CS 171: Introduction to Computer Science II
Assignment #2 (Advanced OOP): A Card Game

[Due: on Gradescope]
[Late Submission: See Syllabus for policy details.]

Note: You can optionally work in **pairs** for this assignment. Only one student should submit the required files on Gradescope while indicating who your partner is. Additionally, please include *both* student names in the honor code at the beginning of all submitted Java files.

1 Let's play cards! Problem Description and Goals

Game description: In this assignment, you will be implementing a simple game of cards. The game uses a standard 52-card deck; there are four *suits* and each suit has 13 *ranks*. For simplicity, we use numbers to identify cards in this assignment. Thus, we have suits 1, 2, 3, and 4, and ranks from 1 to 13 (inclusive). Each card is identified by its suit number followed by its rank. For example, card 102 refers to the card whose suit is 1 and rank is 02. Therefore, we can use the simple formula `suit*100+rank` to produce a card's identifier.

Figure 1 below shows the different entities involved in this game: a **dealer** who has a **deck** of **cards**, two **players**, and a **table** with exactly four *places* where players can place their cards during the game. Each player has a *hand* of cards that is private to them (i.e., not visible to anyone else), and a *bank* of cards containing all cards they won during the game (visible to the public).

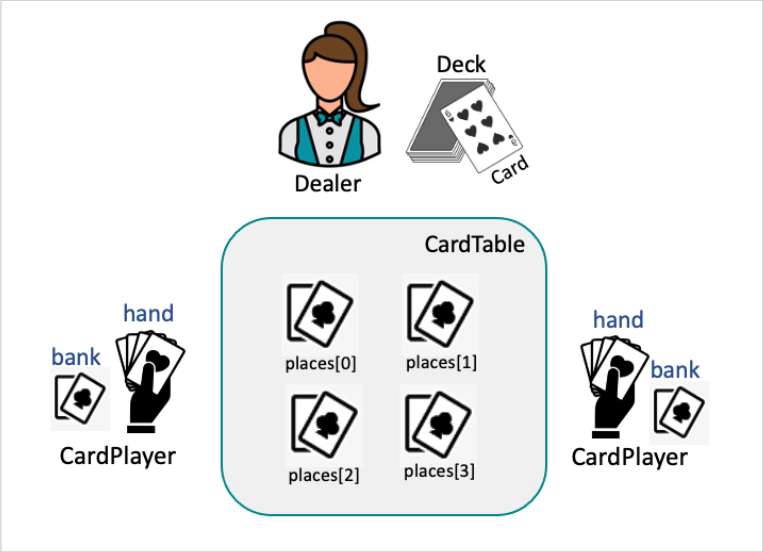

Figure 1: An illustration of the different entities involved in our card game.

The game begins with the dealer shuffling the deck then distributing (i.e., dealing) the cards to the two players. The players then take turns in placing their cards on the table, one card a time, starting with place 1 (i.e., places[0] in our array implementation), followed by place 2, etc.; when a player places a card in place 4, the next player places their card in place 1, and so on. The top card in each place on the table is visible to everyone. Note: We use the term **current place** to indicate where the current player can place their card.

When a player wants to play a card, there are two possible scenarios:

1. If there is another visible card (in places *other* than the current place) whose *rank* is the same as the current player's card rank, then the player takes that card from the table and adds *both* cards to their bank. One point is added to the player's score.
2. If none of the visible cards (in places *other* than the current place) have a rank that matches the current player's card rank, then this new card becomes the top card in the current place on the table. No points are added to the player's score.

Then, the next player plays a card, and so on. The game continues until both players finish their cards. We then count the number of pairs of matching cards they have collected in their *banks*, and the winner is the player with more pairs (the most points). If there is a tie, then for simplicity we can consider player 1 to be the winner (after all, this is just a simulation ;-).

Sample Game: Below is the output of a sample game (where all the rules are implemented correctly). We print the content of all 4 table places in each iteration, with -1 indicating that no card is placed in that position: **Your output should match the following exactly if you implement the game as specified.**

```
CardTable Places
-------------------------
| p1  | p2  | p3  | p4  |
-------------------------
| 205 |  -1 |  -1 |  -1 |
| 205 | 106 |  -1 |  -1 |
| 205 | 106 | 102 |  -1 |
| 205 | 106 | 102 | 304 |
| 109 | 106 | 102 | 304 |
| 109 | 303 | 102 | 304 |
Matched ranks: 109 (on table) and 409 (Player 1's card)
| 205 | 303 | 102 | 304 |
| 205 | 303 | 102 | 307 |
| 210 | 303 | 102 | 307 |
| 210 | 404 | 102 | 307 |
| 210 | 404 | 203 | 307 |
| 210 | 404 | 203 | 112 |
| 306 | 404 | 203 | 112 |
| 306 | 301 | 203 | 112 |
| 306 | 301 | 407 | 112 |
| 306 | 301 | 407 | 202 |
| 413 | 301 | 407 | 202 |
| 413 | 309 | 407 | 202 |
| 413 | 309 | 212 | 202 |
| 413 | 309 | 212 | 101 |
| 107 | 309 | 212 | 101 |
| 107 | 209 | 212 | 101 |
| 107 | 209 | 311 | 101 |
| 107 | 209 | 311 | 410 |
| 313 | 209 | 311 | 410 |
| 313 | 204 | 311 | 410 |
| 313 | 204 | 206 | 410 |
| 313 | 204 | 206 | 308 |
| 201 | 204 | 206 | 308 |
| 201 | 104 | 206 | 308 |
| 201 | 104 | 211 | 308 |
| 201 | 104 | 211 | 113 |
```

```
| 207 | 104 | 211 | 113 |
| 207 | 108 | 211 | 113 |
| 207 | 108 | 402 | 113 |
| 207 | 108 | 402 | 310 |
| 411 | 108 | 402 | 310 |
| 411 | 312 | 402 | 310 |
| 411 | 312 | 213 | 310 |
| 411 | 312 | 213 | 103 |
| 208 | 312 | 213 | 103 |
| 208 | 111 | 213 | 103 |
| 208 | 111 | 105 | 103 |
Matched ranks: 105 (on table) and 405 (Player 2's card)
| 208 | 111 | 213 | 103 |
Matched ranks: 103 (on table) and 403 (Player 1's card)
| 208 | 111 | 213 | 310 |
| 208 | 401 | 213 | 310 |
| 208 | 401 | 412 | 310 |
| 208 | 401 | 412 | 406 |
| 110 | 401 | 412 | 406 |
| 110 | 302 | 412 | 406 |
| 110 | 302 | 305 | 406 |
------------------------
End of game. Total #iterations = 51
Player 1 bank has 4 cards: 409 109 403 103
Player 2 bank has 2 cards: 405 105
The winner is: Player 1 (Points: 2)
```

Goal: Your goal in this assignment is to write a Java program that simulates this card game and prints the sequence of played cards and the winner at the end, as shown above. You will take advantage of various exciting OOP principles in Java to implement this game simulator, including inheritance, abstraction, interfaces, generics, and ArrayList.

OOP is a fantastic programming paradigm for this game simulator, as it will help us represent and group various entities of the game in a way that reduces code rewriting and offers modularity, encapsulation, flexibility, and reliability — among other benefits!

2 Project Structure and Starter-Code Description

You will be given starter-code that includes fully implemented interfaces and classes (Card, Dealer, Deck, GeneralPlayer, Table, and Main) which you are required to read and understand carefully. Then, you will implement and submit a new class named **CardPlayer** which represents an individual player. Finally, you will be given a partially implemented class named **CardTable** which you need to complete and submit as well.

Important: Start by **carefully reading and understanding the completed Java** files given to you first, before moving on to the parts that you need to implement. To make your job easier, we generated a comprehensive documentation of all classes and interfaces in our assignment package, available here: `https://www.cs.emory.edu/~nelsay2/cs171_s23/a2/doc/index.html` Please read it very carefully and let us know during office hours or Piazza if you have any questions!

More details about Main.java: This class will help you test your code and make sure that your implementation of CardPlayer and CardTable are correct. In the `main` method, a new `Deck` object is created to represent the deck of cards used in the game. Two `CardPlayer` objects are created. A `Dealer` object is created and the players and deck are assigned to it. And a `CardTable` object is created to represent the table in the game.

The first player's turn is set to `true` and the game loop begins. The loop will continue as long as both players have cards in their hands.

When it is a player's turn and their card is played (i.e., placed on the table), the proper methods from class CardTable are called in Main to check if the ranks of other cards on the table are a match to the newly placed card or not, and so on. After each player turn, the cards on the table are displayed. Once the game loop is finished, each player's bank of cards is displayed, followed by the declaration of the winner.

Note that the methods that *you* will implement in other classes will be called in class Main. This can help you test and debug your code.

Required Classes: CardPlayer.java and CardTable.java. These are the two (and only) classes that you need to **submit** on Gradescope for this assignment.

(a) CardPlayer represents a player in this card game and it extends the given abstract class GeneralPlayer. You are responsible for writing this class entirely on your own, but your implementation must adhere to the full documentation given here (i.e., class properties, type parameters, fields, constructors, methods, etc.):
`https://www.cs.emory.edu/~nelsay2/cs171_s23/a2/doc/cs171a2/CardPlayer.html`

(b) CardTable represents the table where a game is being played. This class must implement the Table interface with type parameters that enable it to work with Card and CardPlayer objects. It is partially implemented for you; your completed implementation must adhere to the full documentation given here (i.e., class properties, type parameters, fields, constructors, methods, etc.):
`https://www.cs.emory.edu/~nelsay2/cs171_s23/a2/doc/cs171a2/CardTable.html`

3 Getting Started:

1. Read this handout again! Yes, please read it carefully and pay attention to all the details mentioned here.
2. Then, go over `Main.java`. Read the implementation and comments carefully and make sure you understand how the game works overall (before going into details).
3. Now read `Deck.java` and the provided documentation for `CardPlayer.java` carefully. Implement `CardPlayer.java` and see how `Dealer.java` works.
4. The core of this game happens in `CardTable.java`. Complete its implementation and verify by running `Main.java` that your game output is correct (see sample output in this handout).
5. Submit only `CardPlayer.java` and `CardTable.java` on Gradescope.
6. **Debugging Tip:** You are free to write helper methods if needed. You can also add printing statements inside the fully implemented classes given to you if that helps. But please **comment** out any printing statements you add to `CardPlayer.java` and `CardTable.java` before submitting them.

Grading

- **CardPlayer:** 50 points (10 points inheritance and generics implementation, 15 points instance variables and constructors, 25 points method implementations)

- **CardTable:** 45 points (10 points inheritance and generics implementation, 15 points instance variables and constructors, 20 points method implementations)
- **Code clarity, style, and comments:** 5 points

Honor Code

The assignment is governed by the College Honor Code and Departmental Policy. Remember, any code you submit must be your own; otherwise you risk being investigated by the Honor Council and facing the consequences of that. We do check for code plagiarism (across student submissions and against online resources). Please remember to have the following comment included at the top of the file.

```
/* THIS CODE WAS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING
CODE WRITTEN BY OTHER STUDENTS OR COPIED FROM ONLINE RESOURCES.
_Student1_Name_Here_   _Student2_Name_Here_ */
```

Submission Checklist: We've created this checklist to help you make sure you don't miss anything important. Note that completing all these items does not guarantee full points, but at least assures you are unlikely to get a zero.

- ☐ Did your file compile on the command line using JDK 11 or above (e.g. JDK 17)?
- ☐ Does the class Main.java compile and run successfully while using your implemented classes?
- ☐ Did your submission on Gradescope successfully compile and pass at least one autograder test case?
- ☐ Have you included the honor code on top of the file?
- ☐ Did you remove the `TODO` prefix from the methods you needed to implement?
- ☐ Did you give your variables meaningful names (i.e., no `foo` or `bar` variables)?
- ☐ Did you add meaningful comments to the code when appropriate?