***Cover Page***

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Student Number | Candidate Number | Allocation |
| Benjamin Finch | 710005114 |  | 50 |
| Patryk Biegalski | 700037440 |  | 50 |

***Development Log***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date** | **Time** | **Driver** | **Observer** | **Ben’s Signature** | **Patryk’s Signature** |
| 27/10/22 | 12:25 – 13:25 | Ben | Patryk |  |  |
| 28/10/22 | 17:30 – 20:00 | Patryk | Ben |  |  |
| 05/11/22 | 13:00-15:00 | Patryk | Ben |  |  |
| 05/11/22 | 16:00-18:00 | Ben | Patryk |  |  |
| 14/11/22 | 16:30-19:00 | Ben | Patryk |  |  |
| 15/11/22 | 13:30-14:30 | Patryk | Ben |  |  |
| 15/11/22 | 15:00-16:30 | Ben | Patryk |  |  |
| 16/11/22 | 12:45-13:45 | Patryk | Ben |  |  |
| 18/11/22 | 16:00-18:00 | Patryk | Ben |  |  |
| 19/11/22 | 14:00-16:00 | Patryk | Ben |  |  |
| 19/11/22 | 16:30-18:30 | Ben | Patryk |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Ben: 9

Patryk: 10.5

## Documentation – Source Code

# Card Class

Our Card class is a class which stores an integer value. The reason we did not use integers without a wrapper class is to protect our code from using the values in incorrect ways. In the game, cards cannot be added, subtracted etc., and should only be stored in data structures intended to store them, such as the “pack” variable in CardGame or the “hand” variable in Player. These data structures are BlockingQueues that require us to override the CompareTo method in order to be able to compare cards when players decide if a card is of their preferred number. In addition, if we were to expand on the game and add suits to the cards, the Card class makes it easier to do so.

# Player Class

Design

Players all have an identifying integer Id and a hand of 4 cards (of type BlockingQueue<Card>). Players will carry out three actions, checking if they have won, drawing from their draw deck and discarding to their discard deck. Draw and discard decks are calculated using the Ids of the players and decks.

The player draws from the deck with a matching Id and discards to the deck with the Id one more than their own; except for the last player, who discards to the first deck. In order to facilitate asynchronous interaction with each other, the Player class contains a static ArrayList<Player> attribute which contains all created player objects.

The players will also carry out an action of ‘exiting’, which will happen if they win or are notified that another player has won. This action will comprise of writing their final hand to a file, officially admitting their loss and then their threads will move to the terminated state.

Threading

Our player class implements the Runnable interface in order to make use of Java’s Thread class. Keeping with the specification, we designed our code to be as thread safe as possible. We did this by using a BlockingQueue to store cards, as it is already a thread-safe data structure. We also implemented try-catch blocks to catch interrupt exceptions so that players would stop playing if they are interrupted.

Hand

The players hand should hold exactly 4 cards at any time, we achieved this by synchronizing the drawCard() and discardCard() functions. When discarding, the player keeps any cards which match their Id, and when all cards match, the player updates the static “winner” attribute with their player Id. As part of this, players should make sure that no cards except their preference stay in their hand forever, to this end, the BlockingQueue enables the players to automatically cycle their cards so that all necessary cards will (at most four turns after being drawn) be discarded.

When the player discards a card, they check the front of their hand. If the card if their preferred type, they add it to the back of their queue. If the card is not their preferred type, the add it to their discard deck and draw a new card. In theory, this could lead to a situation where a player checks their already winning hand for a card to discard. However discardCard() can only be executed from run() if the hand is not a winning hand, therefore this situation is unlikely, if not impossible to occur.

Output

The Player class should write updates to an external text file to document the game running. To achieve this we used the BufferedWriter class. BufferedWriter writes text to a character-output stream, buffering characters in order to provide for the writing of arrays, lists and strings. BufferedWriter also overwrites duplicate files, which simplifies repeated testing substantially.

# CardDeck Class

Design

The CardDeck class is a wrapper class for a BlockingQueue<Card> data structure, that also outputs data to an external text file. We decided to make the decks a separate class because it would both make it easier to make changes in CardGame and Player, without affecting the behaviour of the decks and make the system easier to visualise when implementing the game loop.

CardDeck uses the same BufferedWriter class as Player to output the contents of the decks to text files when the game has ended.

Threading

When the card decks are created in the CardGame class they are created as AtomicReference type, so that the draw and discard methods of that class are considered atomic and cannot be interrupted part way through their execution.

# CardGame Class

CardGame uses a main() function to play the card game, then uses extra functions to simplify main(), the goal being to make main() look as self-explanatory as possible by outsourcing any sections of complex code to their own small, self-contained (and easily documented) functions.

The main() function asks for number of players (n E N) and a list of card values (the size of which should be 8n) before generating the required number of Player and CardDeck objects. It then distributes Card objects to each Player and CardDeck and starts all the Player threads. Note that, because decks are assigned to players during player initialisation, CardDecks are generated first; however, players are still dealt to first in keeping with the specification.

By starting the Player threads as the last step of main(), main() does not execute any code while the Player threads are running. This is intentional as it reduces the amount of code running simultaneously and therefore maintains orthogonality, since any change in the Player’s run() method do not affect the execution of main() and changes in main() (other than any which prevent Player threads starting) do not affect the execution of run().

## Documentation – Testing in Junit 4

# TestCard Class

|  |  |  |
| --- | --- | --- |
| **Test Name** | **Query** | **Purpose** |
| testCompareEqual() | Do cards with equal values return 1 when compared? | This test confirms that the code is returning the correct value whenever cards are being compared. And that we as developers fully understand the compareTo() function. |
| testCompareUnequal() | Do cards with unequal values return 0 when compared? | This test confirms that the code is returning the correct value whenever cards are being compared. And that we as developers fully understand the compareTo() function. |

# TestPlayer Class

Player card drawing is implicitly tested during other tests.

\*The purpose of this test is to repeat the previous under different conditions.

|  |  |  |
| --- | --- | --- |
| **Test Name** | **Query** | **Purpose** |
| testConstructorDraw() | Do players choose the correct draw pile? | We wanted to check that players would choose the correct deck for to draw from. |
| testConstructorDiscard() | Do players choose the correct discard pile? | We wanted to check that players would choose the correct deck for to discard to. This test forced us to change how discard decks are assigned to players. |
| testDiscard() | Do players discard the first card in their hand if it isn't the preferred card? | The discardCard() method requires that the first card be discarded. |
| testDiscard2() |  | \* |
| testDiscardCycle() | Do players discard the second card in their hand if the first is the preferred card? | This test is used to make sure that players keep their preferred card even if it is at the start of their hand. |
| testDiscardCycle2() |  | \* |
| testDiscardFullCycle() | Do players discard the last card in their hand if the first three are the preferred card? | This test is used to make sure that players keep their preferred cards even if only the last card in their hand is not preferred. |
| testDiscardFullCycle2() |  | \* |
| testWinningAtStart() | Do players declare victory if they start the game with a winning hand? | This test makes sure that the player will not play the game if they start with a winning hand and that they declare their victory. |
| testWinningAtStart2() | Do players declare victory if they start the game with a winning hand that is not their preferred number? | This test makes sure that a player can win at the start of the game with a hand that is not made up of their preferred cards. |
| testWinningAtStart3() |  | \* |
| testWinningAtStart4() |  | \* |
| testWinningAfterStart() | Do players declare victory if they draw a card and get a winning hand? | The purpose of this test is to check that when a player draws into a winning hand, the thread checks for the win condition, sets themselves as the winner and stops drawing more cards. |
| testWinningAfterStart2() |  | \* |
| testLosingAtStart() | Do players concede defeat if another starts with a winning hand? | Checks that if there is a winner when a player thread starts running, the thread does not attempt to draw cards and immediately admits defeat. |
| testLosingAtStart2() |  | \* |
| testLosingAfterStart() | Do players concede defeat if another obtains a winning hand? | This tests whether a thread that is running will stop running when another player thread has won. |
| testLosingAfterStart2() |  | \* |

# TestCardGame Class

|  |  |  |
| --- | --- | --- |
| **Test Name** | **Query** | **Purpose** |
| testMainEmptyPack() | Does cardGame refuse an empty pack? | This tests that the game does not start if the given card pack does not contain any data. |
| testMainFakePack() | Does cardGame refuse a non-existant pack? | This tests that the game does not start if the given card pack does not exist. |
| testMainSmallPack() | Does cardGame refuse a pack with too few numbers? | This tests that the game does not start if the given card pack does not contain enough cards for the given number of players. |
| testMainLargePack() | Does cardGame only take the require numbers from a list with too many numbers? | This tests that the game does start if the given card pack contains more cards than required, but only takes the required number of cards from that pack file. |
| testMainInvalidLinesPack() | Does cardGame refuse a pack with invalid data? | This tests that the game does not start if the given card pack contains data that is not integers stored in the expected format. |
| testMainZeroPlayers() | Does cardGame refuse to play with zero players? | This tests that the game does not start if the given number of players is 0. |
| testMainNegativePlayers() | Does cardGame refuse to play with negative players? | This tests that the game does not start if the given number of players is negative. |
| testGetPlayers() | Does the method ask for an input and read it correctly? | This tests that the method can accept a terminal input and return it as an Integer. |
| testGetPack() | Does the method ask for an input and read it correctly? | This tests that the method can accept a terminal input and return it as a String. |
| testGeneratePlayerThreads() | Does the method create the given number of player threads and save them? | This tests that the method creates the player threads and asserts that they are saved to a static variable. |
| testGenerateDecks() | Does the method create the given number of decks and save them? | This tests that the method creates the decks and asserts that they are saved to a static variable. |
| testDealToPlayers() | Does the method deal the right number of cards in the right order? | This tests whether the method deals cards from a pack to the player threads in the expected order and whether the players get the right number of cards. |
| testDealToDecks() | Does the method deal the right number of cards in the right order? | This tests whether the method deals cards from a pack to the player decks in the expected order and whether the decks get the right number of cards. |