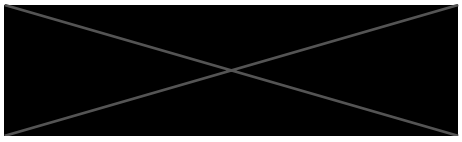
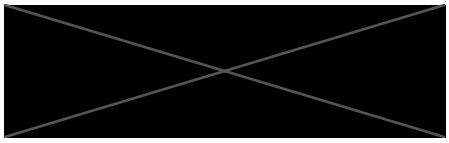
ECM2414 – Software Development

Student Number	[redacted]	[redacted]
Contribution	50	50

Development Log

Date	Time	Duration	720056545	720058039	What we worked on
6th November	9:30 pm	1 hr 30 min	Driver	Driver	OO Design
13th November	9:30 pm	1 hr 45 min	Driver	Observer	Methods & Classes
20th November	9:30 pm	1 hr 15 min	Observer	Driver	Methods & Classes
27th November	9:30 pm	45 min	Driver	Driver	Redid OO Design
1st December	5:00 pm	2 hr 30 min	Observer	Driver	Methods & Classes
4th December	9:30 pm	2 hr 40 min	Driver	Observer	Methods; Classes & Error Handling
5th December	10:00 pm	1 hr	Observer	Driver	Error Handling
7th December	2:30 pm	3 hr	Driver	Driver	Development
8th December	2:30 pm	2 hr 45 min	Driver	Driver	Development
9th December	1:00 am	3 hr	Driver	Observer	Testing & Error Handling
9th December	9:00 pm	1 hr 30 min	Observer	Driver	Report

Testing classes were created simultaneously with the respective classes.

signatures

Design Choice (Production Code)

- Modularity & Scalability

  There are separate classes for each task
  - Card: sets and gets each card's value
  - CardDeck: generates a list of Cards which are later used in the game for adding, drawing & discarding cards from a deck to a player and then to the next deck.
  - Player: contains information regarding a player such as ID, preferred value, left deck, right deck, wining condition in the start of the game & cards in hand during the game. It also handles thread safe card dealing.
  - CardGame: The main class. It manages various variables of the game like validating the pack given by the user, the number of players, the output files, dealing the cards in a round robin fashion & thread management for each player.

- Libraries
  - java.utill: used ArrayList, Collections, List, LinkedList, Queue, Map for collecting/sorting data
  - java.utill.Random: used to shuffle the pack provided by the user
  - java.util.concurrent.locks: used Lock & ReentrantLock to avoid data corruption/overlay
  - java.io: BufferedWriter, FileReader: for reading the pack and generating the output files
  - java.io.IOException: for error handling

- Error Handling

  Created multiple custom exception classes for neat and extensive error handling.
  - CardDeckException: not allowing null cards to be added to the deck
  - PlayerException: cannot receive null cards and player ID cannot be negative
  - CardGameException: positive integer for number of players; invalid & missing pack; players, decks, or card values cannot be null or negative
  - Using java.util.concurrent.locks to avoid data corruption
  - Earlier when we were running the code multiple times, we would have to delete the older version of the output files ourselves. Now the system clears all the files in the output directory before it starts the game.
  - There are multiple cases where there is a possibility that no one wins which would lead to an infinite loop/game, we fixed this issue by validating a pack before the game starts. The program validates that there is at least one set of cards in the pack provided leading to a winning condition and some other conditions.

Design Choice (Testing Code)

Deviced multiple testing classes using Junit5 for efficiently examining the methods

- CardTest: checks the reliability of the getter and setter
- CardDeckTest: making sure the decks are made correctly with no null cards
- PlayerTest: testing if a player's information is stored accurately
- CardGameTest: ensuring the game runs as expected and errors are raised when necessary

Why we chose Junit5?

- Modular Architecture:
  There is no need to make a jar file for running the test classes
- More Powerful Assertions:
  Better exception handling and grouping of assertions which also improves readability and precision over older JUnit methods.
- Dynamic and Flexible:
  Features like @TestFactory made tests for iterative logic like testing multiple card values more concise.
- Backward Compatibility:
  JUnit 5 supports JUnit 4 tests through the JUnit Vintage engine, so if we add older tests in the project, they will still run without modification.

Performance Issues

- The system empties any file that is inside the output folder before starting the game, but ideally it should delete all the files in the output folder before starting the game.
- When there are more than 5 players it is a possibility for some games ending with one player going over the number of moves made as compared to the rest of the players.
- Sometimes more than one player can win the game due to multi-threading.
- Unable to run test classes after making the jar file, but the test classes have been added to "cardsTest.zip" for reference.