OOP Programming Assignment

Term: January 2018

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 28 (Wednesday) 5pm**.
5. Each group has to submit ONE (1) report and indicate how the work is distributed (refer to section 3.4). Marks will be allocated based on your contribution.
6. You are to submit both 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

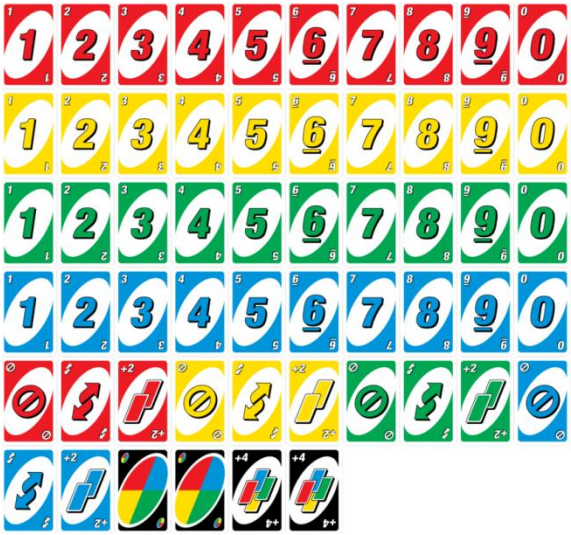
In this assignment, your team is asked to develop a version of UNO card game using Java and object-oriented programming concepts.

The UNO card game consists of a deck of **56** cards with 4 sets of color: red, yellow, green and blue. Each color set consists of 13 cards including 10 rank cards (0, 1, 2, ...9) and 3 action cards (**Skip**, **Reverse**, **DrawTwo**). In addition, the UNO deck also contains 4 wild cards (2 **Wild** cards and 2 **WildDrawFour** cards). Figure 1(a) shows the UNO deck and the action cards and wild cards are highlighted in Figure 1(b).

Rules and Instructions for Game Play

The followings describe the game rules of the proposed card game.

- The game is played by **2~4** players. Each player gets **5** cards initially.
- The remaining **deck** is face down and one card is put face up to start the discard **pile**.
- The objective of the game is to get rid of all the cards in a player's hand onto the discard pile by matching the color or symbol of the previous discard or play a special action card.



(a) UNO deck

(b) Action cards and wild cards

Figure 1. UNO cards.

- Starting from the first player, and continuing clockwise, each player in turn can choose to do one of the following actions:
    a) Discard a card that matches (same color or symbol) the card discarded by previous player (e.g. if the previous player played red 6, the current player can play any red color card or any color of 6).
    b) Discard a **Wild** card and declare a color (regardless of previous color/symbol, a player can play a Wild card and select one of the four colors, and the next player must play the declared color).
    c) Discard a **WildDrawFour** card and declare a color (same as above). Note that by rule a player can play WildDrawFour only if he/she has no matching card (excluding Wild and WildDrawFour) to play. Therefore, the player who plays a WildDrawFour may be challenged (see below for WildDrawFour action).
    d) Draw a card from the top of the deck. If a player has no playable card (a~c) then the player must draw a card from the deck. However, even if the player has playable card, he/she may choose to draw a card from the deck (strategy to confuse others).
- The action cards (**Skip**, **Reverse**, **DrawTwo**, **Wild**, and **WildDrawFour**) have the following special effects when played.
    - **Skip** – the next player loses a turn and the turn passes to the following player.
    - **Reverse** – the direction of play reverses.
    - **DrawTwo** – the next player draws 2 cards from the deck and loses a turn.
    - **Wild** – current player declares a color to be matched and the next player plays any numbers/symbols of that color or a wild card to overwrite that color.

- **WildDrawFour** – current player declares a color to be matched, the next player draws 4 cards and loses a turn, and the following player plays the declared color. A player can play WildDrawFour only if he/she has no matching card (excluding wild cards). The next player may challenge whether the current player is eligible to play a WildDrawFour card.

  Challenge – If a player plays a WildDrawFour card, the next player may elect to challenge if he/she thinks the current player has playable cards. If a challenge is issued and it is successful, the player who played the WildDrawFour must draw 4 cards and the challenging (next) player can play normally. In the case of a failed challenge, the challenging player must draw two more cards in addition to the normal four (2+4=6) and loses a turn.

Note that the action cards only affect the next player. If the first pile card when the game starts is an action card, then the first player should response to the action.

- If the deck is exhausted, the top card of the discard pile is set aside and the rest of the pile is shuffled to create a new deck.
- A game ends when any player gets rid of all the hand cards and that player wins the game.

3. Requirements

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

3.1 Program Functions (55%)

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 5 cards to each player and start a game.
- Display the top card of the discard pile.
- Display hand cards of the current player. The hand cards should be **sorted by color and then symbol**. Wild cards should be positioned at one end of the list.
- Provide intuitive input method that allows players to play a card or draw one or more cards from the deck.
- Check if a player is making valid selection.
- Handle special effect for the action cards mentioned in Section 2.
- Handle special effect for the wild cards mentioned in Section 2
- Handle challenge raised by a player.
- End the game when a player has discarded all hand cards, and declare the 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 such as:

- Single responsibility – a class should have responsibility for a single functionality of a program and that responsibility should be encapsulated by the class.
- Open/closed principle – classes or modules should be open for extension, but closed for modification.
- Efficient and no redundancy – keep it simple.

The followings are the basic classes that **MUST** exist in your program. The responsibilities of each of the classes, as well as some of their properties and methods are suggested. You are free to select the data types and method signatures for the classes, and to add additional classes if necessary.

Card:

Card is the base class for representing a card. This class should define generic card properties:

- *color* – "red", "green", "blue" and "yellow" for regular cards. For Wild and WildDrawFour, the value can be "any" until it is declared.
- *symbol* – rank (0~9) and name (e.g. "Skip", "Reverse", "Wild", …)

The Card class should provide methods to:

- Compare this card with another card object and return result indicating whether this card matches the other card (e.g. same color or symbol).
- Implement the Java `Comparable<Card>` interface (see unit 7), so that an array of cards can be sorted using Java generic sorting algorithm.
- Take action when the card is played. For a regular card, this method does nothing. The action cards should override this method to perform the actions as described in section 2.

Action Cards:

Cards that have special effect such as **Skip**, **Reverse**, **DrawTwo**, **Wild**, and **WildDrawFour** are subclasses of **Card**. They override and implement the action method of the Card class to perform special action accordingly.

Player:

Player class is used to represent a player in the game. A *player* object has a name and consists of the hand cards, and should have methods to:

- Select and remove a card from hand cards.
- Add one or more cards to the player's hand cards.
- Determine if the player has playable cards.

Game:

The **Game** class is the game engine which controls the flow of the game, imposes the basic game rules, displays messages and gets inputs from the players. It should at least have the following attributes and methods:

- An array/list of **Player** objects. Number of players can be **2~4**.
- An array/list of **Card** objects named *deck*. Player draws card from this array. The deck should be shuffled at the start of game.
- An array/list of **Card** objects named *pile*. Played cards are added to this array. When the deck is empty during game play, the top card of the pile is set aside and the rest of the pile are moved to the deck and reshuffled.
- Public methods to support game playing such as changing current player to the next player, reversing play order, skipping a player's turn, requesting user selection, etc. These methods provide interfaces for the action cards to perform special effects.
- A `main`() method that starts the game.

3.3 Extra Effort (10%)

- Game extension – Extend the game by adding new types of action card. A good design should allow extending the game with minimum changes to the main components of the program.
- User Interface – Enhance the application by providing easy to use and nice user interface.

3.4 Report (10%)

- 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 and how it works.
- UML Class diagrams showing the program structure.
- Java source code. The source code must be formatted to fit nicely in the report and well commented.