Card Deck Project

1. Project Overview

This project involves developing a Java application that simulates a deck of playing cards. The program is designed to manage a standard 52-card deck, allowing functionalities such as shuffling, drawing cards, and sorting the cards based on specific criteria. The goal is to provide a clear, object-oriented approach to card management using Java, with a focus on clean and maintainable code.

2. Technical Specifications

- **Language**: Java
- **IDE**: Eclipse
- **Java Version**: Java SE 8 or higher

3. Card Representation

The core of the simulation involves representing each card in the deck. This is achieved using a Card class that encapsulates the properties of a card—its suit and rank.

Card Class Implementation:

- Enums for Suit and Rank:
    - Suit: SPADE, CLUB, HEART, DIAMOND
    - Rank: A, 2-10, J, Q, K

Fields:

- Suit suit: Represents the suit of the card.
- Rank rank: Represents the rank of the card.

Methods:

- Card(Suit suit, Rank rank): Constructor to initialize the card's suit and rank.
- getSuit(): Returns the suit of the card.
- getRank(): Returns the rank of the card.
- toString(): Provides a string representation of the card (e.g., "A of HEART").

```
public class Card {
  public enum Suit {
    SPADE, CLUB, HEART, DIAMOND
  }

  public enum Rank {
    A, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, J, Q, K
  }

  private Suit suit;
  private Rank rank;

  public Card(Suit suit, Rank rank) {
    this.suit = suit;
    this.rank = rank;
  }

  public Suit getSuit() {
    return suit;
  }

  public Rank getRank() {
    return rank;
  }

  @Override
  public String toString() {
    return rank + " of " + suit;
  }
}
```

4. Deck Management

The Deck class is responsible for managing the entire deck of 52 cards. It provides functionalities to shuffle the deck, draw cards, and check the size of the deck.

Deck Class Implementation:

- **Fields**:
  - List<Card> cards: A list to hold all 52 cards.
- **Methods**:
  - Deck(): Initializes the deck by creating all possible combinations of suits and ranks.
  - shuffle(): Randomly shuffles the cards in the deck.
  - draw(): Draws a card from the deck, removing it from the list.
  - size(): Returns the number of remaining cards in the deck.

```
import java.util.ArrayList;

import java.util.Collections;

import java.util.List;


public class Deck {

  private List<Card> cards;


  public Deck() {

    cards = new ArrayList<>();

    for (Card.Suit suit : Card.Suit.values()) {

      for (Card.Rank rank : Card.Rank.values()) {

        cards.add(new Card(suit, rank));

      }

    }

  }


  public void shuffle() {

    Collections.shuffle(cards);
  }

  public Card draw() {
    if (cards.isEmpty()) {
      throw new IllegalStateException("Deck is empty");
    }
    return cards.remove(cards.size() - 1);
  }

  public int size() {
    return cards.size();
  }
}
```

5. Card Sorting

To enable sorting of cards based on custom criteria such as color, suit, and rank, a CardComparator class is implemented. This comparator is used in conjunction with Java's Collections.sort() method to order the drawn cards.

CardComparator Class Implementation:

- **Methods**:
  - compare(Card c1, Card c2): Defines the sorting logic:
    - Red cards (Hearts and Diamonds) are ordered before black cards (Spades and Clubs).
    - Within each color group, cards are sorted by suit and then by rank.

```
import java.util.Comparator;

public class CardComparator implements Comparator<Card> {
  @Override
  public int compare(Card c1, Card c2) {
    int colorCompare = Integer.compare(
      (c1.getSuit() == Card.Suit.HEART || c1.getSuit() == Card.Suit.DIAMOND) ? 1 : 0,
      (c2.getSuit() == Card.Suit.HEART || c2.getSuit() == Card.Suit.DIAMOND) ? 1 : 0
    );
    if (colorCompare != 0) {
      return colorCompare;
    }

    int suitCompare = c1.getSuit().compareTo(c2.getSuit());
    if (suitCompare != 0) {
      return suitCompare;
    }

    return c1.getRank().compareTo(c2.getRank());
  }
}
```

6. Main Application

The Main class ties all the components together. It simulates the card drawing and sorting process by drawing 20 random cards from the deck, displaying them unsorted, and then sorting them using the custom comparator.

Main Class Implementation:

- **Process**:
    - Shuffle the deck.
    - Draw 20 random cards into a list.
    - Display the unsorted list of cards.
    - Sort the list using Collections.sort() with CardComparator.
    - Display the sorted list of cards.

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Main {
  public static void main(String[] args) {
    Deck deck = new Deck();
    deck.shuffle();

    // Draw 20 random cards
    List<Card> drawnCards = new ArrayList<>();
    for (int i = 0; i < 20; i++) {
      drawnCards.add(deck.draw());
    }

    // Print unsorted cards
    System.out.println("Unsorted Cards:");
    for (Card card : drawnCards) {
      System.out.println(card);
    }

    // Sort cards
    Collections.sort(drawnCards, new CardComparator());

    // Print sorted cards
    System.out.println("\nSorted Cards:");
    for (Card card : drawnCards) {
      System.out.println(card);
    }
  }
}
```

7. Conclusion

This Java application successfully simulates a deck of cards, providing essential operations such as shuffling, drawing, and sorting based on custom criteria. The project follows an object-oriented design, using classes and enums to encapsulate the behavior and properties of cards and the deck. The use of a custom comparator allows flexible sorting of cards, making the application extensible for further enhancements.