Title : Project 5B - Let's Go Crazy
Modules : 10 Advanced Classes & 11 Polymorphism
Point Value : 50
Deadline : Due before 11pm on Thursday December 8th
Topics Covered : Object-oriented programming, encapsulation, inheritance, polymorphism, UML diagramming

Note: You absolutely may NOT use ArrayLists or other Java Collection classes in the code you write for this project.

Introduction

In Project 5 Part A, you created classes that represent cards and decks. Now, it's time to make a card game! Crazy Eights is a classic card game for two or more players. Since two decks are needed for groups of players greater than 5, we'll keep things simple and restrict our game to groups of 2-5 players only. We've also made a few small adjustments to the rules of game as stated in Wikipedia, so keep reading to learn the rules of our version:

The main objective is to be the first player to get rid of all your cards. Here's how to play:

- Begin with a standard 52-card deck of playing cards, which has been shuffled, and deal out cards to each player in the usual "round-robin" fashion, meaning that each player gets one card before any player gets their second card, and so on. When two players are playing, each player should be dealt a total of 7 cards. When 3, 4, or 5 players are playing, each player should be dealt a total of 5 cards.
- Once the appropriate number of cards are "dealt" to each player, the remaining cards of the deck are placed face down at the center of the table to create the "draw pile". The top card in the draw pile is then moved to a separate empty pile known as the "discard pile" (and placed face up) to start the game.
- Players then take turns placing a single "matching" card on the discard pile. To match, the card must equal the rank or suit of the previously played card, or, since eights are "wild" in Crazy Eights, we'll consider any two cards to match if either one of them has rank eight. (Note: This is a slight departure from the rules as stated in Wikipedia.)
- If a player doesn't have a matching card (or an eight) in their hand when their turn begins, they repeatedly draw cards from the draw pile until they get a playable card, at which point they must play it on the discard pile. A player may not ever draw a card from the draw pile if their hand contains a playable card.
- Whenever the top card on the discard pile is a Queen, the next player's turn is automatically skipped. (Note: This is also a slight departure from the rules as stated in Wikipedia.)
- If the draw pile ever runs out, all the cards in the discard pile except the top card are moved to the draw pile and shuffled.

- The game ends as soon as any player empties their hand. That player is declared the winner.

This project is fairly different from previous projects this term. You will spend most of your time reading the code that we've provided to you (and making sense of it) rather than writing code.

What is given to you?

You are given a partially-implemented project (starter code) as a zip file to download. Comments in the given code will help you understand how the project is designed to work. The zip archive contains the following files:

```
Card.class //this class is completed and already compiled for you
CardCollection.java
Computer.java
Deck.java *
Demo.java
Game.java *
Hand.java *
Player.java *
Start.java
User.java *
```

The provided files compile as-is and satisfy Checkstyle, except for 9 warnings about TODO: comments indicating code you'll need to fill in. The files you will be editing are marked with an asterisk (*) above. You may not modify any others.

How does Project 5B relate to Project 5A?

You implemented a version of `Card` and `Deck` classes in Project 5A, but we're using **modified** versions of these for Project 5B, so you **won't reuse your work from Project 5A** here.

The code provided to you is missing the `Card.java` file. Instead, we've provided a pre-compiled version of the class named `Card.class`. Remember that .class files are just compiled .java files that contain machine code (and thus aren't legible to humans). As a result, you won't be able to see the exact code in Card.java. You must use the supplied `Card.class` file for Project 5B, rather than your own implementation from Project 5A. This class contains the same methods that were specified in Project 5A, so you can create a Card using a two-argument constructor (rank should be passed in before suit), and also call `getRank()`, `getSuit()`, `toString()`,

`equals(Object o)` and `compareTo(Card other)` on a Card object. These are all the Card methods that you will need.

The starter code contains `Deck.java` but this version of `Deck` is different from the version that you were asked to implement in Part A.

> Are you submitting Project 5A late? You may **not** use the version of `Deck` provided in Part 5B as your solution to Part 5A.

Miscellaneous

- The game as it is implemented displays each player's "hand" when it is their turn. This is for the benefit of you (the programmer) to check whether you have implemented everything correctly. In reality, for this game to make sense as played, the players would not be able to see each other's hands at all.
- The provided code implements the essential features of the game. You may assume that any user will provide the expected type of input when prompted. Note that in some cases, when the user enters a value of the right type but in the wrong range, we throw an exception to indicate the problem.

What is expected of you?

Your task is:

- to explore and understand the project source code; particularly reflect on the structure (Object-Oriented Design) of the program, making use of the comments and Javadocs.
- to complete the project (the missing parts) according to the specifications of the game and the specifications (documents/comments) in each method in the starter code. You must not change the provided code, including any instance variable declarations.

What to submit?

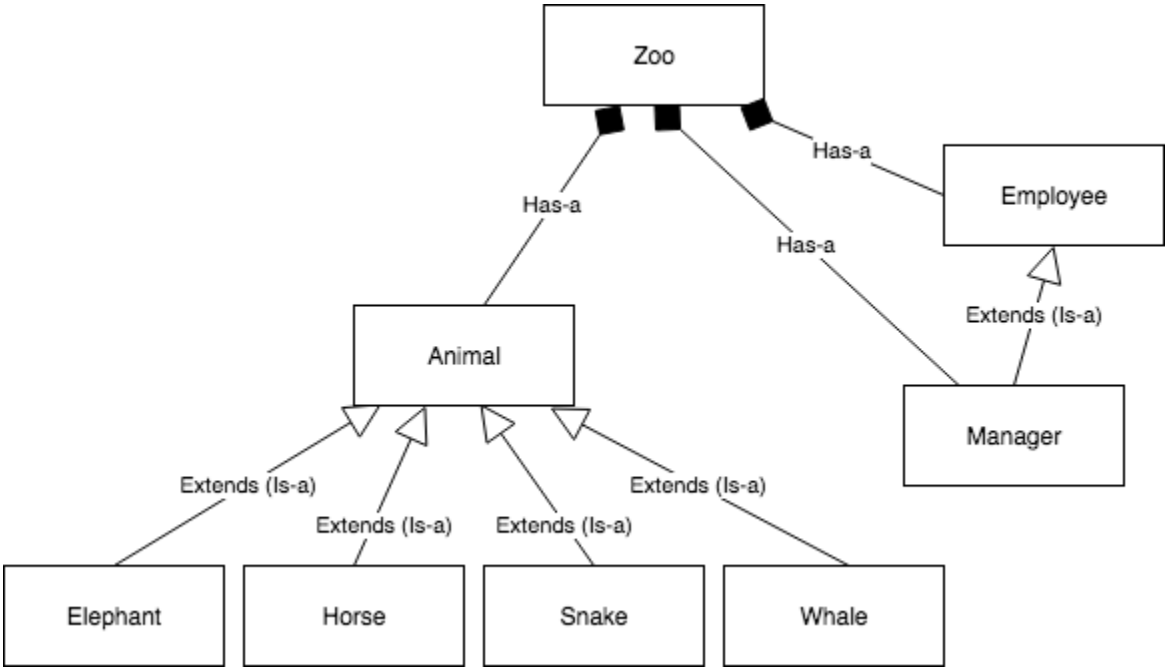
You will submit 6 files together via the Project 5B Gradescope submission link for this project:

1. Code: Only submit the files marked with a star in the list above. All the java files listed below must be included in your final submission, but don't include anything extra. Do not submit the .class files or any of the example runs. The files to submit are:
    a. `Deck.java`
    b. `Game.java`
    c. `Hand.java`
    d. `Player.java`
    e. `User.java`

2. Writeup: This is a 1-2 page writeup in PDF format, named `Writeup.pdf`. See below for requirements.

The writeup must contain your name, JHED, and two sections:

1. A short description (200 to 300 words) of "How object-oriented programming (OOP) is used to implement this game". Discuss the idea of OOP, note how e.g. Encapsulation, Composition, Inheritance (and other aspects of OOP) are used in the project.
2. A Unified Modeling Language (UML) class diagram of all the classes which are part of this project (except for `Demo` and `Start`). The diagram shown below is an example of a UML diagram from Module 11. You do not need to include fields or methods in the box for each class type, just its name.



Your diagram should represent relationships between classes as illustrated in the example diagram above. That is, in addition to displaying different arrowheads to represent the different relationships, you must also label each arrow in your diagram with either "Extends (Is-a)" or "Has-A", as appropriate. Notice that for the inheritance relationship the arrowhead points into the superclass. For composition, the diamond (collector) is located at the class that contains a data member of the other class type. We advise you to create the UML diagram first (before writing code) as it will help you understand the program structure.

Draw your UML diagram using a software tool such as Microsoft Powerpoint or this free, online tool: https://app.diagrams.net/. Do not use tools that automatically generate UML from source code. This will be considered cheating, and you will receive a zero for such a submission. If you are not comfortable generating a PDF file, please visit an office hour to get help. Act early on this, well before the deadline.

What to do?

The process of completing this assignment is broken into four tasks to guide your efforts:

Task 1

- Download our starter code into a new folder where you'll complete this project. Take some time to review the structure of the files, and to sketch a UML diagram representing that structure before proceeding.

Task 2

After carefully reading the supplied documentation in the starter code, implement the following constructors and methods:

- In `Deck.java`
  - `public Deck()`
  - `public Deck(String label)`
- In `Hand.java`
  - The missing portion of `public Card discard(int i)`
- In `Game.java`
  - The missing portion of `private void initializeGame()`
  - `private Player nextPlayer()`
  - `public static boolean cardMatches(Card card1, Card card2)`
  - `private boolean gameOver()`

Hint: Be sure that the `discard` method in the `Hand` class does not disrupt the order of the remaining `Cards` in the `Hand`.

Once you are done, run the `main` method in the `Demo.java` to "test" your implementation up to this point. This version will automatically create two computer players, and you should now see them play against each other until one of them wins the game.

Task 3

Carefully reading the supplied documentation in the starter code, implement the last two remaining unfinished methods to enable play using human users:

- In `Player.java`
  - `public boolean hasPlayableCard(Card top)`
- In `User.java`
  - `public Card makeMove(Game crazyEight, Card top)`

Once you are done, you can run `Start.java`, and play a game with a mix of computer and human users. A sample run with computer and human users is given as SampleRun.txt.

> **Hint:** In implementing `makeMove` for the User class, review how the same method was implemented in `Computer.java` to get some ideas. You need to read some input data from the user – refer to the sample runs mentioned above for an indication of how the game is played between a user and computer.

Task 4

Reflect on the object-oriented principles at work in this project and complete your writeup document.

Submission & Grading

- Using the submission link named Project 5B, you must submit exactly the following 5 required java files and one PDF file:
  - `[4 points] Deck.java`
  - `[12 points] Game.java`
  - `[4 points] Hand.java`
  - `[4 points] Player.java`
  - `[15 points] User.java`
  - `[8 points] Writeup.pdf`

- `[3 points]` All code you submit for this assignment must be Checkstyle-compliant and should adhere to proper style guidelines in general.
- Do not submit any `.class` files, any sample runs, or any other starter files beyond those listed above.
- If you submit multiple times on Gradescope, be sure to include all required files in each submission.
- You will earn 0 points for a task in which your submitted code does not compile.