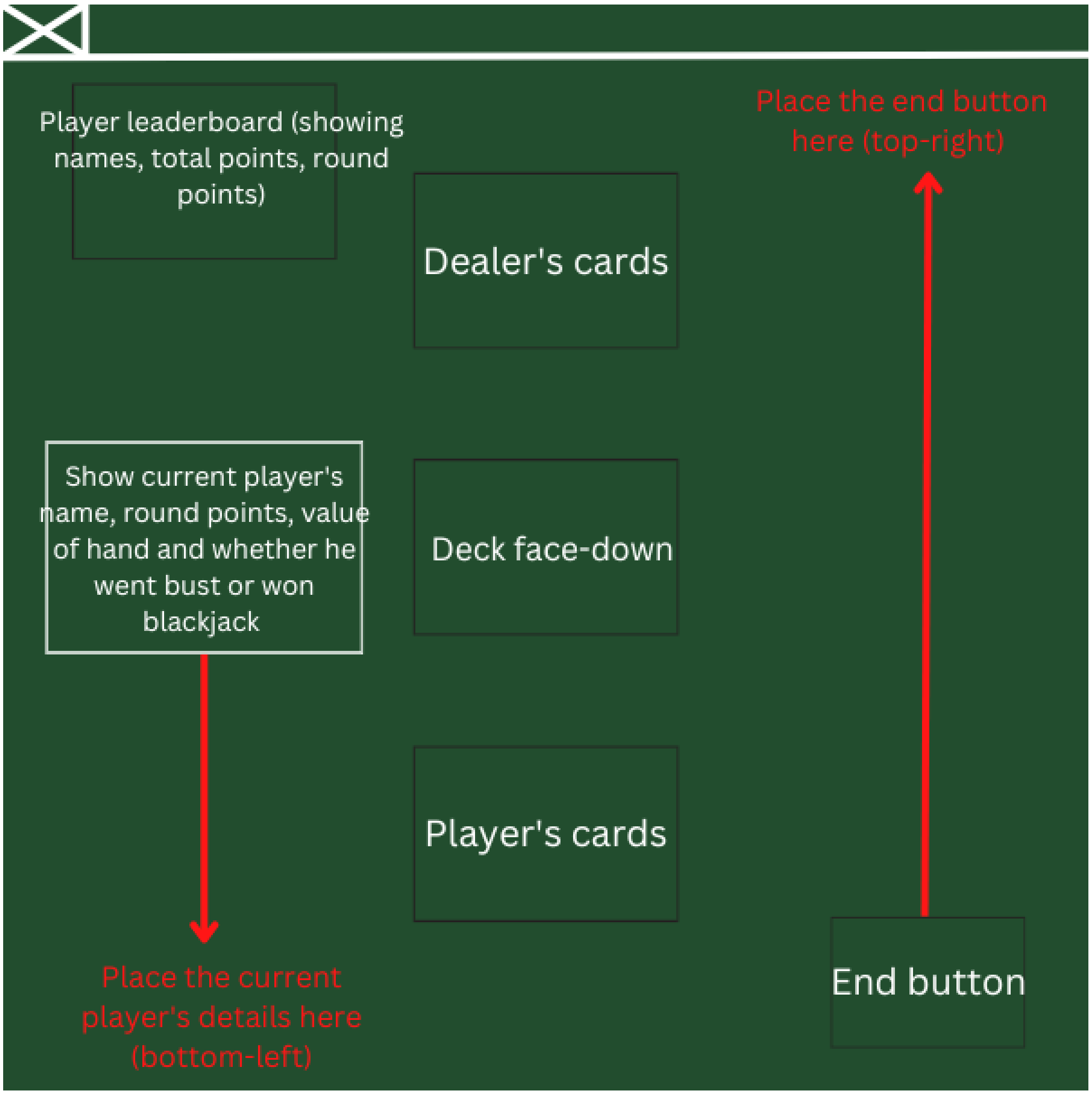

Player leaderboard (showing names, total points, round points)
Place the end button here (top-right)
Dealer's cards
Show current player's name, round points, value of hand and whether he went bust or won blackjack
Deck face-down
Player's cards
Place the current player's details here (bottom-left)
End button

Figure 2: Main frame (where the game is played)

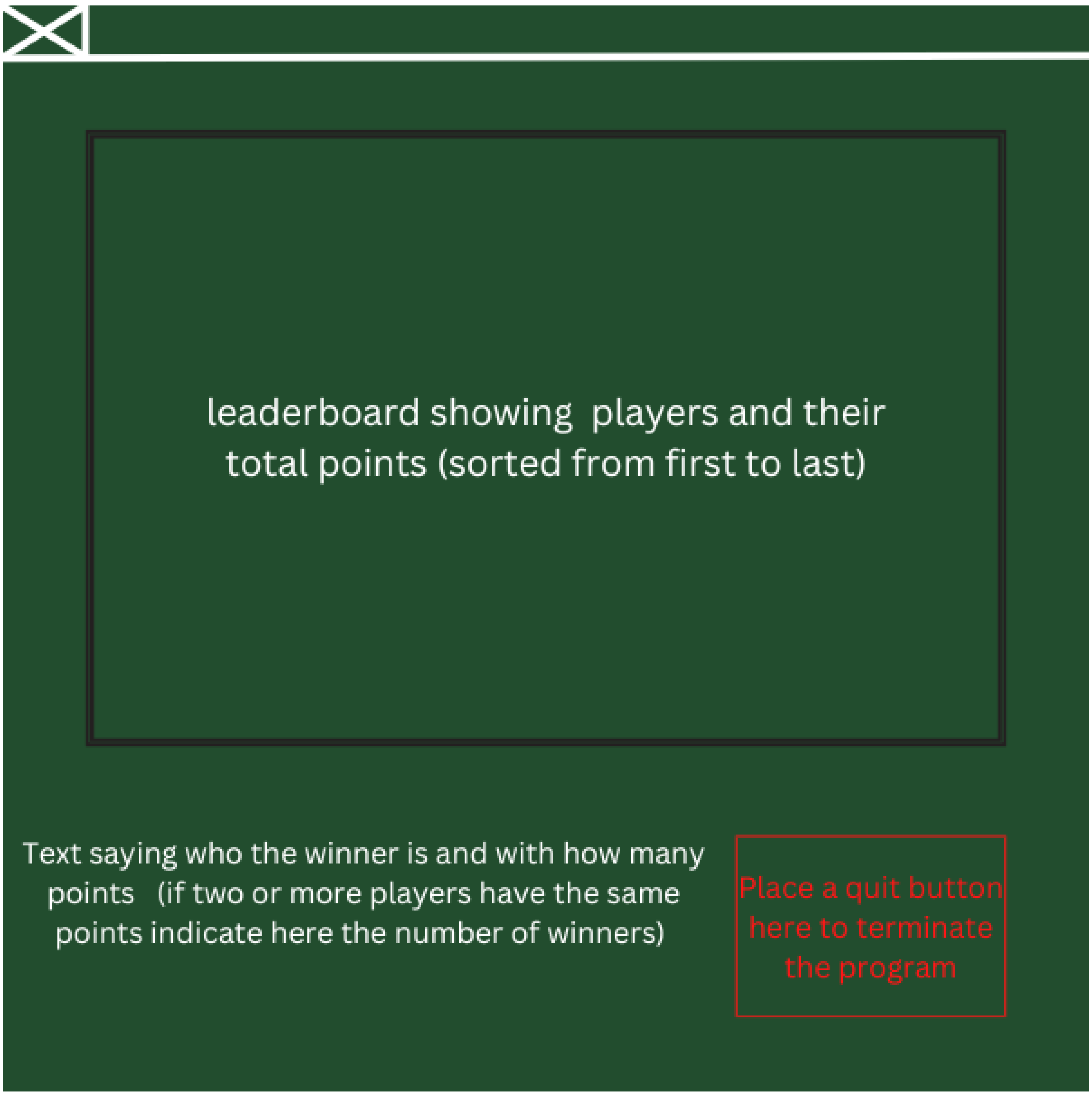

leaderboard showing players and their total points (sorted from first to last)
Text saying who the winner is and with how many points (if two or more players have the same points indicate here the number of winners)
Place a quit button here to terminate the program

Figure 3: Leaderboard frame (after the end button is pressed)

List of Objects

Properties of main objects used

Each object's variables will be declared as private, and make use of encapsulation in OOP, to prevent accidental data access. Player and Dealer will be subclasses of Person, using Person's properties.

Object	Properties
Person	• Name • Hand • Blackjack from drawing or not • Blackjack from first two cards or not • Bust (hand's value over 21) or not • Last drawn card
Player	• All person's properties • Total points • Round points (betted this round)
Dealer	• All person's properties (no points, unlike Player)
Hand	• ArrayList holding hand's cards
Deck	• ArrayList holding 312 cards (six 52-card decks)
Card	• Rank • Suit • Value • Image of the card • Number of deck it belongs (from the six decks the masterDeck consists of)

Table 1: Object Properties Table

Class Responsibility Table

Class Name	**Class Responsibilities**
Game	Main class to run, creates a GUI with player's details and cards, dealer's cards and end-button. Allows players to bet points and hit or stand. Checks for blackjack or bust, sets points.
Person	Contains its constructor, accessor and mutator methods of its fields, checks if Person's value of hand is 21 (blackjack) or over 21 (bust).
Player	Contains its constructor, checking the validity of given name, accessor and mutator methods of its fields.
Dealer	Contains its constructor, method for Dealer to draw cards while the value of hand is under 17.
Hand	Contains its constructor, an accessor method of its field (no mutator method as changes are undesired), methods to draw cards and calculate the value of a hand considering when Aces change value.
Deck	Contains its constructor, creating the master deck, accessor and mutator methods of its field and a method to reshuffle the master deck.
Card	Contains its constructors, accessor methods of its fields (no mutator methods as changes are undesired), toString method.

Table 2: Class Responsibility Table

Class Relationships Diagram

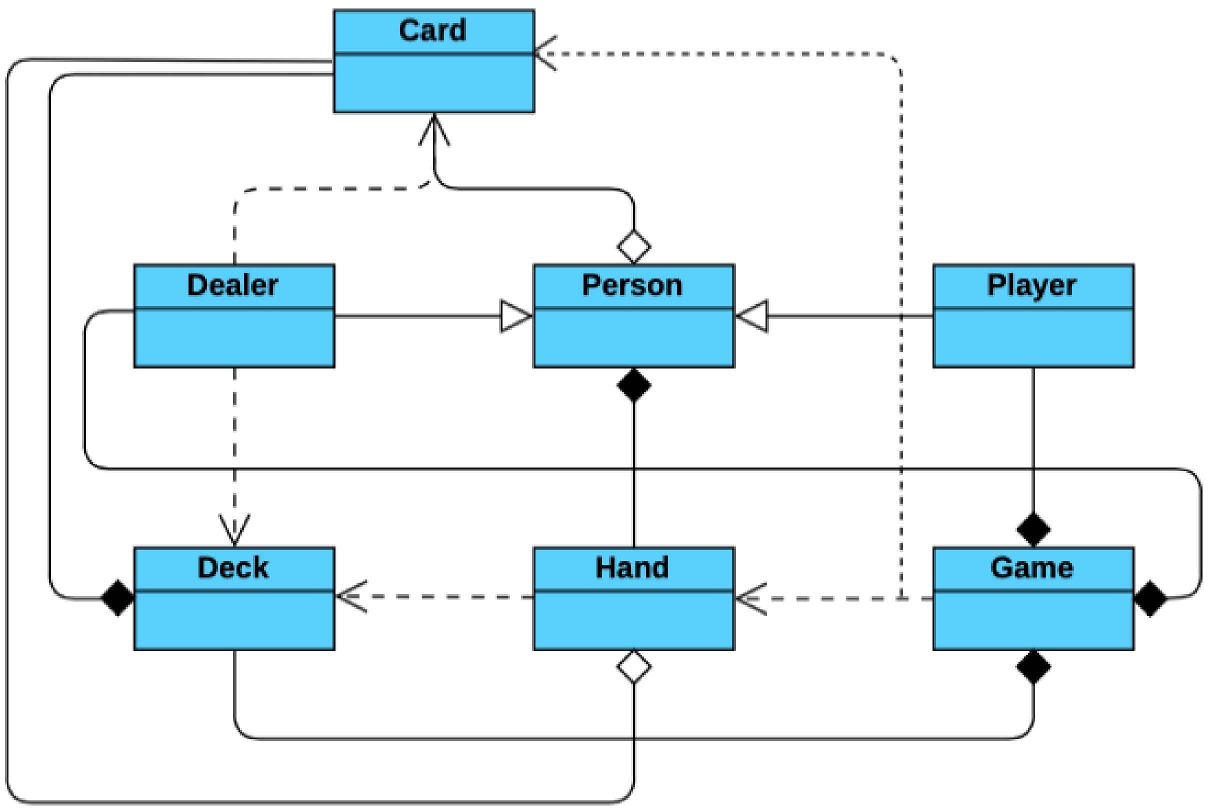
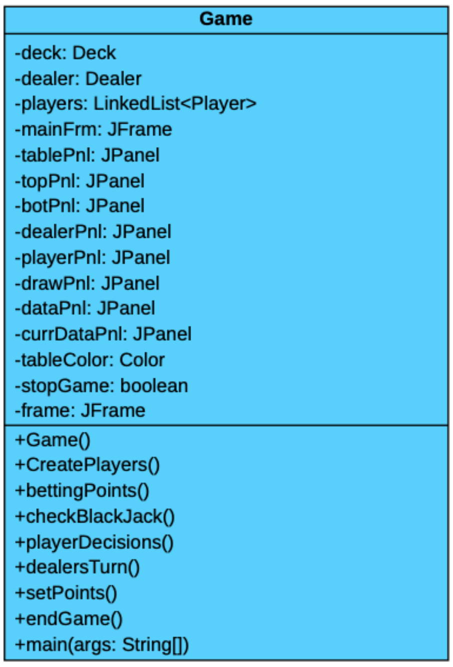


Figure 4: Class Relationships Diagram

UML Diagrams

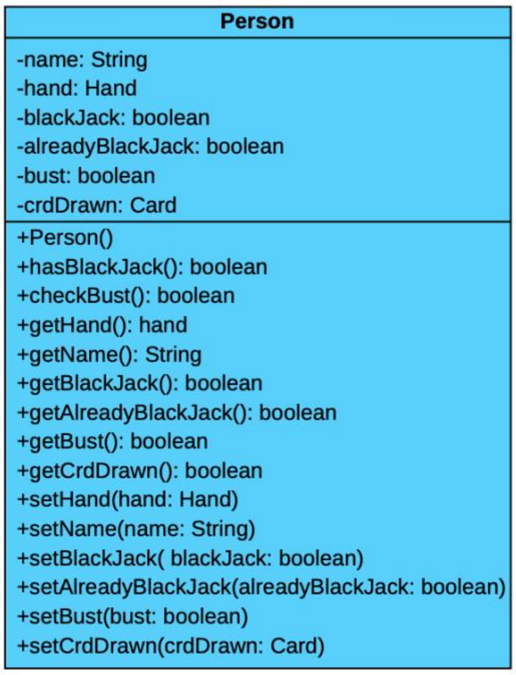


Game

-deck: Deck
-dealer: Dealer
-players: LinkedList<Player>
-mainFrm: JFrame
-tablePnl: JPanel
-topPnl: JPanel
-botPnl: JPanel
-dealerPnl: JPanel
-playerPnl: JPanel
-drawPnl: JPanel
-dataPnl: JPanel
-currDataPnl: JPanel
-tableColor: Color
-stopGame: boolean
-frame: JFrame

+Game()
+CreatePlayers()
+bettingPoints()
+checkBlackJack()
+playerDecisions()
+dealersTurn()
+setPoints()
+endGame()
+main(args: String[])

Figure 5: Game class UML diagram



Person

-name: String
-hand: Hand
-blackJack: boolean
-alreadyBlackJack: boolean
-bust: boolean
-crdDrawn: Card

+Person()
+hasBlackJack(): boolean
+checkBust(): boolean
+getHand(): hand
+getName(): String
+getBlackJack(): boolean
+getAlreadyBlackJack(): boolean
+getBust(): boolean
+getCrdDrawn(): boolean
+setHand(hand: Hand)
+setName(name: String)
+setBlackJack(blackJack: boolean)
+setAlreadyBlackJack(alreadyBlackJack: boolean)
+setBust(bust: boolean)
+setCrdDrawn(crdDrawn: Card)

Figure 6: Person class UML Diagram

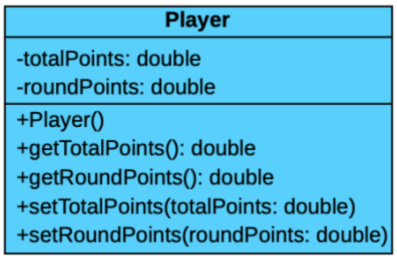

Player
-totalPoints: double
-roundPoints: double
+Player()
+getTotalPoints(): double
+getRoundPoints(): double
+setTotalPoints(totalPoints: double)
+setRoundPoints(roundPoints: double)

Figure 7: Player Class UML Diagram

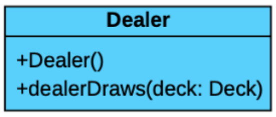


Figure 8: Dealer Class UML Diagram

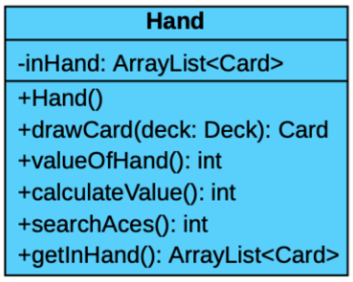


Figure 9: Hand Class UML Diagram

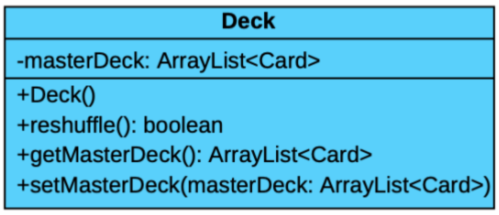

Deck
-masterDeck: ArrayList<Card>
+Deck()
+reshuffle(): boolean
+getMasterDeck(): ArrayList<Card>
+setMasterDeck(masterDeck: ArrayList<Card>)

Figure 10: Deck Class UML Diagram

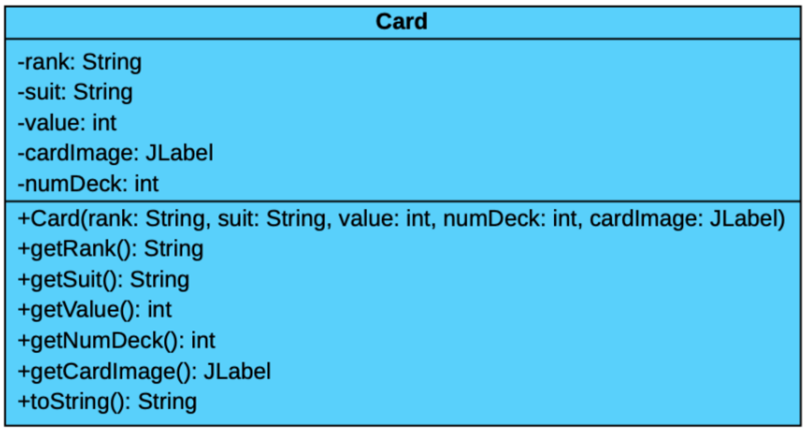


Figure 11: Card Class UML Diagram

Flowcharts

Figure 12 displays how data flows when the client uses the program. It is color coded; figure numbers refer to flowcharts analyzing key processes. These follow from **figure 13** to **21**.

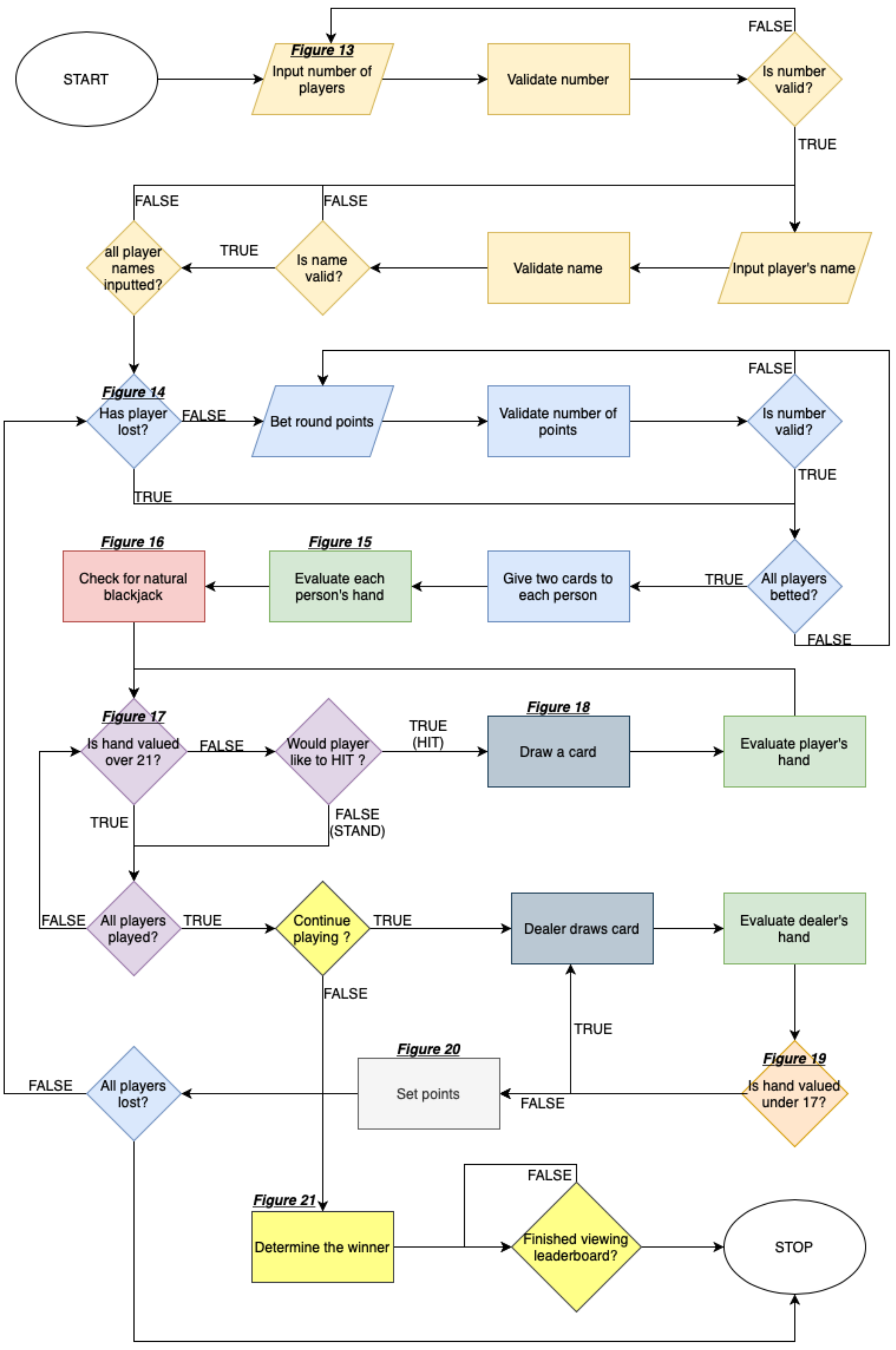


Figure 12: Flowchart for data flow from User's perspective

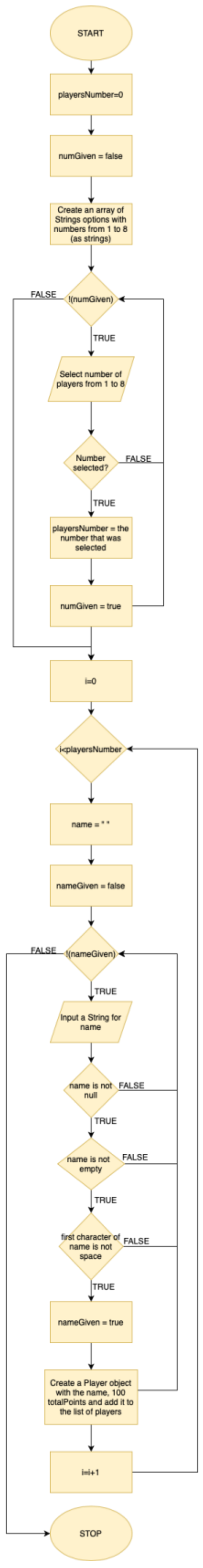
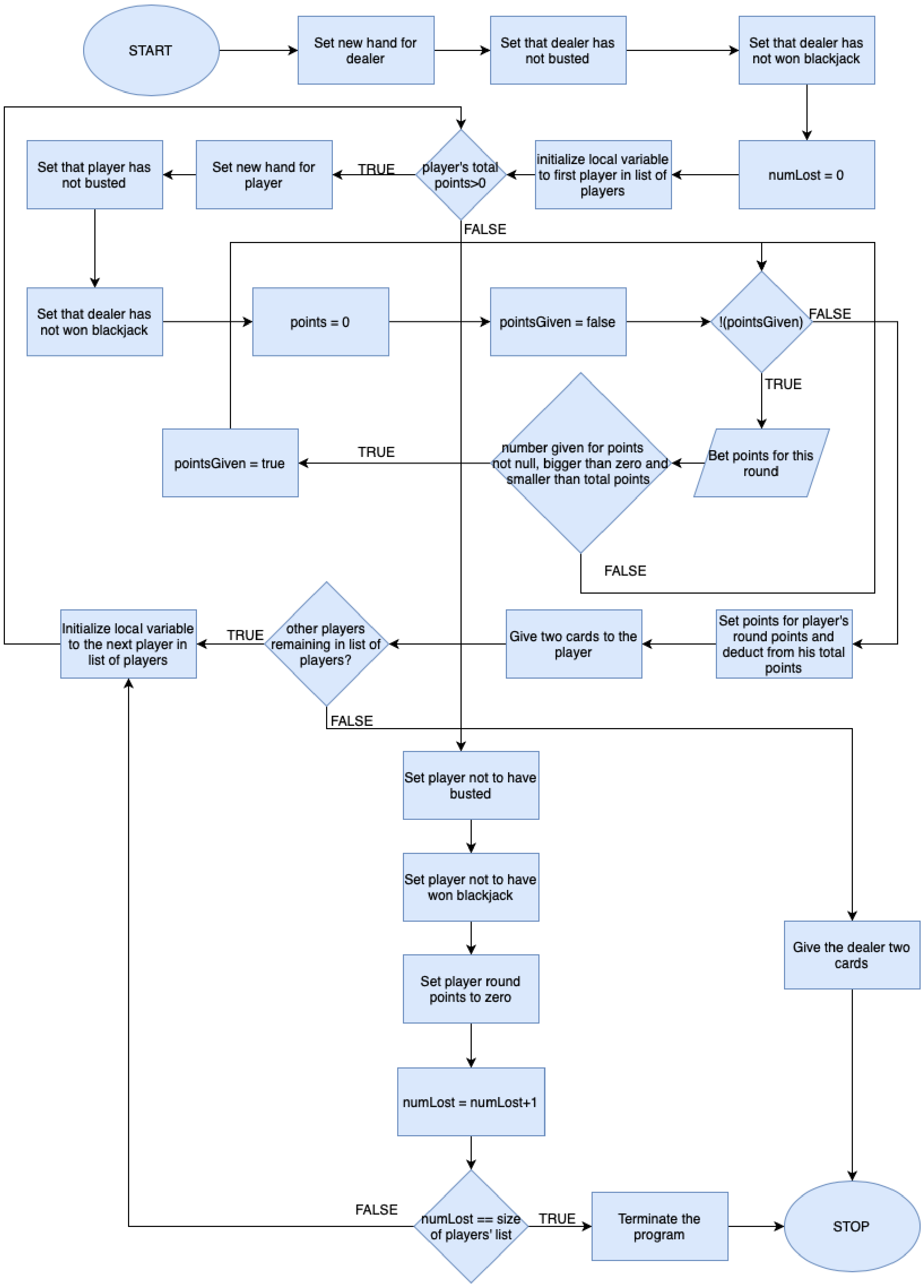

START
playersNumber=0
numGiven = false
Create an array of Strings options with numbers from 1 to 8 (as strings)
!(numGiven)
FALSE
TRUE
Select number of players from 1 to 8
Number selected?
FALSE
TRUE
playersNumber = the number that was selected
numGiven = true
i=0
i<playersNumber
name = " "
nameGiven = false
!(nameGiven)
FALSE
TRUE
Input a String for name
name is not null
FALSE
TRUE
name is not empty
FALSE
TRUE
first character of name is not space
FALSE
TRUE
nameGiven = true
Create a Player object with the name, 100 totalPoints and add it to the list of players
i=i+1
STOP

Figure 13: Creation of players (in Game class)



START
Set new hand for dealer
Set that dealer has not busted
Set that dealer has not won blackjack
numLost = 0
initialize local variable to first player in list of players
player's total points>0
TRUE
FALSE
Set new hand for player
Set that player has not busted
Set that dealer has not won blackjack
points = 0
pointsGiven = false
!(pointsGiven)
FALSE
TRUE
Bet points for this round
number given for points not null, bigger than zero and smaller than total points
TRUE
FALSE
pointsGiven = true
Set points for player's round points and deduct from his total points
Give two cards to the player
other players remaining in list of players?
TRUE
FALSE
Initialize local variable to the next player in list of players
Set player not to have busted
Set player not to have won blackjack
Set player round points to zero
numLost = numLost+1
numLost == size of players' list
FALSE
TRUE
Terminate the program
Give the dealer two cards
STOP

Figure 14: Betting points each round (in Game class)

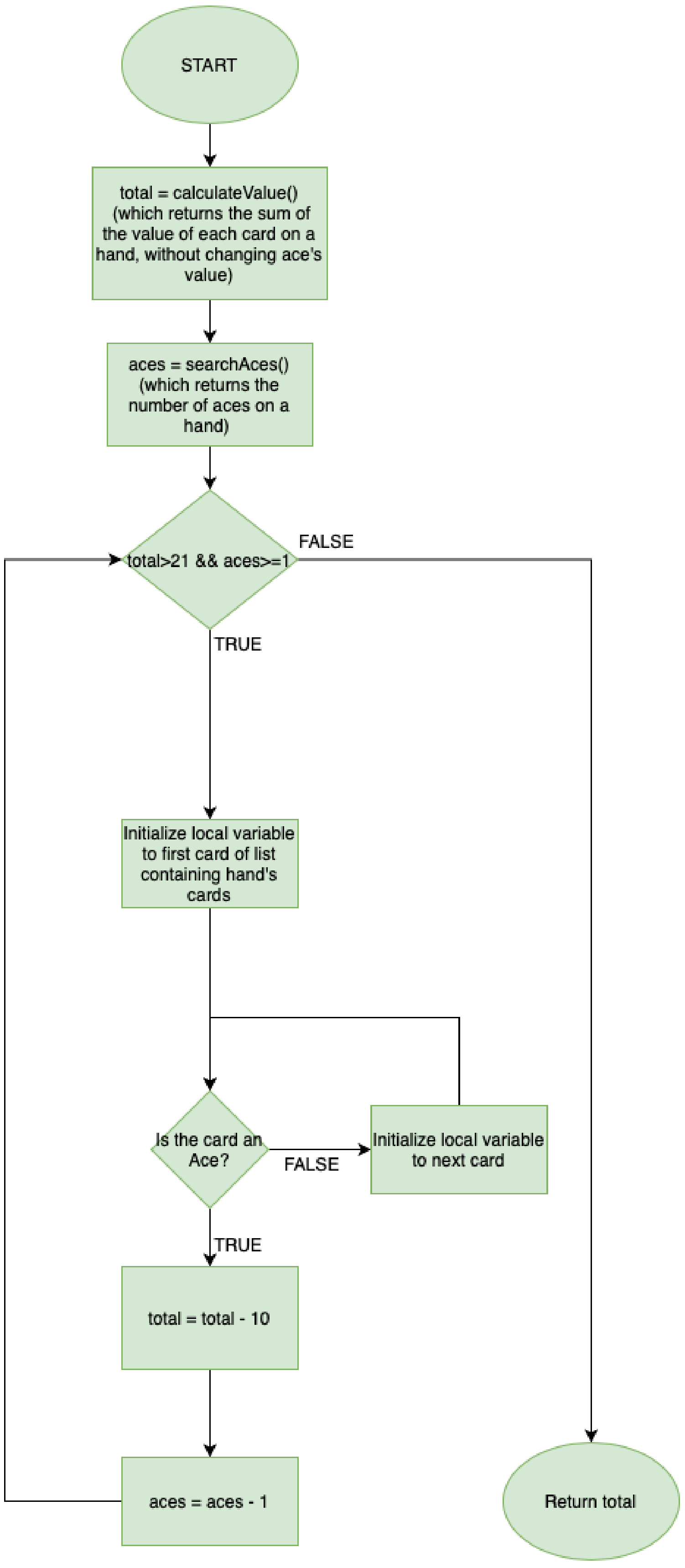

START
total = calculateValue() (which returns the sum of the value of each card on a hand, without changing ace's value)
aces = searchAces() (which returns the number of aces on a hand)
total>21 && aces>=1
FALSE
TRUE
Initialize local variable to first card of list containing hand's cards
Is the card an Ace?
FALSE
Initialize local variable to next card
TRUE
total = total - 10
aces = aces - 1
Return total

Figure 15: Value of hand (in Hand class)

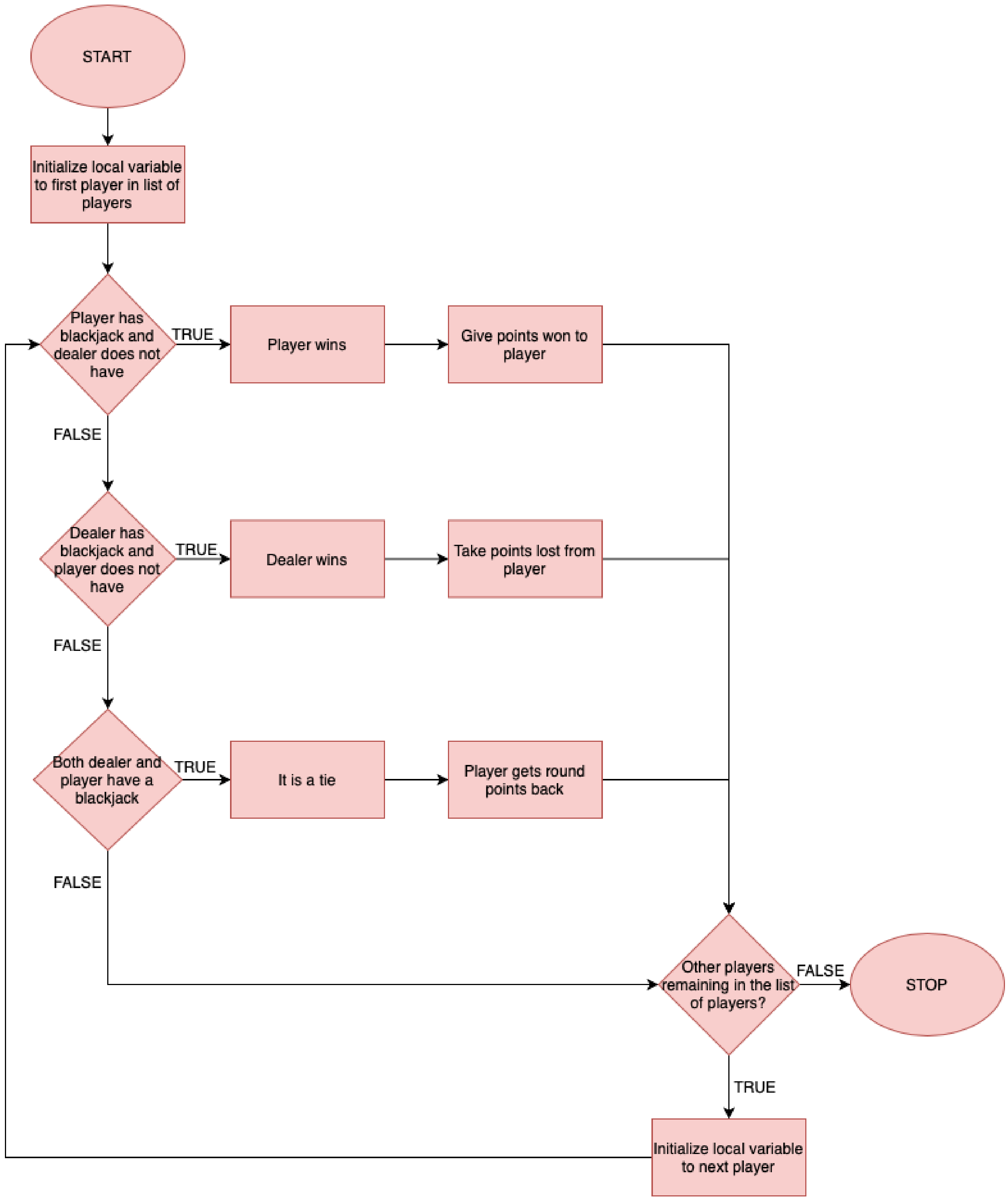

START
Initialize local variable to first player in list of players
Player has blackjack and dealer does not have
TRUE
Player wins
Give points won to player
FALSE
Dealer has blackjack and player does not have
TRUE
Dealer wins
Take points lost from player
FALSE
Both dealer and player have a blackjack
TRUE
It is a tie
Player gets round points back
FALSE
Other players remaining in the list of players?
FALSE
STOP
TRUE
Initialize local variable to next player

Figure 16: Check for natural blackjack (in Game class)

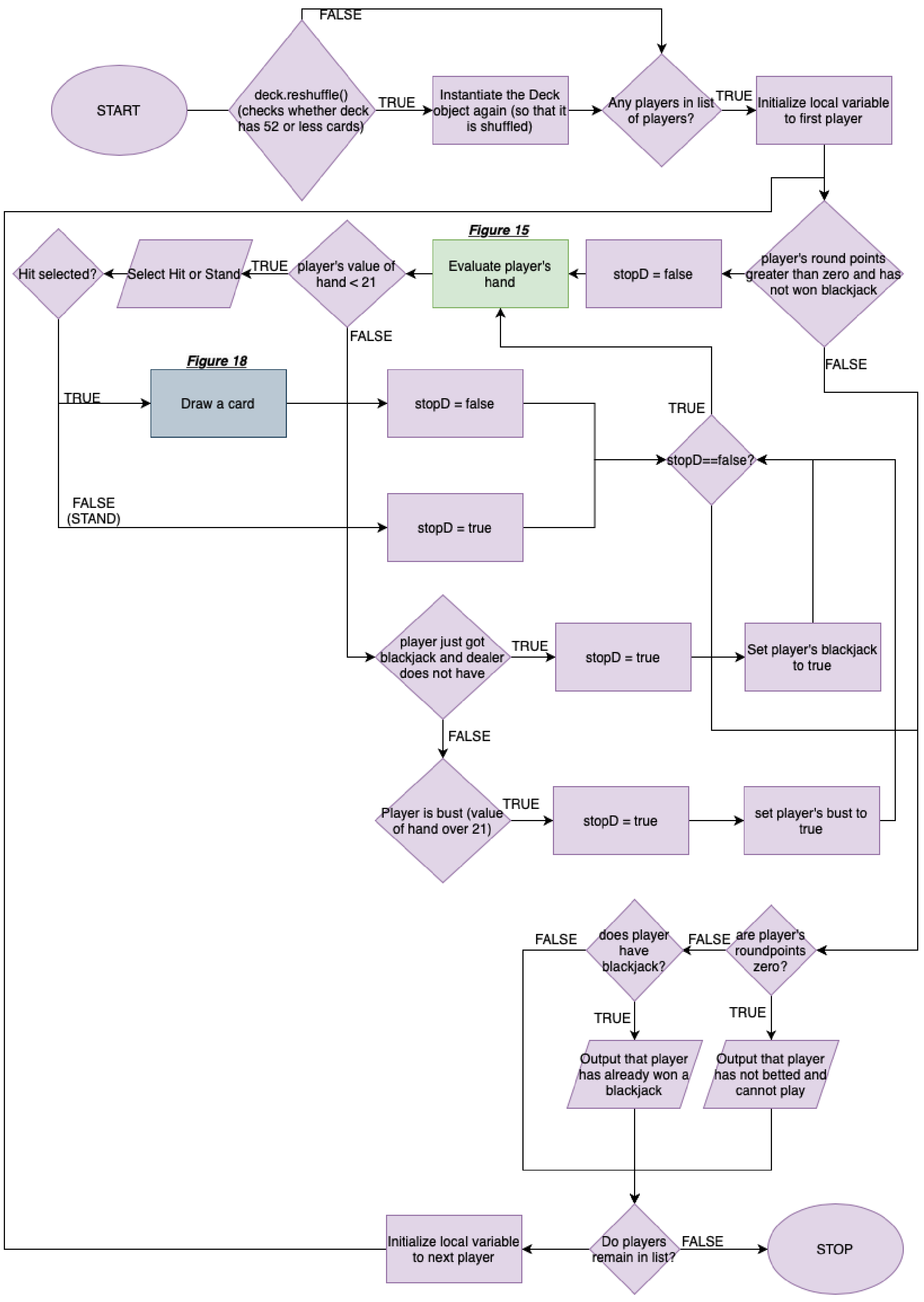

START
deck.reshuffle() (checks whether deck has 52 or less cards)
FALSE
TRUE
Instantiate the Deck object again (so that it is shuffled)
Any players in list of players?
TRUE
Initialize local variable to first player
player's round points greater than zero and has not won blackjack
FALSE
stopD = false
Figure 15
Evaluate player's hand
player's value of hand < 21
TRUE
FALSE
Select Hit or Stand
Hit selected?
TRUE
Figure 18
Draw a card
stopD = false
FALSE (STAND)
stopD = true
TRUE
stopD==false?
player just got blackjack and dealer does not have
TRUE
stopD = true
Set player's blackjack to true
FALSE
Player is bust (value of hand over 21)
TRUE
stopD = true
set player's bust to true
FALSE
does player have blackjack?
FALSE
are player's roundpoints zero?
TRUE
TRUE
Output that player has already won a blackjack
Output that player has not betted and cannot play
Initialize local variable to next player
Do players remain in list?
FALSE
STOP

Figure 17: Player's decision to Hit or Stand (in Game class)

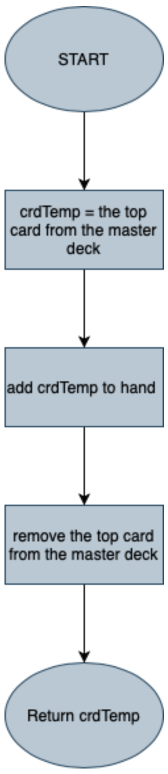

START
crdTemp = the top card from the master deck
add crdTemp to hand
remove the top card from the master deck
Return crdTemp

Figure 18: Draw a card (in Hand class)

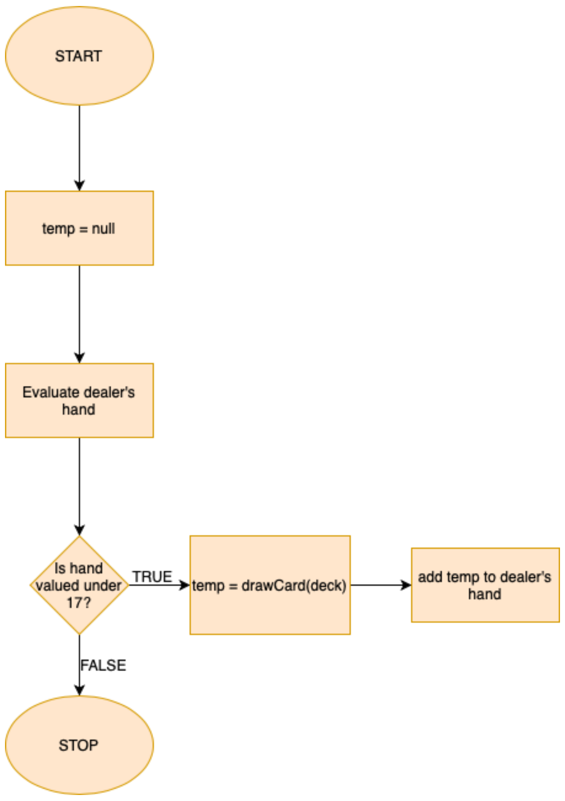


Figure 19: Dealer Draws (in Dealer class)

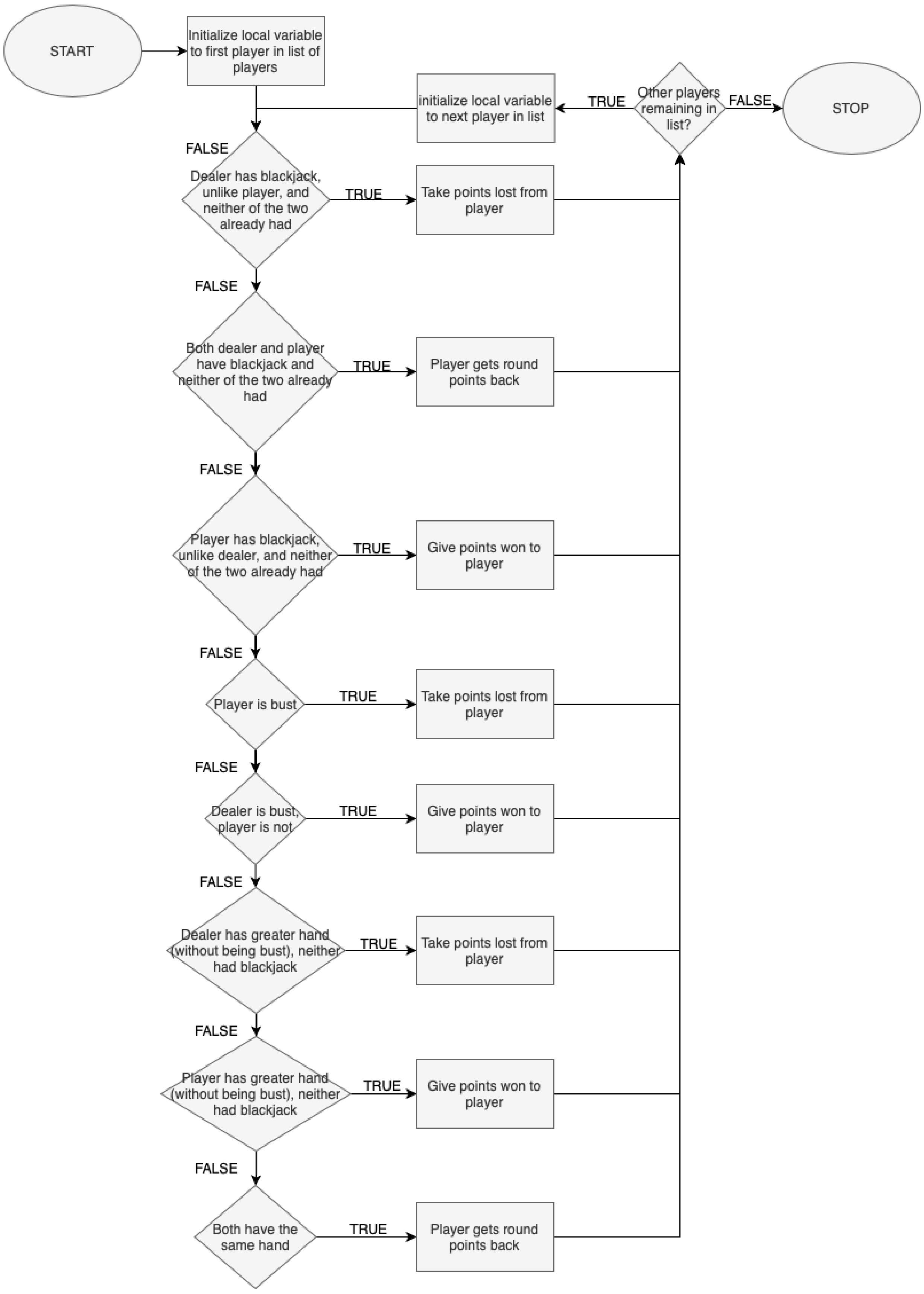

START
Initialize local variable to first player in list of players
initialize local variable to next player in list
TRUE
Other players remaining in list?
FALSE
STOP
FALSE
Dealer has blackjack, unlike player, and neither of the two already had
TRUE
Take points lost from player
FALSE
Both dealer and player have blackjack and neither of the two already had
TRUE
Player gets round points back
FALSE
Player has blackjack, unlike dealer, and neither of the two already had
TRUE
Give points won to player
FALSE
Player is bust
TRUE
Take points lost from player
FALSE
Dealer is bust, player is not
TRUE
Give points won to player
FALSE
Dealer has greater hand (without being bust), neither had blackjack
TRUE
Take points lost from player
FALSE
Player has greater hand (without being bust), neither had blackjack
TRUE
Give points won to player
FALSE
Both have the same hand
TRUE
Player gets round points back

Figure 20: Set Points (in Game class)

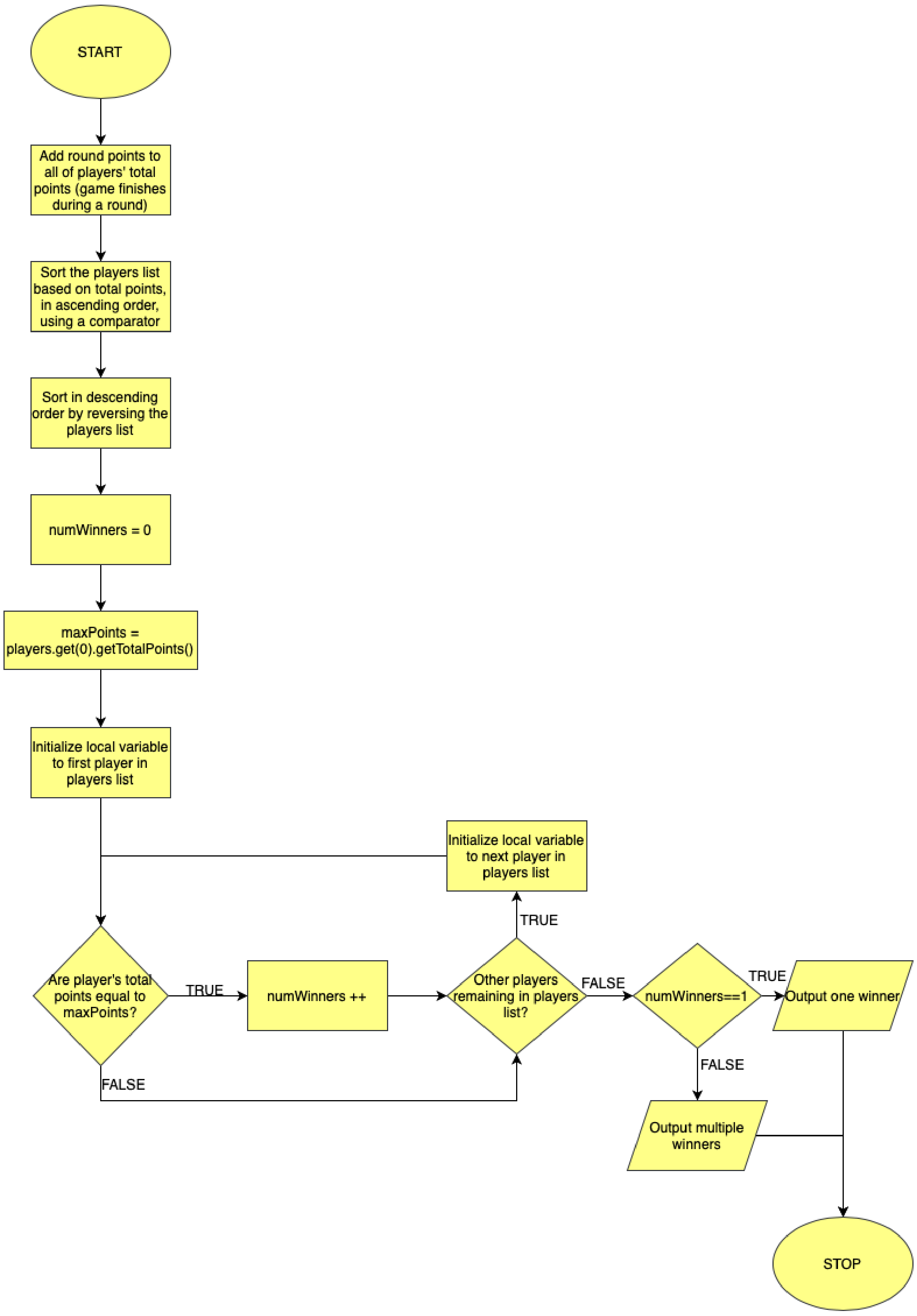

START
Add round points to all of players' total points (game finishes during a round)
Sort the players list based on total points, in ascending order, using a comparator
Sort in descending order by reversing the players list
numWinners = 0
maxPoints = players.get(0).getTotalPoints()
Initialize local variable to first player in players list
Initialize local variable to next player in players list
TRUE
Are player's total points equal to maxPoints?
TRUE
numWinners ++
Other players remaining in players list?
FALSE
numWinners==1
TRUE
Output one winner
FALSE
FALSE
Output multiple winners
STOP

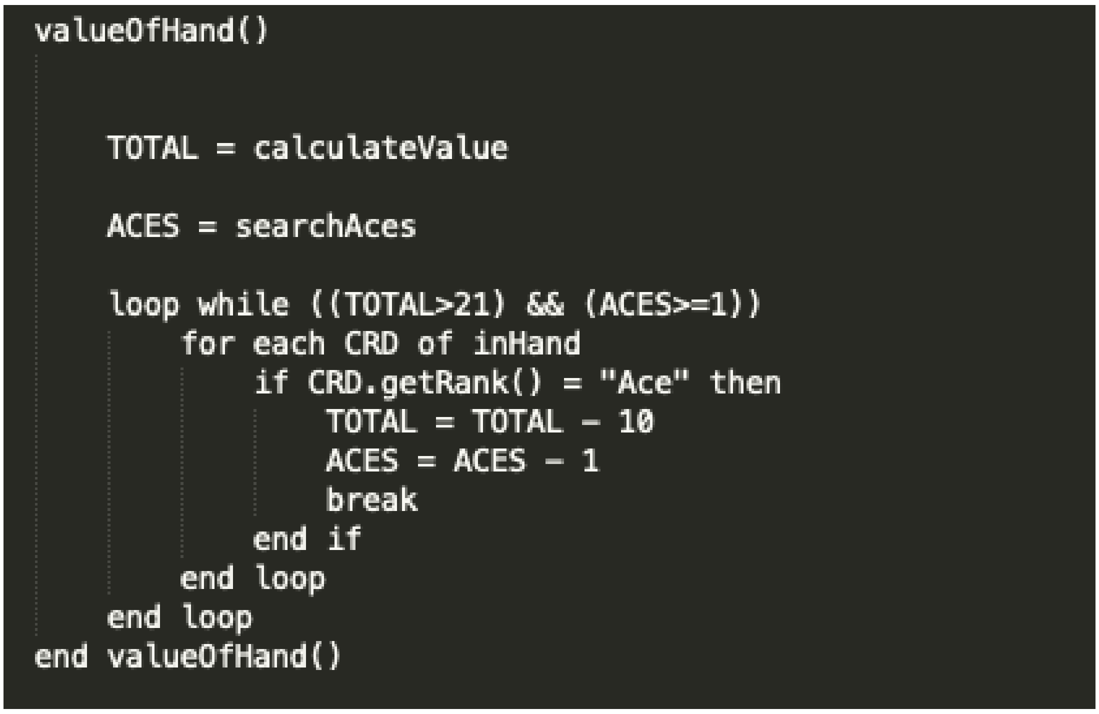
Figure 21: Determine the winner (in Game class)

Key Algorithms

Pseudocode is used to go to a next level of detail in the design of some key algorithms.

Name	valueOfHand() (**figure 15**)
Goal	Return the value of a person's hand.
Pre-conditions	1. There is a person (dealer or player) with a hand containing an ArrayList of cards inHand. 2. Method calculateValue returning the sum of the value of all cards in hand has been created. 3. Method searchAces returning the number of aces in hand has been created.
Post-condition	Return the value of a person's hand, having made any necessary adjustments to aces' values.

Table 3: valueOfHand() algorithm

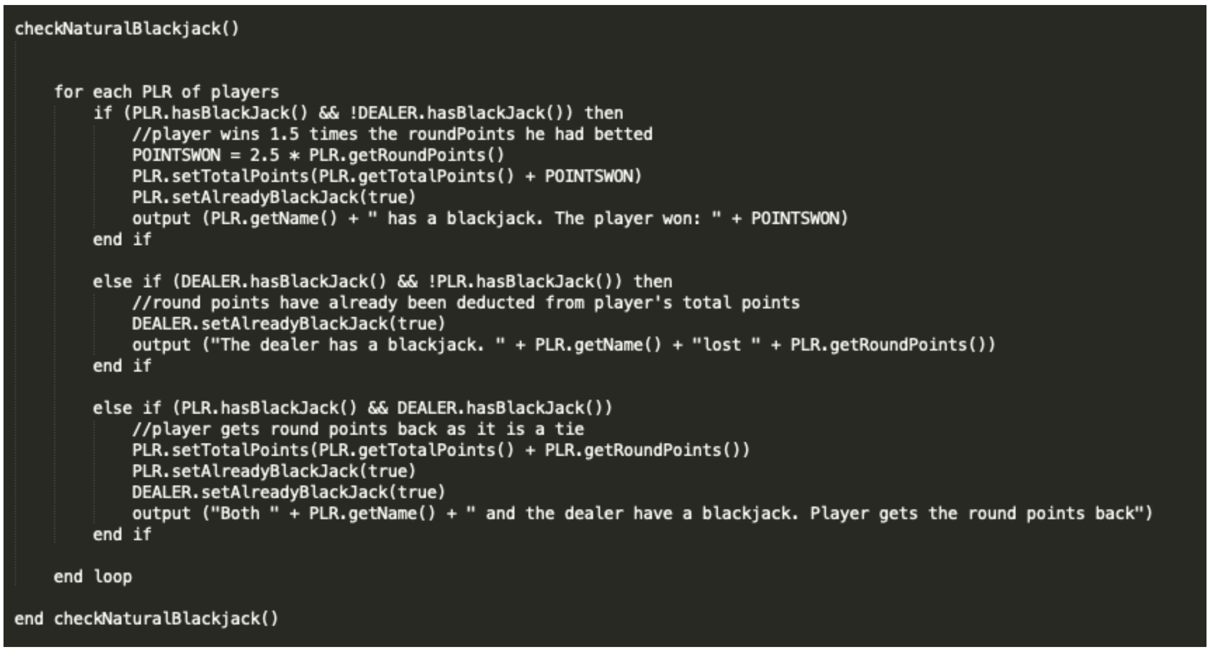


```
valueOfHand()


    TOTAL = calculateValue

    ACES = searchAces

    loop while ((TOTAL>21) && (ACES>=1))
        for each CRD of inHand
            if CRD.getRank() = "Ace" then
                TOTAL = TOTAL - 10
                ACES = ACES - 1
                break
            end if
        end loop
    end loop
end valueOfHand()
```

Figure 22: Pseudocode for valueOfHand()

Name	checkNaturalBlackjack() (**figure 16**)
Goal	Check whether player or dealer got a natural blackjack.
Pre-conditions	1. There is an ArrayList of instantiated players and a dealer. 2. Accessor and mutator methods for persons have been created.
Post-condition	Output whether player or dealer got a blackjack from their first two cards and points the player won or lost.

Table 4: checkNaturalBlackjack() algorithm

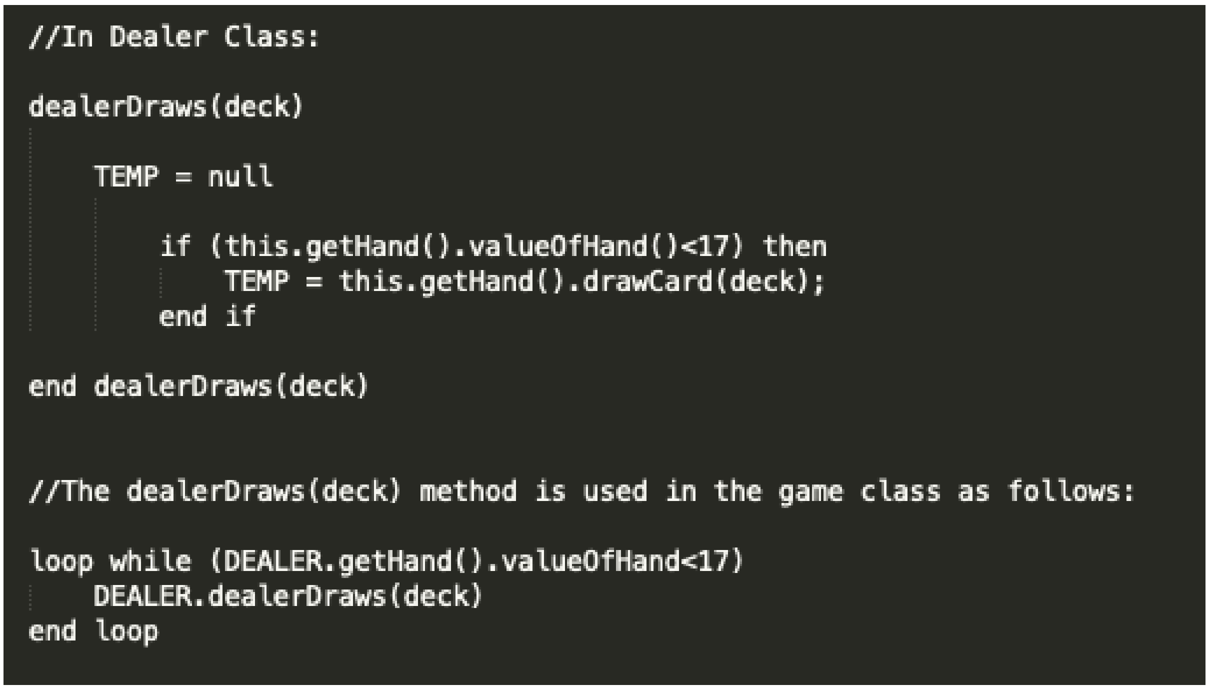


```
checkNaturalBlackjack()


    for each PLR of players
        if (PLR.hasBlackJack() && !DEALER.hasBlackJack()) then
            //player wins 1.5 times the roundPoints he had betted
            POINTSWON = 2.5 * PLR.getRoundPoints()
            PLR.setTotalPoints(PLR.getTotalPoints() + POINTSWON)
            PLR.setAlreadyBlackJack(true)
            output (PLR.getName() + " has a blackjack. The player won: " + POINTSWON)
        end if

        else if (DEALER.hasBlackJack() && !PLR.hasBlackJack()) then
            //round points have already been deducted from player's total points
            DEALER.setAlreadyBlackJack(true)
            output ("The dealer has a blackjack. " + PLR.getName() + "lost " + PLR.getRoundPoints())
        end if

        else if (PLR.hasBlackJack() && DEALER.hasBlackJack())
            //player gets round points back as it is a tie
            PLR.setTotalPoints(PLR.getTotalPoints() + PLR.getRoundPoints())
            PLR.setAlreadyBlackJack(true)
            DEALER.setAlreadyBlackJack(true)
            output ("Both " + PLR.getName() + " and the dealer have a blackjack. Player gets the round points back")
        end if

    end loop

end checkNaturalBlackjack()
```

Figure 23: Pseudocode for checkNaturalBlackjack()

Name	dealerDraws(deck) (**figure 19**)
Goal	Dealer can draw cards based on the rules of the game.
Pre-conditions	1. There is a dealer. 2. A deck has been instantiated and passed as parameter. 3. The method valueOfHand, calculating the value of a hand, has been created. 4. The method drawCard(deck), allowing persons to draw cards, has been created. 5. Accessor and mutator methods for persons have been created.
Post-condition	Cards are added to the dealer's hand while his hand is valued under 17.

Table 5: dealerDraws(deck) algorithm



```
//In Dealer Class:

dealerDraws(deck)

    TEMP = null

        if (this.getHand().valueOfHand()<17) then
            TEMP = this.getHand().drawCard(deck);
        end if

end dealerDraws(deck)


//The dealerDraws(deck) method is used in the game class as follows:

loop while (DEALER.getHand().valueOfHand<17)
    DEALER.dealerDraws(deck)
end loop
```

Figure 24: Pseudocode for dealerDraws(deck)