OOP Programming Assignment

Term: January 2017

1. General Instructions

The general guideline for this assignment is as follows:

1. Evidence of plagiarism or collusion will be taken seriously and University regulations will be applied fully to such cases, in addition to ZERO marks being awarded to all parties involved.
2. The total marks for the assignment is 100 and contributes 20% to the total grade.
3. This is a **group assignment of 3-4 persons**.
4. The deadline for the submission of reports is on **March 30, 4pm**.
5. You have to submit ONE (1) report but indicate how the work is distributed (refer to section 3.4). Marks will be allocated based on your contribution.
6. You are to submit hardcopy and softcopy of your reports, and complete source code in softcopy in a form of CD. The hardcopy should be bound using **tape binding with two punch holes on the left**.
7. No presentation is required.

2. Background

In this assignment, your team is asked to develop a card game using Java and object-oriented programming concepts. This game is played using standard 52-card deck (13 ranks of 4 suits). The followings describe the game rules of the proposed card game.

Rules and Instructions for Game Play

- The game is played by **2~4** players. Each player gets **4** cards, and another **8** cards are placed face up in a pool. The objective of the game is to *capture* as many cards (points, see **Scoring** below) as possible from the pool of cards using the hand cards
- Starting from the first player, each player plays **one** card at a time, performing one of the following actions:
  - **Capturing a card by forming a Pair**: A player can capture one face-up card in the pool that matches the *rank* (number) of a card from the player's hand (see Figure 1). The player places a hand card down and then collects the matching pair from the pool cards. If the pool contains more than one matching cards, only one may be captured.
  - **Capturing cards by forming a Combo**: A player can capture two or more numerical cards (Ace to 10) in combination if the values of the pool cards add up to the value of a card in the player's hand. For instance, a player with a 5 could

capture a 4 and an Ace (as shown in Figure 2), or two 2s and an Ace, or any combination of cards that add up to 5. Face cards (J, Q, K) cannot be part of a combo.

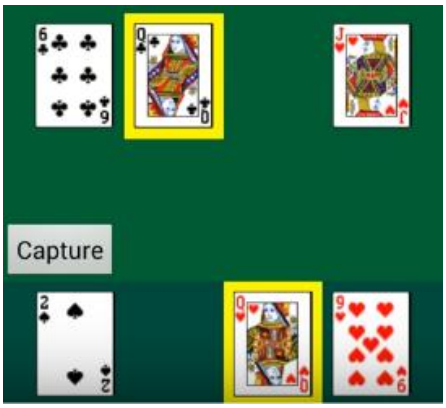

Figure 1. Pair

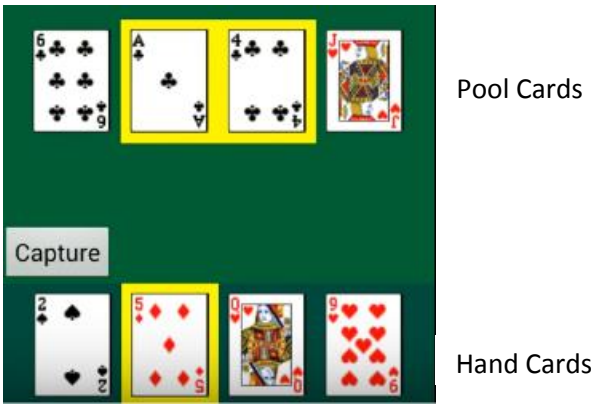

Figure 2. Combo

- **Capturing cards by forming a Run**: A player can capture two or more pool cards to form a *run* (cards of the same *suit* in sequence) with a card in the player's hand. For instance, a player with a ♣6 could capture ♣5 and ♣7 to form run [♣5♣6♣7]. A sequence in different suits cannot form a run, and a run can consist of more than three cards.
- **Trailing**: If a player cannot capture any cards, or for strategic reasons wants to lay off a card, the player may discard a card face-up in the pool. Other players may then capture or utilize this card.

- When there are no more cards in the pool, the current player must discard a card into the pool (trailing).
- A round ends when all players have played all their hand cards. Each of the players then count and accumulate their scores based on the cards they captured.
- **Scoring**: Pair (2 points), Combo (1.2 × number of cards), Run (1.5 × number of cards).
- Whoever first reaches a total of 21 or more points wins the game. If there is a tie, another round is played until a winner is declared (sudden death).

3. Requirements

Your assignment is to develop a program to simulate the card game described in the previous section.

3.1 Program Functions (60%)

Your program must fulfill the following functional specifications:

- Allow 2~4 human players to play the game. No computer player (AI) is required.
- Shuffle the deck, distribute cards to each player, set up the pool cards, and start a round.

- Display the pool cards. The cards should be **sorted** by suit and then by rank.
- Display hand cards and the captures of the current player.
- Allow a player to decide whether to capture cards or discard a card.
- Provide intuitive input methods that allow players to select and capture cards. Check invalid user input.
- **Automatically** determine the correct type (Pair, Combo, or Run) of capture from the selected cards by a player.
- Allow players to select and discard a card to the pool.
- Skip a turn if a player fails to perform a valid action in 3 tries.
- Accumulate score for each player after a round.
- Start a new round.
- End the game and declare a winner.

3.2 Object-Oriented Design (25%)

You much use Java and object-oriented programming techniques in your implementation. Your design should follow good object-oriented design principles. Your implementation should be

- Extensible – a good design should allow extending the game with minimum changes to the main components of the program, and lowering maintenance cost.
- Readable and consistent – codes are well commented and easy to read; writing style and variable names are consistent.
- Efficient and no redundancy – keep it simple.

The followings are the basic classes that MUST exist in your program. The responsibilities for each of the classes, as well as some of their properties and methods are suggested. You are free to decide the data fields and methods of the basic classes, and to add additional classes when necessary.

Card:

Card is the class for representing a card.

- This class should define generic card properties such as suit (*spade*, *heart*, *club*, *diamond*) and rank (*A, 2~10, J, Q, K*).
- A card object should be **immutable**; meaning its content (suit and rank) cannot be changed once it is created.
- Should support sorting of an array of cards using Java generic sorting algorithm.

Capture:

An **abstract** class for representing a generic *capture* object.

- Contain a list of cards that are used to form the capture.
- A method to form a capture when given a list of cards. Should be **abstract**.
- A method to return the score of the capture. Should be **abstract**.

Pair, Combo, Run:

Pair, **Combo**, and **Run** are concrete subclasses of the **Capture** class for representing *pair*, *combo*, and *run* respectively.

- Implement base class methods.
- Responsible for forming a capture of its type when given a list of cards.

Player:

Player class is used to represent a player in the game.

- Contain the hand cards and a list of *captures* that the player has captured.
- Able to remove a card from the player's hand cards.
- Add one or more cards to the player's hand cards.
- Play a card when given a list of selected cards.
- Calculate the total score.

Game:

The **Game** class is the game engine which is responsible for controlling the flow of the game such as distributing cards, displaying cards and messages, getting inputs from the players, maintaining the pool, etc.. It should at least have the following attributes and methods:

- An array of **Player** objects. Number of players can be **2~4**.
- An array of **Card** objects to form the *deck*. The deck should be shuffled at the start of a round. Player draws card from this array initially.
- An array of **Card** objects to represent the *pool*. Played cards are added to or removed from this array during game play.
- A `main`() method that starts the game.

3.3 Extra Effort (10%)

- Extend the game by adding new types of captures.
- Provide easy to use and nice user interface.

3.4 Report (5%)

- A title page stating the title of the assignment, student names and student IDs.
- List the effort and contribution of each team member in terms of percentage in a table such as:

Student 1	Student 2	Student 3	Total
30%	30%	40%	100%

- Program description, including what is the program about, how it works, and how do you design your program to achieve those functionalities.
- UML Class diagrams showing the program structure.
- Java course code. Clearly comment and highlight key points in your code.