Candidate Numbers: 024575 and 032116

Production Code

Classes:

Card:

- Purpose = represents a card in the game with a numeric value.
- Attributes = value – an integer representing the card's value
- Methods =
    - Card(int value) – constructor that initializes a card's value and ensures it's not negative
    - getValue() – a getter method to retrieve the card's value

CardGame

- Purpose = orchestrates the game logic, including setup and execution
- Attributes =
    - numberOfPlayers – number of players in the game, and so the number of decks
    - pack – list of all cards loaded from a file
    - decks – list of decks used in the game
    - players – list of player objects
    - gameWon – atomic flag indicating the end of the game
        - atomic variables are used so; to allow multiple threads to access and modify shared variables safely and so synchronization overhead is minimized
    - winner – atomic flag which allows the threads to share the winner (an integer) to make the output file correctly
    - threadPool – executor servie managing the player's threads
- Methods =
    - loadPack(String packPath) – reads the card values from a file and initializes the pack
    - validatePack() – ensures that the pack contains the correct number of cards
    - initializeDecks() – creates the required number of decks
    - initializePlayers() – sets up players with references to their respective draw and discard decks
    - dealCards() – distributes cards to players and decks
    - writeDecksToFile() – output the final state of each deck to a file
    - play() – manages the game's execution; starts player threads, monitors for a winner, stops players and writes results to files

Deck

- Purpose = represents a deck of cards, functioning as both a draw and discard pile for players
- Attributes =
    - deckNumber – an identifier for the deck
    - cards - a double ended queue storing the cards in the deck
    - lock – a reentrant lock ensuring thread-safe access to the deck
- Methods =
    - drawCard() – removes and returns the first card in the deck, ensuring thread safety

  - addCardToTop(card) – add a card to the top of the deck, using a lock to ensure the operation is thread-safe, and avoids resource contention issues
  - addCard(Card card) – adds a card to the end of the deck, also thread-safe
  - getCurrentDeck() – returns a snapshot of the current cards in the deck
  - getDeckNumber() – retrieves the deck's unique identifier

Player

- Purpose = models a game participant, handling their interactions with the game
- Attributes =
  - hand – a list representing the player's current hand of cards
  - drawDeck – the deck from which the player draws cards
  - discardDeck – the deck to which the player discards cards
  - gameWon – an atomic Boolean indicating whether the game has ended
  - winner – an atomic integer indicating who won
  - output – a print write for logging the player's actions
  - alive – a flag determining if the player is active
  - playerNumber – the player's unique identifier
- Methods =
  - drawCard(Card card) – adds a card to the player's hand
  - discardCard(Card card) – removes a card from the player's hand and places it in the discard deck
  - getHand() – retrieves the current hand of cards from the player's hand
  - chooseDiscardCard() – selects a card to discard, it prefers cards not matching the player's number, otherwise it picks randomly
  - hasWinningHand() – checks if the player's hand contains four cards of the same value
  - handToString() – converts the player's hand to a string for logging purposes
  - run() – implements the player's game loop which is to: draw a card, discard a card, log the actions and check for a winning condition
  - stopPlaying() – terminates the player's activity

Performance Issues

The use of re-entrant lock in the Deck class adds slight overhead due to synchronization but it is necessary for concurrent access.

Copying the deck in getCurrentDeck() in the Deck class can be expensive if the deck is large, especially in high-frequency calls.

In the Player class, the frequent logging to files of output.flush might slow down execution, particularly if the players are active.

The thread sleep of 10ms in the Player class reduces CPU usage but might introduce some small delays.

In the CardGame class, the file I/O operations may become bottlenecks because of logging and writing deck states.

The game relies heavily on thread coordination, so it requires a careful management of shared resources.

Candidate Numbers: 024575 and 032116

Tests – using JUnit 4x Framework

Classes:

AllTests:

- Purpose: acts as a test suite container that combines multiple Junit test classes for batch execution
- Annotations:
  - RunWith – specifies that the test class is a suite
  - SuiteClasses – lists the suite's test classes: PlayerTest, CardGameTest, DeckTest and CardTest

CardGameTest:

- Purpose: tests the functionality of the CardGame class
- Attributes:
  - TemporaryFolder – JUnit rule for creating temporary files
  - cardGame – instance of the CardGame class being tested
- Utility Methods:
  - cleanFolder (@Before) – deletes residual test-generated files to access private fields of CardGame
  - getDeckField – uses Java Reflection to access private fields of CardGame
  - getPlayersField – uses Java Reflection to access private fields of CardGame
  - setUpGame – prepares a CardGame instance with temporary pack data for testing
  - deleteVariables (@After) – resets attributes and cleans temporary data post-tests
- Test Methods:
  - testValidPack – verifies proper initialization with a valid pack
    - True – game initializes successfully
    - False – fails if the pack file is invalid or incomplete
  - testInvalidPack – ensures exceptions are throw for invalid packs and tests edge cases were the pack file is too small (less than the required cards) or empty
    - True – exception is thrown
    - False – exception is not thrown for valid inputs
  - testNonExistentFile – tests that creatinga game with a non-existent file throws an IOException
    - True – IOException is thrown as expected
    - False – no exception thrown for invalid files means game can run unexpectedly
  - testValidPackForAllPlayerCounts – checks correct initialization of players and decks for varying player counts
    - True – each player and deck are initializes correctly
    - False – any mismatch between expected and actual counts of players or decks
  - testDealCards – ensures each player and deck starts with the correct number of cards

        - True – all hands and decks have exactly n (in this case 4) cards
        - False - any mismatch in card counts
    - testInitializePlayers – verifies proper setup of player's draw and discard decks
        - True – draw and discard decks are assigned correctly
        - False – any misassignment breaks game logic
    - testWriteDecksToFile – ensures decks are written to files as expected
        - True – files are created and populated correctly
        - False – missing or empty files
    - testGameEnds – confirms the game terminates when a player wins
        - True – threads terminate when a winning hand exists
        - False – threads remain alive or game fails to end

CardTest:

- Purpose: tests the functionality of the Card class
- Test Methods:
    - testCreation – verifies that cards are created with the correct value
        - True – card value matches the input
        - False – card value is incorrect
    - testNegative – ensures an exception is thrown for negative card values
        - True – Illegal argument exception is thrown for negative values
        - False – cards with invalid values can and have been created
    - testEquality – validates proper equality and inequality of Card's instances
        - True – equality check works correctly for matching and mismatched cards
        - False – incorrect comparisons

DeckTest:

- Purpose: tests the functionality of the Deck class
- Attributes: deck – instance of the Deck class being tested
- Utility Methods:
    - setUpTest (@Before) – initializes a test deck instance
    - cleanUpTest (@After) – cleans up deck instance after tests
- Test Methods:
    - testAddAndDrawCard – checks the ability to add and draw cards correctly
        - True – drawn card matches the added card
        - False – mismatched cards or the inability to draw cards
    - testGetCurrentDeck – verifies the deck's current state is accurately returned
        - True – current deck matches added cards
        - False – any discrepancy in card order or count
    - testGetDeckNumber – confirms the deck number matches the initialization value
        - True – correct deck number is returned
        - False – deck number is incorrect or uninitialized

  - testAddCardOrder – tests the behavior of the addCard and addCardToTop methods for order integrity
    - True – cards are in the expected order
    - False – the card order is incorrect

PlayerTest:

- Purpose: tests the functionality of the Player class
- Attributes:
  - Player – instance of the Player class being tested
  - drawDeck – deck for the player's draw actions
  - discardDeck – deck for the player's discard actions
  - gameWon – shared atomic Boolean flag indicating if the game is won
  - winner – shared atomic Boolean flag used to track the winner
- Utility Methods:
  - setUpTest (@Before) – initializes the Player class with drawDeck, discardDeck, gameWon and winner, it's attributes
  - cleanUpTest (@After) – cleans up all attributes
- Test Methods:
  - testDrawAndDiscard – ensures players can draw and discard cards properly
    - True – player's hand reflects the accurate card count and order
    - False – any discrepancy in hand management
  - testWinningHand – validates winning condition detection with a hand of identical cards
    - True – winning hand detected correctly
    - False – winning hand is not detected or misdetected
  - testNonWinningHand – ensures non-winning conditions are identified
    - True – empty hand is non-winning
    - False – empty hand is incorrectly deemed winning
  - testChooseDiscardCard – verifies correct discard card selection based on player logic with Java Reflection
    - True – the correct card is chosen for discard
    - False – the incorrect card is retained or discarded
  - testStopPlaying – ensure the Player threads stop when require
    - True – alive is toggled to False after stopping
    - False – the player class remains active after stopping
  - testOutputGeneration – confirms proper generation of output files for player actions
    - True – files are created and populated correctly
    - False – there are missing or empty files

Development Log

Date	Duration	Pilot	Driver
22/10	2 hours	032116	024575
24/10	3 hours	024575	032116
28/10	2 hours	032116	024575
2/11	2 hours	024575	032116
12/11	2 hours	032116	024575
14/11	2 hours	024575	032116
17/11	2 hours	032116	024575
21/11	2 hours	024575	032116
24/11	2 hours	032116	024575
28/11	2 hours	024575	032116
3/12	2 hours	032116	024575
5/12	4 hours	024575	032116
7/12	4 hours	032116	024575
9/12	4 hours	024575	032116