Program Summary:

The goal of our program is to create a virtual imitation of the casino card game, War. This program is aimed to serve as a method of entertainment, especially for card game and cassino enthusiasts. The program begins with taking user input to determine the number of decks in the game, either being 1 or 6 decks. Then, the user is asked for their name, which would be used in the creation of a player object instantiation to represent the user. After the user enters their wager for the round, the game begins with the dealer and player pulling cards. The player with the higher card collects the money wagered. However, if both players pull the same card, regardless of suit, the game enters a state of “war”. This is when both players must pull three unknown cards, and the values of the fourth cards pulled determine which player obtains the bets. In the case that these fourth cards are the same, the war continues until these cards are different. After a winner has been decided, another round repeats, following the same process as described. These rounds continue until the player runs out of money, of which they are initially given 500, or the player opts out of the game. One strength of this program is that it utilizes several programming constructs that allow for greater optimization and efficiency. These constructs include arrays, which reduce the number of lines required to create a similar program without arrays. Combined with for loops to traverse and manipulate the large amount of data organized in these arrays, this program has a greater simplicity and order, which allows for faster run-time. One weakness of this program is that it does not incorporate much user interactivity. Since this program is purely console-based, the target audience is likely to lose interest much quickly compared to a War program made with GUI. This leads to the future improvement, as the game could move from the console to an interactive application. Using Swing, users will be able to play a visual version of War, with buttons and images. Another improvement that can be made is that the game could add a way to play between friends, instead of with the program. This allows for greater enjoyment, as people will be able to play against each other.

```
// Casino War Game Pseudocode

// Main method
MAIN:

  // Initialize Scanner for user input
  input = new Scanner()

  // Get deck size from user
  PRINT "Enter number of decks (1, 2, 4, or 6):"
  deckSize = input.nextInt()

  // Validate and adjust deck size
  IF deckSize is 1 OR deckSize is even AND deckSize is less than or equal to 6
    deckSize = deckSize * 52
  ELSE
    deckSize = 52
  ENDIF

  PRINT "Deck size: " + deckSize

  // Clear scanner buffer
  input.nextLine()

  // Get player name
  PRINT "Enter your name:"
  playerName = input.nextLine()
```

```
// Create Player and Dealer objects
player = new Player1(playerName)
player.setCardDeck(deckSize)
dealer = new Dealer("Dealer")
dealer.setCardDeck(deckSize)

playAgain = TRUE

PRINT "Are you ready to play?"
input.nextLine()

PRINT "Welcome to Casino War!"

// Game loop
WHILE playAgain

  gameLoop = TRUE

  PRINT "Your balance: " + player.getBalance()
  PRINT "Enter bet amount:"
  playerBet = input.nextInt()
  dealerBet = playerBet

  WHILE gameLoop

    playerWins = FALSE
```

```
dealerWins = FALSE

input.nextLine() // Clear buffer

// Adjust bets if exceeding balance
IF playerBet > player.getBalance()
  PRINT "Bet exceeds balance. Bets split in half."
  playerBet = playerBet / 2
  dealerBet = playerBet
ELSE IF dealerBet > dealer.getBalance()
  PRINT "Bet exceeds dealer's balance. Bets split in half."
  dealerBet = dealerBet / 2
  playerBet = dealerBet
ENDIF

PRINT "Player ready, Dealer ready"
PRINT "Ready to flip your card?"
input.nextLine()

playerDeck = player.getCardDeck()
dealerDeck = dealer.getCardDeck()
PRINT "Cards in your deck: " + getLastIndex(playerDeck)

PRINT "You drew: " + playerDeck[0].getCardName()
PRINT "Dealer drew: " + dealerDeck[0].getCardName()

// Handle null cards
```

```
IF playerDeck[0] is NULL OR dealerDeck[0] is NULL
  moveElements(dealerDeck)
  moveElements(playerDeck)
ENDIF

// Compare cards and determine winner
IF playerDeck[0].getCardValue() > dealerDeck[0].getCardValue()
  PRINT "You win the round!"
  playerDeck[getFirstEmptyIndex(playerDeck)] = dealerDeck[0]
  playerDeck[getFirstEmptyIndex(playerDeck)] = playerDeck[0] // Add player's card
  moveElements(dealerDeck)
  moveElements(playerDeck)
ELSE IF playerDeck[0].getCardValue() == dealerDeck[0].getCardValue()
  PRINT "WAR"
  PRINT "Continue war?"
  input.nextLine()
  startWar(playerDeck, dealerDeck)
ELSE // Player loses
  PRINT "You lost the round!"
  dealerDeck[getFirstEmptyIndex(dealerDeck)] = playerDeck[0]
  dealerDeck[getFirstEmptyIndex(dealerDeck)] = dealerDeck[0] // Add dealer's card
  moveElements(dealerDeck)
  moveElements(playerDeck)
ENDIF

shuffleArray(playerDeck, dealerDeck)
```

```
  // Check for game over (one player has all cards)
  IF getLastIndex(playerDeck) == deckSize
    playerWins = TRUE
  ELSE IF getLastIndex(dealerDeck) == deckSize
    dealerWins = TRUE
  ENDIF

  PRINT "Continue?"
  input.nextLine()

  IF playerWins
    PRINT "You win the round!"
    PRINT "You gained $" + dealerBet
    player.addBalance(dealerBet)
    dealer.deductBalance(dealerBet)
    gameLoop = FALSE
  ELSE IF dealerWins
    PRINT "You lost the round! Dealer wins!"
    PRINT "You lost $" + playerBet
    player.deductBalance(playerBet)
    dealer.addBalance(playerBet)
    gameLoop = FALSE
  ENDIF

ENDWHILE // gameLoop

// Check if game can continue
```

```
IF player.getBalance() > 0 AND dealer.getBalance() > 0
  PRINT "Play another round? (Yes/No)"
  redo = input.nextLine()
  IF redo is "Yes" (case-insensitive)
    playAgain = TRUE
    player.setCardDeck(deckSize)
    dealer.setCardDeck(deckSize)
  ELSE
    playAgain = FALSE
    PRINT "Game ended. Results:"
    // Determine overall winner
    IF player.getBalance() > dealer.getBalance()
      PRINT "You won the game!"
    ELSE IF dealer.getBalance() == player.getBalance()
      PRINT "Game is a tie!"
    ELSE
      PRINT "You lost the game!"
    ENDIF
  ENDIF
ELSE // One player out of money
  playAgain = FALSE
  PRINT "Game ended. Results:"
  // Determine overall winner (same as above)
  IF player.getBalance() > dealer.getBalance()
      PRINT "You won the game!"
    ELSE IF dealer.getBalance() == player.getBalance()
      PRINT "Game is a tie!"
    ELSE
```

```
        PRINT "You lost the game!"
      ENDIF
    ENDIF

  ENDWHILE // playAgain

ENDMAIN

// Helper functions (pseudocode descriptions)

FUNCTION moveElements(arr): // Shift elements left
  FOR i from 1 to arr.length -1
    arr[i-1] = arr[i]
  ENDFOR
ENDFUNCTION

FUNCTION getFirstEmptyIndex(arr): // Find first null element
  index = 0
  FOR i from 1 to arr.length - 1
    IF arr[i] is NULL AND arr[i-1] is NOT NULL
      index = i
      BREAK // Exit loop
    ENDIF
  ENDFOR
  RETURN index
ENDFUNCTION
```

```
FUNCTION getLastIndex(arr): // Find last non-null element
  index = 0
  FOR i from 1 to arr.length - 1
    IF arr[i] is NOT NULL
      index = i + 1
    ENDIF
  ENDFOR
  RETURN index
ENDFUNCTION

FUNCTION shuffleArray(arr1, arr2): // Shuffle both arrays
  // Shuffle arr1 (similar logic for arr2)
  FOR i from 0 to getLastIndex(arr1) - 1
    randomNumber = random number between 0 and getLastIndex(arr1) - 1
    temp = arr1[i]
    arr1[i] = arr1[randomNumber]
    arr1[randomNumber] = temp
  ENDFOR
  // Shuffle arr2 (same logic as arr1)
ENDFUNCTION

FUNCTION startWar(arr1, arr2): // War logic
  index = 3
  war = TRUE
  WHILE war
    IF arr1[index].getCardValue() > arr2[index].getCardValue()
      // Player wins war round
```

```
      // ... (move cards to player's deck)
      war = FALSE
    ELSE IF arr1[index].getCardValue() < arr2[index].getCardValue()
      // Player loses war round
      // ... (move cards to dealer's deck)
      war = FALSE
    ELSE
      // Tie in war round
      index = index + 4
    ENDIF
  ENDWHILE
ENDFUNCTION

// DealerInterface interface
INTERFACE DealerInterface
  FUNCTION getCardDeck() RETURNS Card array
  FUNCTION setCardDeck(deckSize)
END INTERFACE

// Dealer class (extends Person, implements DealerInterface)
CLASS Dealer EXTENDS Person IMPLEMENTS DealerInterface

  // Private cardDeck (Card array)
  cardDeck = new Card array

  // Constructor: Dealer(dealerName)
  CONSTRUCTOR(dealerName)
```

```
    CALL super(dealerName) // Call Person constructor
  END CONSTRUCTOR

  // getCardDeck() method
  FUNCTION getCardDeck() RETURNS Card array
    RETURN cardDeck
  END FUNCTION

  // setCardDeck(deckSize) method
  FUNCTION setCardDeck(deckSize)
    cardDeck = new Card array of size deckSize

    // Initialize cards in the deck
    FOR i from 0 to deckSize/2 - 1
      cardDeck[i] = new Card()
      cardDeck[i].setCardValue()
    ENDFOR
  END FUNCTION

END CLASS


// Person class (parent class for Player and Dealer)
CLASS Person

  // Private instance variables
  name = "" (String)
```

```
totalBalance = 500 (Integer)

// Constructor: Person(theName)
CONSTRUCTOR(theName)
  name = theName
END CONSTRUCTOR

// Getter methods

FUNCTION getBalance() RETURNS Integer
  RETURN totalBalance
END FUNCTION

FUNCTION getName() RETURNS String
  RETURN name
END FUNCTION

// Setter methods

FUNCTION deductBalance(balance)  // balance is an Integer
  totalBalance = totalBalance - balance
END FUNCTION

FUNCTION addBalance(balance) // balance is an Integer
  totalBalance = totalBalance + balance
END FUNCTION
```

```
END CLASS

// PlayerInterface interface
INTERFACE PlayerInterface
  FUNCTION getCardDeck() RETURNS Card array
  FUNCTION setCardDeck(deckSize)
END INTERFACE

// Player1 class (extends Person, implements PlayerInterface)
CLASS Player1 EXTENDS Person IMPLEMENTS PlayerInterface

  // Private cardDeck (Card array)
  cardDeck = new Card array

  // Constructor: Player1(playerName)
  CONSTRUCTOR(playerName)
    CALL super(playerName) // Call Person constructor
  END CONSTRUCTOR

  // getCardDeck() method
  FUNCTION getCardDeck() RETURNS Card array
    RETURN cardDeck
  END FUNCTION

  // setCardDeck(deckSize) method
  FUNCTION setCardDeck(deckSize)
    cardDeck = new Card array of size deckSize
```

```
    // Initialize cards in the deck
    FOR i from 0 to deckSize/2 - 1
      cardDeck[i] = new Card()
      cardDeck[i].setCardValue()
    ENDFOR
  END FUNCTION

// CardInterface interface
INTERFACE CardInterface
  FUNCTION getCardValue() RETURNS Integer
  FUNCTION getCardName() RETURNS String
END INTERFACE

// Card class (implements CardInterface)
CLASS Card IMPLEMENTS CardInterface

  // Instance variables
  theCardNumber = 0 (Integer)
  CardName = "" (String)

  // Setter methods

  FUNCTION setCardValue()
    cardNumber = random integer between 2 and 15 (inclusive)
    theCardNumber = cardNumber
    CALL setCardName(cardNumber)
```

```
END FUNCTION

FUNCTION setCardName(cardNumber)
  SWITCH cardNumber
    CASE 2: CardName = "Two"
    CASE 3: CardName = "Three"
    CASE 4: CardName = "Four"
    CASE 5: CardName = "Five"
    CASE 6: CardName = "Six"
    CASE 7: CardName = "Seven"
    CASE 8: CardName = "Eight"
    CASE 9: CardName = "Nine"
    CASE 10: CardName = "Ten"
    CASE 11: CardName = "Jack"
    CASE 12: CardName = "Queen"
    CASE 13: CardName = "King"
    CASE 14: CardName = "Ace"
    CASE 15: CardName = "Joker"
    DEFAULT:
      theCardNumber = 2
      CardName = "Two"
  END SWITCH
END FUNCTION

// Getter methods

FUNCTION getCardValue() RETURNS Integer
```

```
    RETURN theCardNumber
  END FUNCTION

  FUNCTION getCardName() RETURNS String
    RETURN CardName
  END FUNCTION

END CLASS
```