Card Dek Project in Java

Here's a structured approach to implementing the card deck game in Java with a Graphical User Interface (GUI), simulating a deck of cards, including functionalities like random card drawing, sorting, and shuffling. Implement shuffling the deck before drawing cards in the Deck class. Provide methods for shuffling (e.g., Fisher-Yates), drawing cards, and checking deck size.

1. Card Representation: You can represent suits and ranks using enum

2. Deck Management and Shuffling: Create a Deck class that manages the deck of 52 cards. Include methods for shuffling (using the Fisher-Yates algorithm) and drawing cards.

3. Sorting with a Custom Comparator: You can define a custom Comparator to sort based on color, suit, and rank.

4. Graphical User Interface (GUI): For the GUI, we can use Java Swing to display the game.

Summary of Code Components

1. **Card Class**: Stores card properties (suit, rank).
2. **Deck Class**: Manages the deck, including shuffling and drawing.
3. **Card Comparator**: Custom sorting logic.
4. **CardGameGUI**: Java Swing GUI for displaying the game

Code Explanation

Suit.java

The code defines an enum named Suit in Java, representing the four suits in a deck of playing cards. Here's what it does:

1. **Enum Definition**:
    - The Suit enum contains four predefined constants: SPADES, CLUBS, HEARTS, DIAMONDS. Each of these corresponds to a suit in a deck of cards.

2. **Purpose**:
    - Enums are used in Java to represent a fixed set of constants. In this case, the Suit enum is used to define the four possible suits for a card.
3. **Usage**:
    - The Suit enum can be used in the Card class to assign a specific suit to each card (as seen in your earlier code). For example, a card can have Suit.SPADES or Suit.HEARTS.

Logic Summary:

The Suit enum provides a type-safe way to represent the four suits of cards (Spades, Clubs, Hearts, Diamonds) in the deck, ensuring only valid suits are used when creating a card.

Rank.java

The code defines an enum named Rank in Java, representing the ranks of playing cards. Here's what it does:

1. **Enum Definition**:
    - The Rank enum contains a predefined list of constants: ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING. Each of these represents a specific card rank.
2. **Purpose**:
    - Enums are used in Java to represent a fixed set of constants. In this case, Rank is used to model the ranks of playing cards.
3. **Usage**:
    - The Rank enum can be used in the Card class (as seen in your earlier code) to assign specific ranks to cards. For example, a card can have Rank.ACE or Rank.KING.

Logic Summary:

The Rank enum provides a type-safe way to represent the ranks of cards in the deck, ensuring that only valid ranks (Ace, Two, etc.) are used when creating a card.

Card.java

The code defines a Card class in Java, representing a playing card with two main properties: Suit and Rank. Here's a breakdown of the key components and logic:

1. **Attributes**:
   - suit: Represents the card's suit (Spades, Clubs, Hearts, Diamonds).
   - rank: Represents the card's rank (Ace, 2, 3, ..., King).
   - rankMap and suitMap: Static maps used to store the abbreviated string representations of card ranks (like "A" for Ace, "K" for King) and suits (like "S" for Spades, "H" for Hearts).
2. **Static Block**:
   - The static block initializes the rankMap and suitMap with appropriate abbreviations. This block runs once when the class is loaded.
3. **Constructor**:
   - The constructor takes a Suit and Rank to create a specific card with those attributes.
4. **Methods**:
   - getSuit() and getRank(): These are getter methods to access the suit and rank of the card.
   - getImagePath(): This method returns a formatted string representing the path to an image of the card. It uses the abbreviations from rankMap and suitMap to generate a filename like images/A-S.png for the Ace of Spades.
   - toString(): Overrides the toString() method to return a human-readable description of the card, such as "Ace of Spades."

Logic Summary:

The class creates a card object with a rank and a suit, provides methods to get these values, generates an image path based on the card’s attributes, and overrides toString() to return a readable description of the card.

Deck.java

This code defines a Deck class in Java, which simulates a deck of playing cards and provides functionality for shuffling, drawing cards, and managing the deck. Here's a breakdown of its components:

1. Attributes:

- cards: An ArrayList of Card objects representing the deck of cards.

2. Constructor:

- The constructor initializes the cards list with all possible card combinations using nested loops through the Suit and Rank enums. Each suit is paired with each rank to create a full deck of 52 cards.

3. Methods:

- **shuffle()**:
    - Implements the Fisher-Yates shuffle algorithm to randomize the order of the cards.
    - For each card from the end of the list to the beginning, a random index (j) is chosen, and the card at that index is swapped with the current card at index i.
    - Collections.swap() is used to swap two cards in the list.
- **drawCard()**:
    - Removes and returns the top card (last card in the list) from the deck. If the deck is empty, it returns null.
- **getDeckSize()**:
    - Returns the current number of cards left in the deck, useful for tracking the remaining cards.
- **drawMultipleCards(int num)**:
    - Draws multiple cards (up to num) by repeatedly calling drawCard(). It returns an ArrayList of drawn cards, stopping when the deck is empty or the requested number of cards is drawn.

Logic Summary:

This Deck class models a standard deck of 52 cards, supports shuffling using the Fisher-Yates algorithm for randomization, and allows drawing single or multiple cards from the deck. The deck is fully dynamic, with cards being removed as they are drawn, and the class can track the deck size in real-time.

CardComparator.java

This code defines a custom CardComparator class in Java that implements the Comparator<Card> interface. The compare method is used to compare two Card objects based on their color, suit, and rank.

1. **Color Precedence**:
    - The cards are divided into two "colors":
        - Red suits (HEARTS and DIAMONDS) are assigned a value of 1.
        - Black suits (SPADES and CLUBS) are assigned a value of 0.
    - The comparator first checks if the two cards have different colors (color1 vs color2). If the colors differ, the card with the lower color value (black) is considered smaller.
2. **Suit Precedence**:
    - If the colors are the same, the comparator compares the suits using Suit's natural ordering (compareTo). Suits are compared alphabetically by their enum order (CLUBS < DIAMONDS < HEARTS < SPADES), but you can adjust the order by changing the enum if needed.
3. **Rank Precedence**:
    - If both color and suit are the same, the cards are compared by rank. The Rank enum's compareTo method is used, which compares based on the order in which the ranks are defined (Ace is first, then Two through King).

Logic Summary:

The CardComparator first sorts cards by color (black suits before red suits). Within the same color, it sorts by suit, and finally, if the suit is the same, it sorts by rank. This provides a structured way to compare cards by multiple criteria for sorting purposes.

CardGameGUI.java

This code creates a Java Swing-based graphical user interface (GUI) for a card deck game using the Card, Deck, and CardComparator classes. The GUI allows users to draw, shuffle, sort, and restart the game. Here's a breakdown of its components and functionality:

1. Class Setup (CardGameGUI):

- Extends JFrame to create a GUI window.
- Uses a Deck to manage the cards and an ArrayList<Card> to store the drawn cards.
- The GUI contains:
    - A JTextArea (displayArea) to display messages.
    - A JPanel (imagePanel) to show drawn card images.
    - Buttons for drawing cards, shuffling the deck, sorting drawn cards, restarting the game, and showing remaining cards in the deck.

2. GUI Components:

- **drawButton**: Draws a card from the deck. Limits drawing to a maximum of 20 cards (MAX_CARDS). The drawn card is displayed in both text and image format.
- **shuffleButton**: Shuffles the deck and updates the display to indicate that the deck has been shuffled.
- **sortButton**: Sorts the drawn cards using the CardComparator, which sorts based on color, suit, and rank. The sorted cards are displayed with images.
- **restartButton**: Restarts the game by resetting the deck, clearing the drawn cards, and refreshing the display.
- **remainingCardsButton**: Displays the number of remaining cards in the deck.

3. Functional Highlights:

- **drawCard()**: Draws one card from the deck, updates the text display, and shows the corresponding card image. If the deck is empty or 20 cards have been drawn, a message is displayed.
- **shuffleDeck()**: Shuffles the deck using the Deck class and updates the display.
- **sortDrawnCards()**: Sorts the drawn cards using CardComparator and re-displays them.
- **displayCardImage()**: Displays an image of the drawn card. Images are resized to 100x140 pixels.
- **restartGame()**: Resets the deck, shuffles it, clears all drawn cards, and refreshes the GUI.
- **showRemainingCards()**: Displays the number of cards left in the deck.

4. Image Resizing (resizeImage):

- This method reads a card image from the file system, resizes it to 100x140 pixels, and returns the resized image. The images are stored in a directory and accessed using the card's rank and suit.

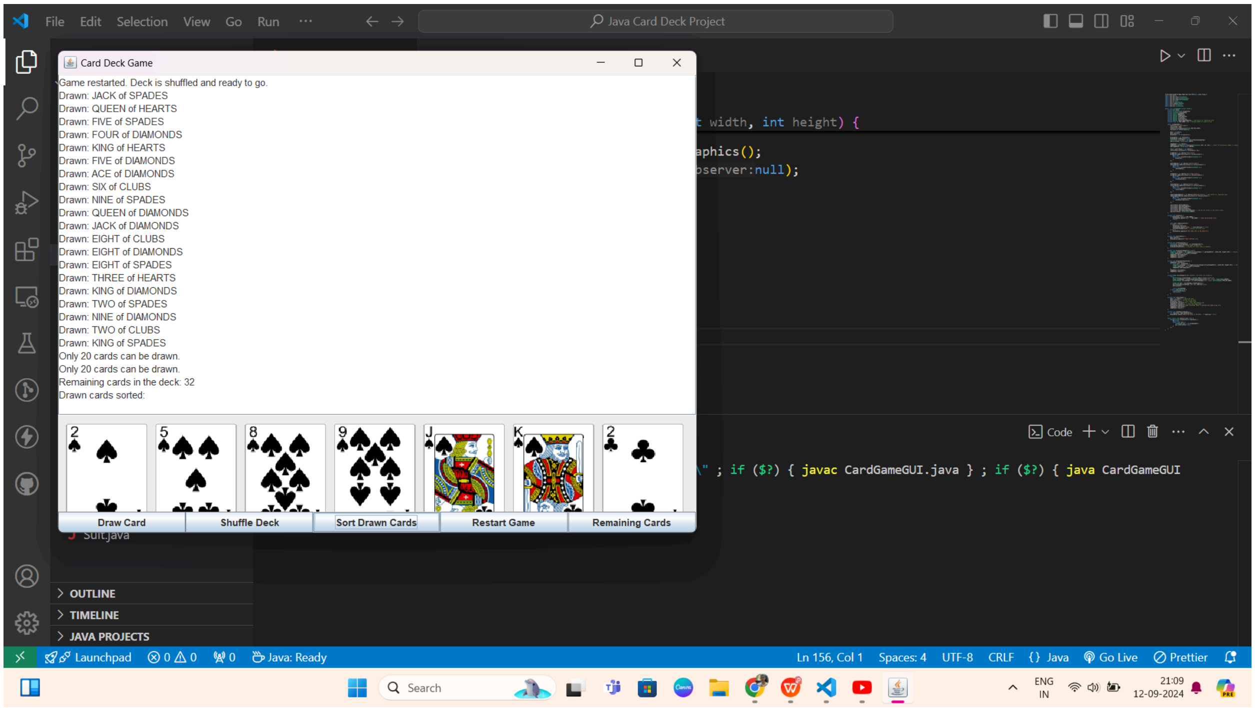
5. Main Method:

- The program starts with the main method, which creates and displays the CardGameGUI window using SwingUtilities.invokeLater() to ensure the GUI runs on the event-dispatching thread.

Logic Summary:

The CardGameGUI allows users to draw and display up to 20 cards from a shuffled deck, with the option to sort the drawn cards and restart the game. The game interface uses buttons to control drawing, shuffling, and sorting, and it provides visual feedback through card images and text messages.

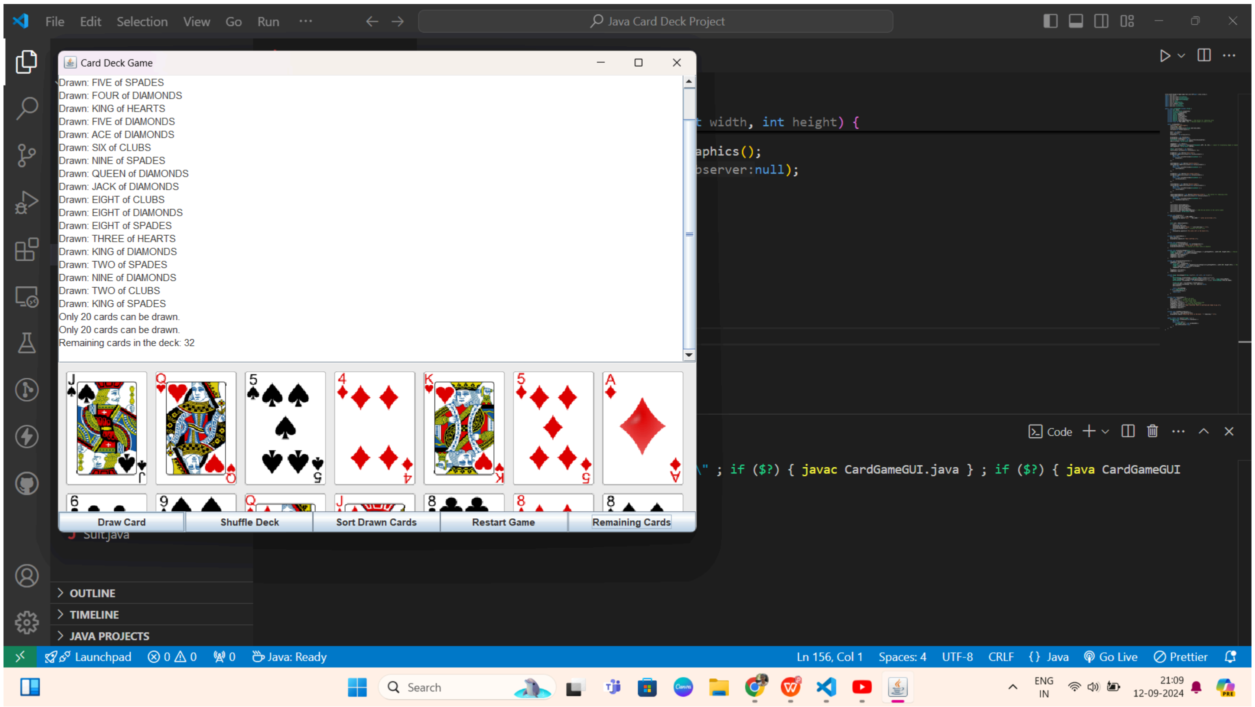
Output Images:



```
Game restarted. Deck is shuffled and ready to go.
Drawn: JACK of SPADES
Drawn: QUEEN of HEARTS
Drawn: FIVE of SPADES
Drawn: FOUR of DIAMONDS
Drawn: KING of HEARTS
Drawn: FIVE of DIAMONDS
Drawn: ACE of DIAMONDS
Drawn: SIX of CLUBS
Drawn: NINE of SPADES
Drawn: QUEEN of DIAMONDS
Drawn: JACK of DIAMONDS
Drawn: EIGHT of CLUBS
Drawn: EIGHT of DIAMONDS
Drawn: EIGHT of SPADES
Drawn: THREE of HEARTS
Drawn: KING of DIAMONDS
Drawn: TWO of SPADES
Drawn: NINE of DIAMONDS
Drawn: TWO of CLUBS
Drawn: KING of SPADES
Only 20 cards can be drawn.
Only 20 cards can be drawn.
Remaining cards in the deck: 32
Drawn cards sorted:
```

```
t width, int height) {
aphics();
bserver:null);
```

```
\" ; if ($?) { javac CardGameGUI.java } ; if ($?) { java CardGameGUI
```

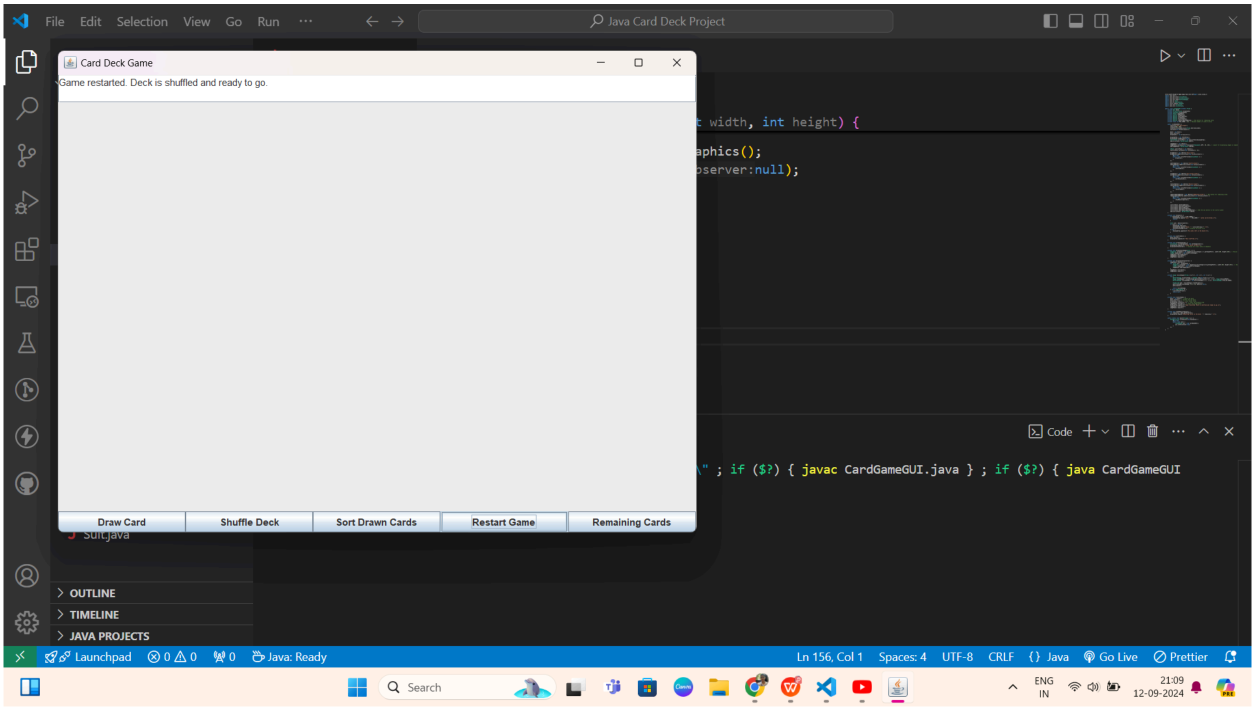


Card Deck Game

```
Drawn: FIVE of SPADES
Drawn: FOUR of DIAMONDS
Drawn: KING of HEARTS
Drawn: FIVE of DIAMONDS
Drawn: ACE of DIAMONDS
Drawn: SIX of CLUBS
Drawn: NINE of SPADES
Drawn: QUEEN of DIAMONDS
Drawn: JACK of DIAMONDS
Drawn: EIGHT of CLUBS
Drawn: EIGHT of DIAMONDS
Drawn: EIGHT of SPADES
Drawn: THREE of HEARTS
Drawn: KING of DIAMONDS
Drawn: TWO of SPADES
Drawn: NINE of DIAMONDS
Drawn: TWO of CLUBS
Drawn: KING of SPADES
Only 20 cards can be drawn.
Only 20 cards can be drawn.
Remaining cards in the deck: 32
```

```
t width, int height) {
aphics();
bserver:null);
```

```
\" ; if ($?) { javac CardGameGUI.java } ; if ($?) { java CardGameGUI
```

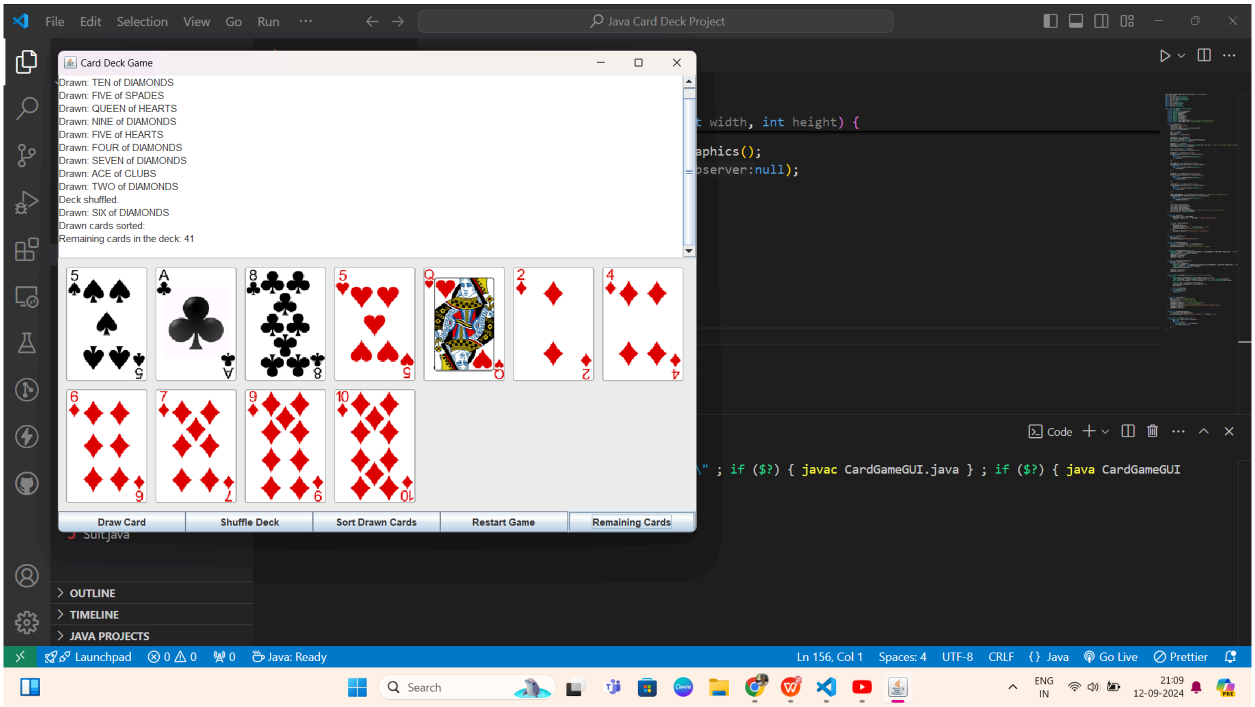


Card Deck Game

```
Game restarted. Deck is shuffled and ready to go.
```

```
t width, int height) {
aphics();
bserver:null);
```

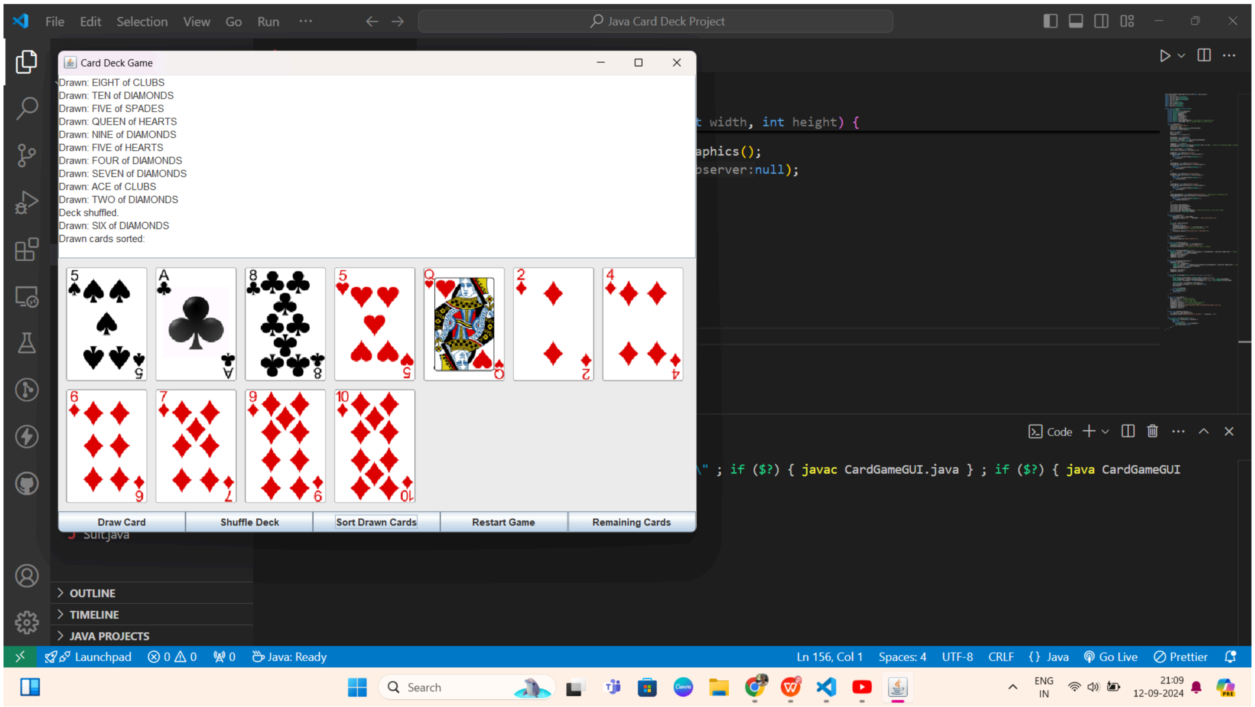
```
\" ; if ($?) { javac CardGameGUI.java } ; if ($?) { java CardGameGUI
```



```
Drawn: TEN of DIAMONDS
Drawn: FIVE of SPADES
Drawn: QUEEN of HEARTS
Drawn: NINE of DIAMONDS
Drawn: FIVE of HEARTS
Drawn: FOUR of DIAMONDS
Drawn: SEVEN of DIAMONDS
Drawn: ACE of CLUBS
Drawn: TWO of DIAMONDS
Deck shuffled.
Drawn: SIX of DIAMONDS
Drawn cards sorted:
Remaining cards in the deck: 41
```

```
t width, int height) {
aphics();
bserver:null);
```

```
\" ; if ($?) { javac CardGameGUI.java } ; if ($?) { java CardGameGUI
```



```
Drawn: EIGHT of CLUBS
Drawn: TEN of DIAMONDS
Drawn: FIVE of SPADES
Drawn: QUEEN of HEARTS
Drawn: NINE of DIAMONDS
Drawn: FIVE of HEARTS
Drawn: FOUR of DIAMONDS
Drawn: SEVEN of DIAMONDS
Drawn: ACE of CLUBS
Drawn: TWO of DIAMONDS
Deck shuffled.
Drawn: SIX of DIAMONDS
Drawn cards sorted:
```

```
t width, int height) {
aphics();
bserver:null);
```

```
\" ; if ($?) { javac CardGameGUI.java } ; if ($?) { java CardGameGUI
```