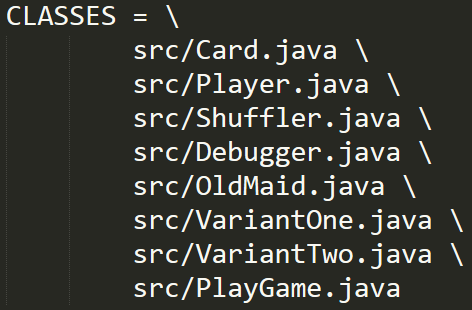
**Homework #3 OldMaid Extensions**

B01902080 資工四 王于青

1. **My design of class structures and reasons**

I use the structures similar to **hw2** as shown on the right.

Card holds the information of each card (e.g., suit, face…).

Player holds a hand of cards and can do some actions with cards (e.g., dropCards()).

Shuffler can shuffle any input array user specified.

Therefore, I shuffle the IDs of each player instead of a deck of cards, and assign each card to player based on the shuffled array. For example:

Input array: [0, 0, …, 0, 1, …, 1, …, 2, …, 2, 3, …, 3]

Shuffled array: [1, 3, 1, 0, 0, 2, …]

Deal cards: card[0] 🡪 player1, card[1] 🡪 player3, card[2] 🡪 player1, …

In this way, there is no need to sort the players’ hands after dealing cards and would save much time.

OldMaid is the original(parent) class, and VariantOne, VariantTwo are the extended(child) classes.

OldMaid is an abstract class and defines two abstract methods: initPlayers() and initCards().

Since the main differences between my variants are:

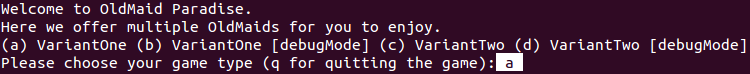
1. The components of cards (or the definition of JOKER).
2. The rules of dropping cards

Therefore, I force each variant to define their cards and player actions by implementing the methods.

The rest of game actions are all in OldMaid.

PlayGame is the main class to start the game, it contains multiple game types for user to choose.

Debugger is used to trace the correctness of the program.

1. **How to play the game**
2. In the correct folder (i.e., hw3/), enter “make” and “make run” to start the game.
3. Choose your game type:

VariantOne:

1. There are two JOKERS, R0 and B0, in each deck of cards.
2. A valid pair is defined as two cards with the same rank and color.

That is, [D3, H3], [C6, S6] are pairs, while [H7, HK], [S5, D5] are not.

VariantTwo:

1. A random card is removed from 52 cards.

The last card in the loser’s hand would be the JOKER.

1. A valid pair is defined as two cards with the same rank (same as hw1).
2. **Methods used to test the correctness of my program**

I implement the class Debugger.java to check the correctness of my program.

Debugger holds a cards-recycle array CARDS to trace the status of each card.

CARD[i] has the initial value 0, which means the card is currently in one of the players’ hand.

Every time when a player drops a pair, we can use debugger.addCards(pair) to change the value of CARDS[i] to 1, which means the card is recycled by debugger.

Debugger can check the following things:

1. The correctness of shuffling and dealing cards.

After dealing cards, I add all of the players’ hand to debugger and check if the values in CARDS are all changed to 1. If yes, it guarantees that cards are shuffled and dealt to players correctly.

1. The correctness of dropping existing pairs before drawing cards.

After dropping pairs, I initialize debugger.CARDS and add all of the players’ remaining hand again.

This time, CARD[i] with the value 0 stands for dropped card (and therefore not in player’s hand).

Then I check every two adjacent cards and see if they are pair.

For example, when I find CARDS[i] and CARDS[j] equal to 0 and assume CARDS[i] is D3, then the possible CARDS[j] would be H3 and S3, the validRange is 2.

Therefore, I check the statement (j <= i + 2) to see if CARD[i] and CARD[j] is a pair.

1. The correctness of dropping a pair after drawing a card from another player.

When player drops a pair, I add the pair to debugger for further cards status tracing.

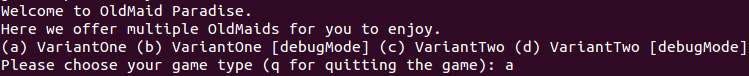
I also use the same method above to do pair checking.

1. The correctness of cards status after the game is over.

After the game is over, I add JOKER to debugger and check if all values in CARDS are changed to 1.

If yes, it guarantees that all cards are recycled from players correctly.

1. **The output from each variant of the game**
2. Play VariantOne :



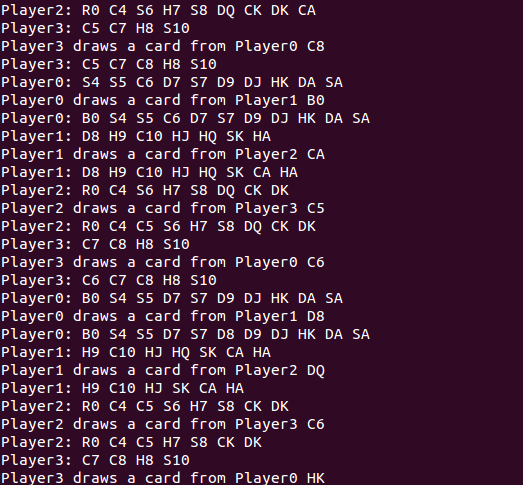
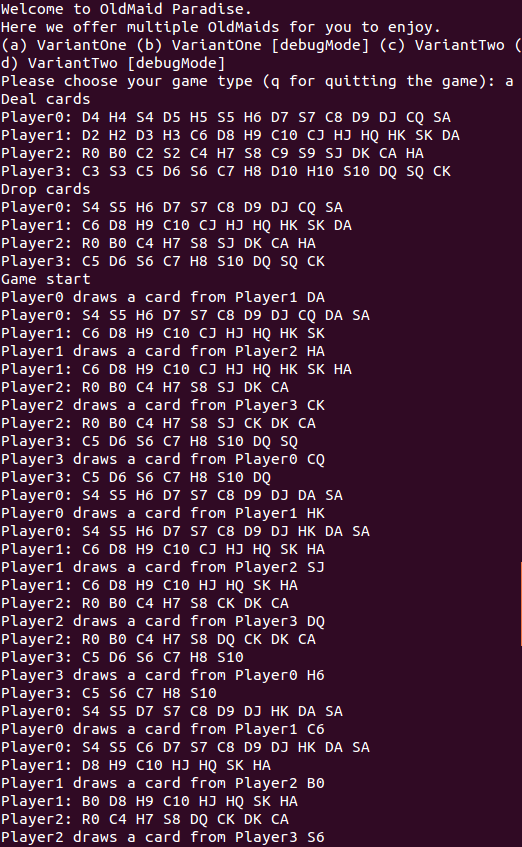
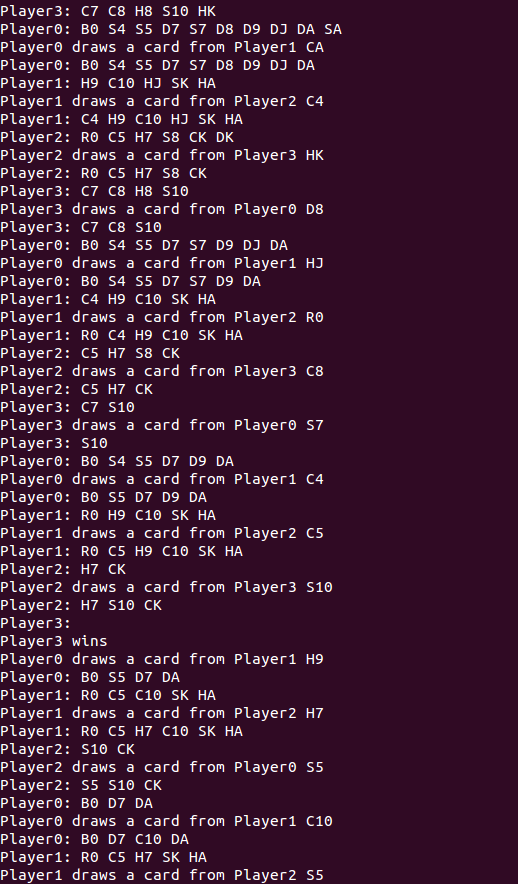
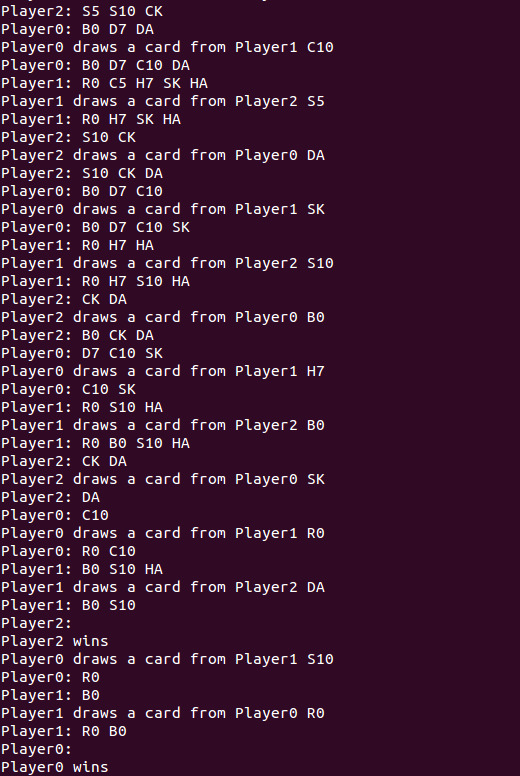
1. Play VariantTwo:

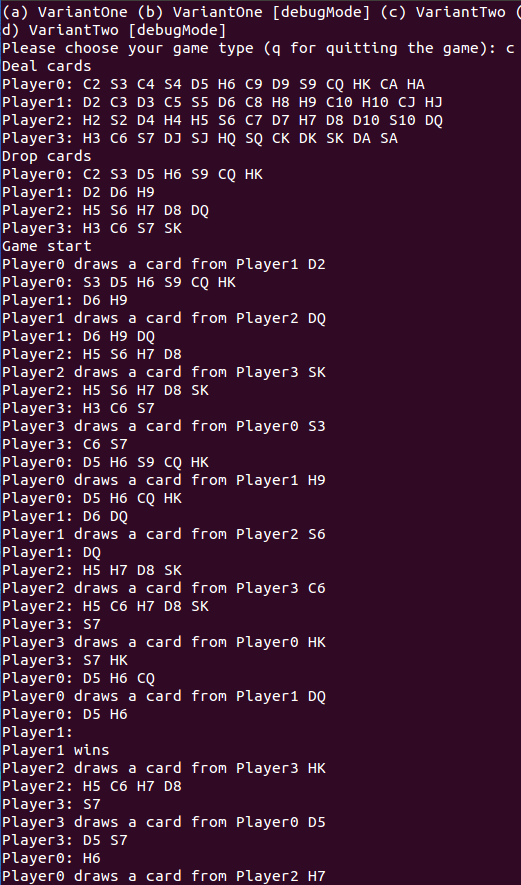


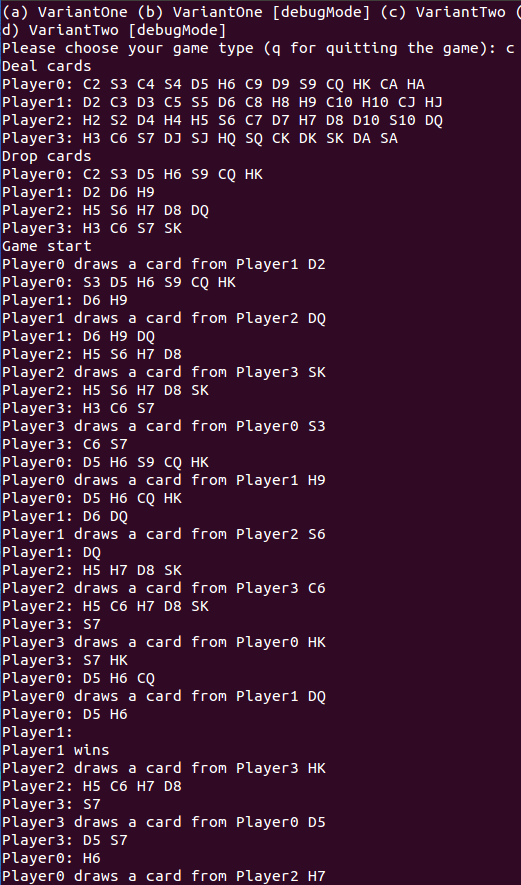
1. Quit game:



1. VariantOne output:



1. VariantTwo output:



1. **BONUS implementation**
2. Player can choose which game type to play.
3. When playing with debug mode, the output will show the correctness checking results.

