Drew Sweeney

Homework4

CSC341

I had a harder time getting through this homework, probably because of some outside factors in my life. Some of the harder concepts to get through in this homework for me was dealing with the player choosing a suit when a 8 was played. Another difficulty I faced was trying to renew the stockpile after all cards had been used but players still had cards in their hand. Instead, I just declared the game inconclusive and finished the game. In this assignment I tried to make my assignment #2 more effective so instead of having class Card have all the Cards I made two more classes called Suit and Rank where I store the values and names of the cards. To me this seemed to be a more effective way to run the program, but I am not actually sure if it does make it more efficient with design or not.

Player:

```java
package homework4;

import java.util.*;

public class Player
{
	public List<Card> hand;
    private int cardsFromStock;
    private int cardsUsed;
    private String name;

    public Player(String name)
    {
        hand = new ArrayList<>();
        cardsFromStock = 0;
        cardsUsed = 0;
        this.name = name;
    }

    public void add(Card card)
    {
        hand.add(card);
    }

    public void playedCard()
    {
        cardsUsed++;
    }

    public void playedCard(Card card)
    {
        hand.remove(card);
        cardsUsed++;
    }

    public void pickedFromStock()
    {
        cardsFromStock++;
    }

    public int getCardsPickedFromStock()
    {
        return cardsFromStock;
    }

    public int getCardsPlayed()
    {
        return cardsUsed;
    }

    public int getRemainingCardsInHand()
    {
        return hand.size();
    }

    public boolean hasNoMoreCards()
    {
        return hand.size() == 0;
    }

    public boolean hasCrazyEights()
    {
        long count = hand.stream().filter(card ->
card.getRank().getName().equals(Rank.EIGHT.getName())).count();

        return count == 2;
    }

    public String getName()
    {
        return name;
    }

    public void showHand()
    {
        for (int i = 0; i < hand.size(); i++)
        {
            System.out.println(hand.get(i));
        }
    }
}
```

Card:

```
package homework4;

import java.awt.*;

public class Card
{
      private Suit suit;
    private Rank rank;
    private Color suitColor;

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

    public Suit getSuit()
    {
      return suit;
    }

    public Rank getRank()
    {
      return rank;
    }

    public int getValue()
    {
      return rank.getValue();
    }

    public Color getSuitColor()
    {
      return suitColor;
    }

    public String toString()
    {
      return String.format("[%s of %s]", rank.getName(), suit.getName());
    }
}
```

Suit:

```
package homework4;

import java.awt.*;

public class Card
{
        private Suit suit;
    private Rank rank;
    private Color suitColor;

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

    public Suit getSuit()
    {
      return suit;
    }

    public Rank getRank()
    {
      return rank;
    }

    public int getValue()
    {
      return rank.getValue();
    }

    public Color getSuitColor()
    {
      return suitColor;
    }

    public String toString()
    {
      return String.format("[%s of %s]", rank.getName(), suit.getName());
    }
}
```

Rank:

```
package homework4;

public enum Rank
```

```
{
	ACE(0, "ACE"),
    TWO(2, "TWO"),
    THREE(3, "THREE"),
    FOUR(4, "FOUR"),
    FIVE(5, "FIVE"),
    SIX(6, "SIX"),
    SEVEN(7, "SEVEN"),
    EIGHT(8, "EIGHT"),
    NINE(9, "NINE"),
    TEN(10, "TEN"),
    JACK(11, "JACK"),
    QUEEN(12, "QUEEN"),
    KING(13, "KING");

	private int value;
	private String name;

	Rank (final int v, final String n)
	{
		value = v;
		name = n;
	}

	public int getValue()
	{
		return value;
	}

	public String getName()
	{
		return name;
	}
}
```

Deck:

package homework4;

import java.util.*;

import static homework4.Rank.*;

```
import static homework4.Suit.*;


public class Deck

{

        private static HashMap<Integer, Suit> Suits = new HashMap<>();

        private static HashMap<Integer, Rank> Ranks = new HashMap<>();



        private static final int numOfSuits = 4;

        private static final int cardsPerSuit = 13;



        static {

          Suits.put(0, CLUBS);

    Suits.put(1, SPADES);

    Suits.put(2, HEARTS);

    Suits.put(3, DIAMONDS);


    Ranks.put(0, ACE);
    Ranks.put(1, TWO);

    Ranks.put(2, THREE);

    Ranks.put(3, FOUR);

    Ranks.put(4, FIVE);

    Ranks.put(5, SIX);

    Ranks.put(6, SEVEN);

    Ranks.put(7, EIGHT);

    Ranks.put(8, NINE);

    Ranks.put(9, TEN);

    Ranks.put(10, JACK);

    Ranks.put(11, QUEEN);

    Ranks.put(12, KING);


        }


        public Stack<Card> cards;

        public Deck()
        {
            cards = new Stack<>();
    for (int i = 0; i < numOfSuits; i++)
    {
      Suit current_suit = Suits.get(i);
      for (int j = 0; j < cardsPerSuit; j++)
      {
        Rank rank = Ranks.get(j);
          Card card = new Card(current_suit, rank, null);
          cards.push(card);
            }
          }
        }

        public Stack<Card> getCards()
        {
```

```
		return cards;

	}

	

	public void shuffle()

	{

		Collections.shuffle(cards);

	}

	

	public void showDeck()

	{

		cards.forEach(System.out::println);

	}

	

}
```

CrazyEights:

```
package homework4;

import java.util.*;

public class CrazyEightsGame
{
	public static void play(Deck deck)
	{
		int cardsForStart = 7;

		boolean isPlayer1Turn = false;
		boolean playing = true;

		Stack<Card> cardsBeingUsed = new Stack<>();

		Player player1 = new Player("Drew"), player2 = new Player("Dr. Hu");

		System.out.println("Before Starting lets shuffle the deck");
		deck.shuffle();

		System.out.println("This is the deck of cards");
		deck.showDeck();

		for (int i = 0 ; i < cardsForStart; i++)
		{
			player1.add(deck.cards.pop());
		}

		System.out.println(player1.getName() + "s cards");
		player1.showHand();

		for (int i = 0 ; i < cardsForStart; i++)
		{
			player2.add(deck.cards.pop());
		}

		System.out.println(player1.getName() + "s cards");
		player2.showHand();

		cardsBeingUsed.push(deck.cards.pop());

		System.out.println("The starting card is: " + cardsBeingUsed.peek());

		System.out.println("Number of cards left in deck is: " +
deck.cards.size());

		System.out.println(player1.getName() + " plays first");

		while(playing)
		{
			if (player1.hasNoMoreCards())
			{
				playing = false;
				winCondition(player1, player2, deck, cardsBeingUsed);

			}
			else if (player2.hasNoMoreCards());
			{
				playing = false;
				winCondition(player2, player1, deck, cardsBeingUsed);
			}
```

```
				{
					if (isPlayer1Turn)
					{
						takeTurn(player1, cardsBeingUsed, deck);
						isPlayer1Turn = false;
					}
					else
					{
						takeTurn(player2, cardsBeingUsed, deck);
						isPlayer1Turn = true;
					}

				}
			}
		}

		private static void takeTurn(Player player, Stack<Card> cardsBeingUsed, Deck
deck)
		{
			boolean isTherePlayableCard = false;
			for (int i = 0; i < player.hand.size(); i++)
			{
				Card card = player.hand.get(i);

				if
(card.getRank().getName().equals(cardsBeingUsed.peek().getRank().getName()) ||
card.getSuit().getName().equals(cardsBeingUsed.peek().getSuit().getName()))
				{
					isTherePlayableCard = true;
					cardsBeingUsed.push(card);
					player.playedCard(card);
					System.out.println(player.getName() + " played " + card +
player.getName() + "s cards left: " + player.getRemainingCardsInHand());
					break;
				}
			}

			if (!isTherePlayableCard)
			{
				if (deck.cards.size() < 1)
				{
					maintainStockPile(cardsBeingUsed, deck);
				}

				Card stockPileCard = deck.cards.pop();
				System.out.println("Stockpile cards left: " + deck.cards.size()
+ " - Cards in the play pile: " + cardsBeingUsed.size());
				if
(stockPileCard.getRank().getName().equals(cardsBeingUsed.peek().getRank().getName())
||
stockPileCard.getSuit().getName().equals(cardsBeingUsed.peek().getSuit().getName()))
				{
					cardsBeingUsed.push(stockPileCard);
```

```
                              System.out.println(player.getName() + " got lucky from the 
stockpile and played " + stockPileCard);
                              player.playedCard();
                         }
                         else
                         {
                               System.out.println(player.getName() + " could not play, 
had to keep stockpile card");
                               player.add(stockPileCard);
                               player.pickedFromStock();
                         }
                  }
           }

           private static void maintainStockPile(Stack<Card> cardsBeingUsed, Deck deck)
           {
                  System.out.println("\n" + "needed to re-fill card 
stockpile".toUpperCase());
                  Card lastCardPlayed = cardsBeingUsed.pop();
                  deck.cards.addAll(cardsBeingUsed);
                  deck.shuffle();
                  cardsBeingUsed.clear();
                  cardsBeingUsed.push(lastCardPlayed);
           }

           private static void winCondition(Player p1, Player p2, Deck deck, Stack<Card> 
cardsBeingUsed)
           {
                  System.out.println("________________GAME OVER________________");
                  System.out.println(p1.getName() + " Wins!");

                  System.out.println("PLAYER STATS");

               System.out.println(p1.getName() + " played " + p1.getCardsPlayed() + " 
card(s). Played " + p1.getCardsPickedFromStock() + " card(s) picked from the 
stockpile");
               System.out.println(p1.getName() + " has " + p1.getRemainingCardsInHand() + " 
card(s) left in their hand");

               System.out.println(p2.getName() + " played " + p2.getCardsPlayed() + " 
card(s). Played " + p2.getCardsPickedFromStock() + " card(s) picked from the 
stockpile");
               System.out.println(p2.getName() + " has " + p2.getRemainingCardsInHand() + " 
card(s) left in their hand");
               System.out.println("________________GAME OVER________________");
           }

}
```