# (420-206-AB) Programming II
# Winter 2019

**Course Project**

**Worth 16%**

**Final Due Date: Tuesday, 21 May 2019 at 23:59 on Lea**
**Part 2 & 3 Only**

*Deductions for late submissions is 10%/day, up to a maximum of 3 days: 24 May 2019.*
***This document has 5 pages.***

## Objective

The objective of this project is to exercise problem analysis, design solutions and implementing them. You should utilize the skills you have acquired in the Programming II courses to tackle the problem. Use a combination of functional decomposition and object-oriented programing to achieve the task.

## Game Description

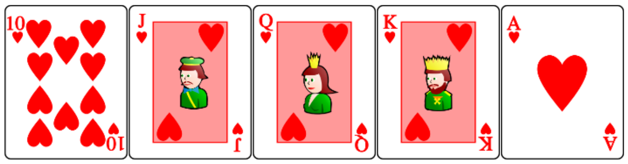
The project will require you to create the following Card game: ***Combine & Win***.

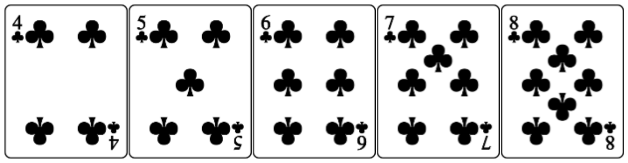
The ***Combine & Win*** game will be based on the two C# classes: `Card` and `Deck`. Create an application for the game that matches the following description:

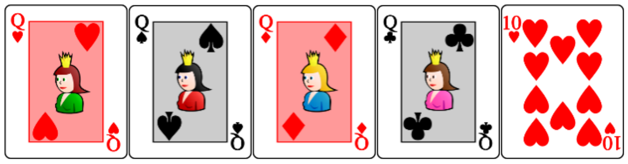
- When the game starts, the player is provided a startup score of **1000 points**.
- The game continues until either the player loses all the points, or he/she decides to quit.
- One round of the game goes as follows:
    1. Deal 5 cards to the player from a shuffled deck.
    2. Allow the player to discard **up to** 4 of the 5 cards.
    3. Deal new cards from the same deck to replace the discarded cards.
    4. Repeat steps 2 & 3 maximum of 3 times.
        - The player will be asked to discard up to 4 cards again.
        - The player may choose not to discard any card and continue.
    5. Find if the player got a **winning combination** or not. (details below)
    6. Based on the **combination**, the player will either gain points or loose points.
        - Check the tables below for each combination winning points.
        - If the player did not match any of winning combinations, the player loses 100 points.
    7. If the score reaches zero (or less), the player loses. Thank him/her for playing and quit the game.
    8. If the score is above zero, ask the player if they wish to continue.
        - Continue: all cards are returned to the deck and shuffled again.
        - Otherwise quit.

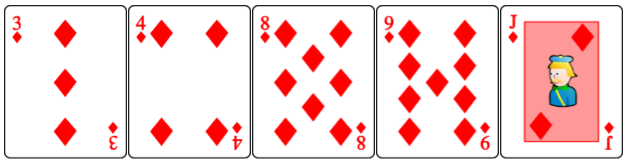
## Winning Combinations

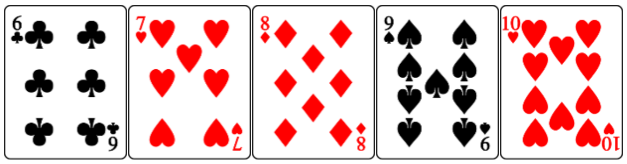
Below are the possible winning combinations and number of points gained for each:

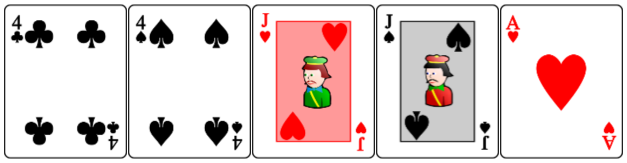
| **Combination Name** | **High-Five** |
|---|---|
| **Description** | Ten, Jack, Queen, King and Ace all of the **same suit**. |
| **Available possibilities** | 4 combinations |
| **Gaining Points** | 800 points |
| **Example** |  |

| **Combination Name** | **Sequence** |
|---|---|
| **Description** | Any five cards of the **same suit** and showing **consecutive** numbers. (Ex: 4, 5, 6, 7, 8)<br>Ace can be used as one, example: A, 2, 3, 4, 5.<br>Wraparound is not allowed, example: K, A, 2, 3, 4. |
| **Available possibilities** | 36 combinations |
| **Gaining Points** | 600 points |
| **Example** |  |

| **Combination Name** | **Quadruplets** |
|---|---|
| **Description** | All four cards of the **same index** and any other card. (Ex: Q, Q, Q, Q, 10) |
| **Available possibilities** | 624 combinations |
| **Gaining Points** | 400 points |
| **Example** |  |

| Combination Name | **Family** |
|---|---|
| **Description** | Any five cards of the **same suit**, but **not in sequence**. |
| **Available possibilities** | 5,108 combinations |
| **Gaining Points** | 200 points |
| **Example** |  |

| Combination Name | **Mixed Sequence** |
|---|---|
| **Description** | Any five cards showing **consecutive** numbers, but **not from the same suit**.<br>Ace can be used as one, example: A, 2, 3, 4, 5.<br>Wraparound is not allowed, example: K, A, 2, 3, 4. |
| **Available possibilities** | 10,240 combinations |
| **Gaining Points** | 100 points |
| **Example** |  |

| Combination Name | **Double Twins** |
|---|---|
| **Description** | Two cards showing the same numbers and another two cards showing the same numbers (but not all four numbers the same) plus any other card. |
| **Available possibilities** | 123,552 combinations |
| **Gaining Points** | 50 points |
| **Example** |  |

All other combinations will result in the player losing **100 points** for the current round.

Example, if the player gets a High-Five, the score will become 1800 points after the first round.

Note: A working game `Combine&Win.EXE` is provided on Lea. Mimic it as much as possible. See me if you want to the output to show something else.

## Project Parts

### Part 1:

**Plan the program (3% of final grade). Due: Sunday, May 5th, 2019 at 23:59.**
No late submissions accepted and 0/3 will be given if late.
Monday, May 6th, 2019 will discuss planning.

1. Create a document that includes your plan/outline/pseudo-code for:
    a. Each class (Card and Deck)
    b. Main()
    c. Functional decomposition design: before implementing the methods, break down the program into methods and have enough of them. Write the method headers like the teacher provided you in assignment 4. For each method explain:
        i. Inputs.
        ii. Outputs.
        iii. Task description in details, step by step.
2. Design and implement the Card and Deck classes.
3. Create flowcharts (Visio) to detect each winning combination: High-Five, Sequence, Quadruplets, etc.

### Part 2:

**Implement the plan (12% of final grade). Due May 21th, 2019.**

1. DO NOT USE Lists, use arrays.
2. **IMPORTANT**: In the code, create a test menu with each hardcoded test scenarios: High-Five, Sequence, Quadruplets, etc. This facilitates testing without modifying the code. See `Combine&Win.EXE`.
3. Find a classmate which will be your tester.
   Mention the tester's name in the program header comments: *Tested by Mike Smith*.

### Part 3:

**Demo: Worth 15%. Due May 21th, 2019.**
Not demoing your working code on the teacher designated time will result in automatically a deduction of 15%.

### Grading scheme example: (final one might differ)

1. Use of constants:
    a. Example (1) number of discarding round: `const int DISCARDING_ROUNDS = 3;`
    b. Example (2) starting score: `const int STARTING_POINTS = 1000;`
2. **No global variables unless they are constants**.
3. Combination detection logical errors or bugs:
    a. Example (1) A bad detection: High-Five is detected as Sequence.
    b. Example (2) A bad detection: Ace, Two, Three, Four and Five of the same suit not detected as Sequence.
    c. Some combination detection bugs.
    d. Many combination detection bugs.
4. Discarding cards: logical errors, bugs or missing messages:
    a. Example (1) Play can discard 4 times the same card.
    b. Example (2) Do not discard and show cards right away. Discard all and then replace/display all.
    c. Example (3) Discarding is max 4 cards not 5.
    d. No error message when invalid card number discarded.

5. Scoring errors and bugs:
    a. Example (1) Game continues when score reaches zero points or less.
    b. Example (2) Total score incorrectly calculated after each round.
    c. Example (3) Total score not updated properly. (adding or subtracting points)
6. Program doesn't loop. It plays only 1 round and quits.
7. Code redundancy: create a print method instead of coding it repeatedly in the test method.
8. For Testing: Call DetectCombination method instead of each isHighFive, isSequence, etc.
9. Missing menu with coded test scenarios. I will have to manually write the test scenarios to correct: a very lengthy process for each student in this situation.

## Requirements and Deliverables:

Your applications must meet the following requirements:

1. Working code with comments.
2. All programs should have pause at the very end: Console.ReadKey();
3. Demo of working code: worth 15%.
4. Clean your Visual Studio Project (**Build->Clean solution**).
5. Compress all solution folders and their softcopy flowcharts into a ZIP file.
   (retain the tree structure of the project please!)
6. Submit the ZIP file on LEA.
7. Do not submit anything printed or send the file through MIO.