READ this ENTIRE specification document before writing any code or asking for clarifications. These specifications will be discussed during the onsite class sessions on November 7.

Overview

Using Java, write a simplified implementation of a Card Game. In this simplified collectible card game, **two players** participate in the game, each with a maximum of **five (5) cards** in hand. A **game master** is responsible for conducting the game, keeping track of whose turn it is, making sure that game-play rules are followed, and declaring the end of the game when one of the players wins. For a player to win the game, they must be the first to **eliminate** three (3) cards from the other player's hand. For every card that a player successfully eliminates, they are awarded a token. Hence, the player wins the game when they have a total of **three (3) tokens**.

Game Play

The game master starts the game by initializing the two players (e.g. Player1 and Player2). The game master has a deck of 26 cards (implemented with an ArrayList). The deck is assembled by a ***private*** method, **assembleDeck()**, the code for which shall be provided on the course website. The topmost card, i.e. the first to be dealt from the deck, is at index 0.

The game master alternately deals a card to each player until each player's hand (implemented with an array) is full with five (5) cards. Each dealt card counts as a player's turn. The **first** card that a player receives from the dealer is automatically the **active card**.

The game master keeps track of the total number of turns that have been played in the game, starting at 1. The game master also uses this number to determine whose turn it is. An odd number indicates that it is Player1's turn, while an even number indicates that it is Player2's turn.

On a player's turn, the following events take place:

- The player must choose between two actions — **attack** or **swap**.
  - A **swap** is an alternative action that a player can perform instead of the usual attack. A swap involves determining the highest "determining product" of the other cards (excluding the current active card) in hand and "activating" it by swapping places with the current active card. Swapping cards counts as a turn.
  - When a player chooses to **attack**, the following take place:
    - First, the game master identifies the active card of each player. For example, if it is Player1's turn, Player1's active card is the card-in-play, while Player2's active card is the target-card.
    - The game master determines how much damage the card-in-play will deal on the target-card by doing the following:
      - The initial damage-to-be-dealt is the value of the card-in-play's attack power.
      - Determine if the target-card's type has a weakness to the card-in-play's type. If the target-card is weak to the card-in-play, the damage-to-be-dealt value is doubled.
        - Dragon cards are weak to Fairy cards.
        - Fairy cards are weak to Ghost cards.
        - Ghost cards are weak to Dragon cards.
      - Determine if the target-card's type has a resistance to the card-in-play's type. If the target-card is resistant to the card-in-play, the damage-to-be-dealt is halved.
        - Dragon cards are resistant to Ghost cards.
        - Ghost cards are resistant to Fairy cards.
        - Fairy cards are resistant to Dragon cards.
      - The computed damage is then dealt to the target-card.
    - The game master checks if the target-card has been eliminated by checking its remaining health. If the value of the target-card's health has reached zero or less, the target-card is eliminated. The player discards the eliminated card, promotes all cards in hand, and takes the next card from their hand as their new active card. The player whose card is eliminated will draw a new card from the game master's deck, which is placed in the next available slot in their hand. The player who eliminated the card gets a token.

- ■ At the end of each turn, the game master checks if the player who has just been awarded a token has accumulated three tokens. If the player has three tokens, that player wins and the game is over. If the player does not yet have three tokens, the other player takes a turn. The game master will continuously allow each player to take a turn until one of them wins.

Implementation

1. Create the following classes that simulate the game play.
   - **Card.java** – has a type, a name, a value for health, and a value for attack power
   - **Player.java** – has a name, a maximum of five cards in hand, an indicator for having a full hand, and a token counter
   - **GameMaster.java** – has an indicator for the game being won, a turn counter, and is associated with two Players
   - **GameConsole.java** – provides the user with a console (terminal) interface through a main method and reads input from a Scanner object

   A complete description of each class is provided in the Java documentation pages on the course website.

2. Refer to the simulated execution of **GameConsole** shown in **GCRun.pdf** on the course website. Text that appear in **black** indicate program output while text in **blue** indicate user input. This is only a prolonged example. You are free to modify the prompts and any other options that you would like your program to support. However, the content and format of output messages specified in the documentation pages must **match** your program's output **exactly**, including white spaces.

3. All of your classes should include javadoc-formatted method comments (in your own words) and class header comments. Class header information should include a description of the class (in your own words), your full name followed by your ID number as the author, and the date you created your program as the version. The class header comment block should be followed by the comment block containing the authorship certification.

4. Place your java source code files in a folder named according to the convention: **CardGame-Section-LastName-GivenName-IDNumber**, for example: CardGame-K2-Medalla-Alberto-999999

5. In the folder above, include a ***properly accomplished*** Individual Certificate of Authorship with the following details:
   - Title of Submission: CSCI 21 Final Project - Card Game
   - Type of Submission: Program
   - Cite your sources in the source citation text input area. This area automatically adjusts to accommodate the length of the text provided.
   - Fill in the Student and Course Information.
   - File Name: **CardGame-IDNumber.pdf**, for example: CardGame-999999.pdf

6. The folder must contain **only** the following:
   - all .java files required for the implementation of your program
   - one .pdf file containing a properly accomplished Individual Certificate of Authorship
   - other necessary GUI-related files, if applicable

7. Archive (or compress) your folder. Properly archiving (or compressing) the folder will produce a file named **CardGame-Section-LastName-GivenName-IDNumber.zip**

8. Submit the zip file through the Canvas assignment page by **Thursday, 07 December 2023, 23:59**.

Grading

The highest score that a student can earn by fulfilling all of the requirements stated above is **80 points (C+)**.

Projects will be tested on the console / terminal. `java GameConsole Player1 Player2`

To receive a higher grade, the student may implement any of the additional options described below. The add-ons are independent of each other and should not interfere with the base implementation described above.

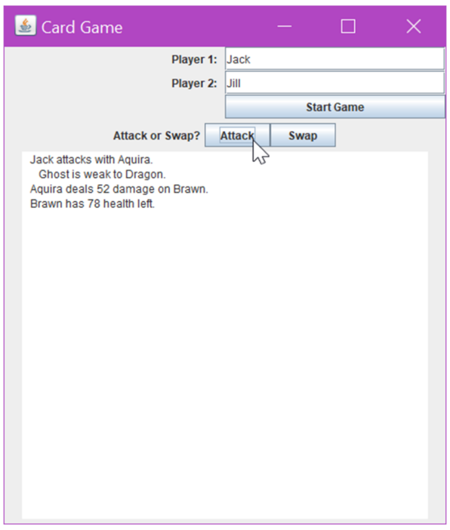
<u>Add-On 1: Random Deal</u> (+5 max)
Enable the GameMaster (whether through the console or the GUI) to have the option to deal random cards from the deck at the beginning of the game, instead of the pre-arranged sequence created when the deck is assembled. Note that the deck is **<u>not</u>** shuffled; and this does **<u>not</u>** affect the way the player draws a card when their card is eliminated. Refer to **RandomDeal.pdf** on the course website for sample executions.

<u>Add-On 2: Start a New Deck from File</u> (+10 max)
Enable the GameMaster (whether through the console or the GUI) to have the option to assemble a special deck of cards from an input file, without using the previously provided assembleDeck() method. It should read from a file called **newCards.txt**, which contains an arbitrary number of lines. Each line indicates a Card's type, name, health, and power, each separated by a single space. Example: `Dragon Zeta 151 25` – Using the data in the input file, the corresponding Cards are created and placed in the GameMaster's ArrayList of Cards. These are the cards that should be used for the game.

<u>Add-On 3: Simple GUI</u> (+15 max)
Develop a simple graphical user interface (GUI) for the Card Game. Assuming the base requirements for the project have been properly implemented, the development of the GUI should work well with the existing implementation of Card, Player, and GameMaster. This is a GUI version of the text-only GameConsole. This does **<u>not</u>** replace the GameConsole program.

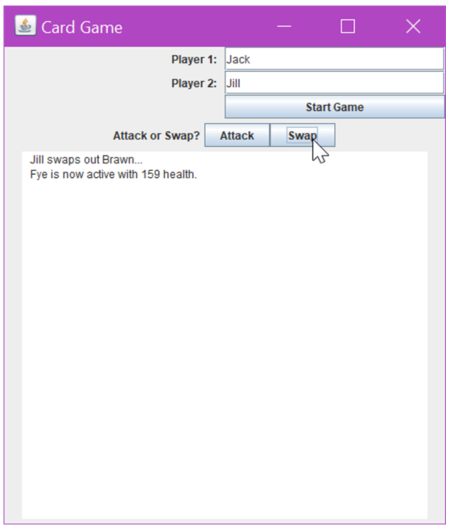
- **SimpleGUI.java** – contains a JFrame that serves as the main window of the graphical user interface for the GameMaster; hence it creates a GameMaster object, as well as the Player objects necessary for the Card Game. It calls methods on the GameMaster object when events occur. The constructor creates the components of the game. Additional methods may be defined that aid in creating GUI components and handling events. The SimpleGUI must enable the user to do the following:
  - specify the names of the two players
  - trigger the start of the game to enable the GameMaster to deal Cards to the Players until their hands are full
  - trigger an attack
  - trigger a swap
  - display the result of an attack or swap
  - display the game statistics when a winner is declared
- **SimpleApp.java** – declares and instantiates a SimpleGUI object and calls any methods that are needed to initialize the graphical user interface. The JFrame must be at most 800 x 600.

Shown below is an **example** of the Simple GUI.



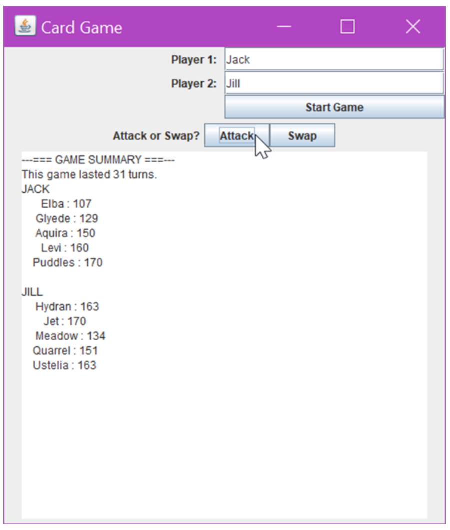
Player 1: Jack
Player 2: Jill

```
Jack attacks with Aquira.
  Ghost is weak to Dragon.
Aquira deals 52 damage on Brawn.
Brawn has 78 health left.
```



Player 1: Jack
Player 2: Jill

```
Jill swaps out Brawn...
Fye is now active with 159 health.
```



Player 1: Jack
Player 2: Jill

```
---=== GAME SUMMARY ===---
This game lasted 31 turns.
JACK
    Elba : 107
   Glyede : 129
   Aquira : 150
    Levi : 160
  Puddles : 170

JILL
   Hydran : 163
     Jet : 170
   Meadow : 134
  Quarrel : 151
  Ustelia : 163
```

<u>Add-On 4: Full GUI</u> (+25 max)
Instead of Add-On 3, develop a full graphical user interface (GUI) for the Card Game. Assuming the base requirements for the project have been properly implemented, the development of the GUI should work well with the existing implementation of Card, Player, and GameMaster. This is a more advanced GUI version than Add-On 3.

- **GameGUI.java** – contains a JFrame that serves as the main window of the graphical user interface for the GameMaster; hence it creates a GameMaster object, as well as the Player objects necessary for the Card Game. It calls methods on the GameMaster object when events occur. The constructor creates the components of the game. Additional methods may be defined that aid in creating GUI components and handling events. This GUI must contain the following features:
  - specify the names of the two players
  - trigger the start of the game to enable the GameMaster to deal Cards to the Players until their hands are full
  - a visual representation of each player's hand and their respective cards
  - click-triggers for all actions within the interface
  - display the result of an attack or swap, or any other action within the interface
  - display the game statistics when a winner is declared
- **GameApp.java** – declares and instantiates a GameGUI object and calls any methods that are needed to initialize the graphical user interface. The JFrame must be at most 800 x 600.

<u>Add-On X: Mystery Mode</u> (+?)
A student may also possibly receive higher marks for adding unique or rare features, without deviating from the required specifications.

<u>Project Defense (Code Review)</u>
Project defenses start on **06 December 2023**, which is a day before the deadline. Sign-ups will be announced on Canvas as the deadline approaches. Students who submit and defend **before** the deadline will receive a bonus of 5 points. There will be limited slots for this.

Grades for projects that are submitted beyond the deadline (Dec 07) ***<u>and</u>*** defended (by Dec 12) will be **capped at 74**.

Students who are unable to sign up for a defense schedule ***<u>due to lacking slots</u>*** are expected to contact their instructor immediately for possible alternative schedules. Students who do not sign up for a defense time slot ***<u>and</u>*** do not contact their instructor before the last day of project defenses (i.e. by Dec 11) forfeit the defense.

Students who fail to appear within the ***first five minutes*** of their selected time slot forfeits their project defense. There is no make-up defense, except in extenuating circumstances.

Grades for projects that are submitted beyond the last day of defenses (Dec 12) will be **capped at 65**, with the defense being forfeited. Students who do not submit their projects by **Thursday, 14 December 2023**, ***<u>might</u>*** receive a grade of *Incomplete* at the end of the semester, or ***<u>might</u>*** forfeit the project, depending on their class standing.

<u>Important Notes</u>
- Carefully and attentively read all specifications.
- Spell all words correctly, according to the sample output.
- Follow file naming conventions.
- Follow submission procedures.
- Do not submit any excess files.