Chap14Yourname.java

Objects of objects

```
/*
 * Chap14 Objects of objects
 * with CardCollection.java
 * with Deck2.java
 * with Hand.java
 * with Player.java
 * with Eights.java
 */

import java.util.Random;
import java.util.ArrayList;
import java.util.Arrays;
import java.io.PrintStream;
```

```
public class Chap14 {
	public static void main(String[] args) {
		PrintStream out = System.out;

		//By now we can create
		//a Card(int rank, int suit) which is an object,
		//a Card[] which is an array of objects,
		//a Deck(int n) which is an object of Card[], or an object of arrays.

		//In this chapter we will make a game: Crazy Eights.
		//Play it online - https://cardgames.io/crazyeights/
		//Our rules is simpler:
```

```
//1. The main objective is to be the first player to get rid of all your
//    cards.
//2. Deal 5 cards to each player to create a "hand" for each player.
//    Deal one card face up to create the "discard pile".
//    Place the remaining cards face down to create the "draw pile".
//    A hand, the discard pile and the draw pile should can be displayed.
//3. Each player takes turns placing a single card on the discard pile.
//    The card must match the rank or suit of the last card on the discard pile,
//    or an eight, which is a "wild card".
//4. When players don't have a matching card or an eight, they must draw
//    new cards until they get one.
//5. If the draw pile ever runs out, the discard pile is shuffled except the
//    last card and becomes the new draw pile.
//6. As soon as a player has no cards, the game ends and all other players
//    score penalty points for their remaining cards. Eights are worth 20,
//    face cards (Jacks, Queens and Kings) are worth 10, and all others are
//    worth their rank.
```

14.1 & 14.2 Decks and hands. CardCollection

```
out.println("14.1 & 14.2 Decks and hands. CardCollection");
//We need to create a deck of cards, a discard pile, a draw pile, and
//a hand for each player. And we need to deal, draw, and discard cards.

//Deck, hands and piles have different sizes, and their sizes change as the
//game progresses. We solve this problem by using an ArrayList, which is in
//the java.util package. An ArrayList is a collection, which is an object
//that contains other objects, so an ArrayList is an object of objects.
//It provides methods to add and remove elements, and it grows and shrinks
//automatically.

//Learn to create the CardCollection class.
out.println();
```

CardCollection.java

```java
/*
 * With Chap14.java
 */

import java.util.ArrayList;
import java.util.Random;
```

CardCollection.java

```java
public class CardCollection {

    public String label;
    public ArrayList<Card> cards;
    //cards is an ArrayList of Card objects, so it is an object of objects.

    public CardCollection(String label) {
        this.label = label;
        this.cards = new ArrayList<Card>();
    }
```

CardCollection.java

```
//……The getters:……


public String getLabel() {
	return label;
}

public Card getCard(int i) {
	return cards.get(i);
}
//ArrayList provides the get(i) method. This is a wrapper method for get(i).
```

CardCollection.java

```
//The last card is the most frequently used one in the CardCollection.
//When a card is placed to the discard pile by a player, it should match the
//    last card in the discard pile.
//When a player needs to draw a card, he/she can only draw the last card in the
//    draw pile.
//So we need a method getting the last card.
public Card last() {
    int i = size() - 1;
    return cards.get(i);
}
//This is not a wrapper method.
```

CardCollection.java

```
public int size() {
      return cards.size();
}
//This is a wrapper method for size(), which is provided by the ArrayList.

//Create the empty() method indicating whether size() is zero.
//When a player's hand is empty, the game is over.
//When the draw pile is empty, the discard pile will be shuffled except the last
//    card and becomes the new draw pile.
public boolean empty() {
      return cards.size() == 0;
}
//This is not a wrapper method.
```

CardCollection.java

```
//......The add and remove methods:......



//We need to be able to add cards to the collection.
//ArrayList provides the add method that adds an element to the collection:
public void addCard(Card card) {
      cards.add(card);
}
//This is a wrapper method for add.
```

CardCollection.java

```
//We also need to remove cards from the collection.
//ArrayList provides the remove method that takes an index, removes the card
//at that location, and shifts the following cards left to fill the gap:
public Card popCard(int i) {
    return cards.remove(i);
}
//This is a wrapper method for remove.
```

CardCollection.java

```java
//If we are dealing cards from a shuffled deck, it's most convenient to remove
//the last card. Here is an overloaded version of popCard:
public Card popCard() {
    int i = size() - 1;
    return popCard(i);
}
//This is not a wrapper method since it's a little complicated.
```

CardCollection.java

```
//The deal method removes n cards from this CardCollection and adds them to
//another CardCollectino that.
public void deal(int n, CardCollection that) {
	for (int i = 0; i < n; i++) {
		Card card = popCard();
		that.addCard(card);
	}
}

//The dealAll method deals all of the cards of this to that.
public void dealAll(CardCollection that) {
	for (int i = 0; i < this.size(); i++) {
		Card card = popCard();
		that.addCard(card);
	}
}
```

CardCollection.java

```
//……The shuffle methods:……


//ArrayList provides the set(i, element) method.
public void swapCards(int i, int j) {
      Card temp = cards.get(i);
      cards.set(i, cards.get(j));
      cards.set(j, temp);
}
//This is not a wrapper method.
```

CardCollection.java

```
public void shuffle() {
    Random random = new Random();
    for (int i = size() - 1; i > 0; i--) {
        int j = random.nextInt(i);    //a random integer in [0, i - 1]
        swapCards(i,j);
    }

//The CardCollection class is completed.
}
```

14.3 & 14.4 Inheritance

```
out.println("14.3 Inheritance");
//The CardCollection class provides the common features of hands and piles.
//But Deck, hands and piles have some different features.

//The deck should have a constructor that makes a standard 52-card ArrayList.

//A hand, the draw pile and the discard pile should can be displayed. They do
//        not need to be standard 52-card ArrayLists. They have the same features and
//        differ from that of the deck.

//Create the Deck2 class inheriting from the CardCollection class.
```

Deck2.java

```
/*
 * With Chap14.java
 */

public class Deck2 extends CardCollection {
	//This means a Deck2 object has the same instance variables and methods as a
	//CardCollection. Deck2 inherits from CardCollection. Constructors are not
	//inherited. CardCollection is a superclass, and Deck2 is one of its subclass.

	//In Java, classes may only extend one superclass. Classes that do not specify
	//a superclass automatically inherit from java.lang.Object.
```

Deck2.java

```
    public Deck2(String label) {
        super(label);
        //This invokes the constructor of the superclass.

        for (int suit = 0; suit <= 3; suit++) {
            for (int rank = 1; rank <= 13; rank++) {
                cards.add(new Card(rank, suit));
            }
        }
        //A Deck2 inherits instance variables and methods from CardCollection and
        //provides a different constructor.
    }
}
```

14.3 & 14.4 Inheritance

```
Deck2 deck = new Deck2("Deck");

//Create the Hand class inheriting from the CardCollection class.
```

Hand.java

```
/*
 * With Chap14.java
 */

public class Hand extends CardCollection {

    public Hand(String label) {
        super(label);
    }
    //A hand inherits instance variables and methods from CardCollection and has the
    //    same constructor.
```

Hand.java

```
    public void display() {
        System.out.println(getLabel() + ": ");
        for (int i = 0; i < size(); i++) {
            System.out.println(getCard(i));
        }
        System.out.println();
    }
    //A Hand provides an additional method, display().
}
```

14.3 & 14.4 Inheritance

```
//We can create hands, a draw pile, a discard pile using the Hand class.
Hand hand1 = new Hand("Hand1");
Hand drawPile = new Hand("Draw pile");
out.println();
```

14.3 & 14.4 Inheritance

```
//Get familiar with the objects and methods.
out.println(deck.getLabel());
out.println(deck.size());
out.println(deck.empty());
out.println(deck.getCard(0));
out.println(deck.getCard(1));
out.println(deck.getCard(50));
out.println(deck.getCard(51));
out.println();

deck.shuffle();
```

14.3 & 14.4 Inheritance

```
//Deal five cards to hand1 and the rest into the drawPile.
deck.deal(5, hand1);
hand1.display();

deck.dealAll(drawPile);
drawPile.display();
```

14.5 The Player class

```
out.println("14.5 The Player class");

//We are able to create a deck of cards, a discard pile, a draw pile, and
//a hand for each player. We are able to deal cards.

//Now we need to create a Player object who follows the Rule 3 and 4 of
//playing and drawing cards. When the game ends, a Player computes penalty
//points for cards left in his or her hand.
//A Player has two private attributes: a name and a hand.
```

Player.java

```
/*
 * With Chap14.java
 */

import java.util.Scanner;

public class Player {
	private String name;
	private Hand hand;
	private Scanner in;

	public Player(String name) {
		this.name = name;
		this.hand = new Hand(name);
	}
```

14.6 The Eights class

```
out.println("14.6 The Eights class");

//Then we will need an Eights class that has two Players, a drawPile and
//a discardPile as private variables.
//According to Rule 3 it should make the players take turns.
//According to Rule 5 if the draw pile ever runs out, it should shuffle the
//          discard pile except the top card and generates the new draw pile.
//According to Rule 6 it need to check whether the game is over.
```

Eights.java

```
/*
 * With Chap14.java
 */
import java.util.Scanner;

public class Eights {
	private Player p1;
	private Player p2;
	private Hand drawPile;
	private Hand discardPile;
	private Scanner in;
```

Eights.java

```java
public Eights() {
    //Create the deck and shuffle it.
    Deck2 deck = new Deck2("Deck");
    deck.shuffle();

    //Deal 5 cards to each player('s hands).
    p1 = new Player("DeepDog the AI Player Version 1.37");
    deck.deal(5, p1.getHand());
    p2 = new Player("You the human player");
    deck.deal(5, p2.getHand());
    //(Turn to Player.java to create the getHand() method.)
```

Player.java

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

public Hand getHand() {
      return this.hand;
}
//(Turn to Eight.java to continue to create the discardPile)
```

Eights.java

```
        //Create the discardPile with 1 card.
        discardPile = new Hand("Dicard pile");
        deck.deal(1, discardPile);

        //Create the drawPile with the rest cards of the deck.
        drawPile = new Hand("Draw pile");
        deck.dealAll(drawPile);

        //To involve the user in the game:
        in = new Scanner(System.in);
    }
```

Eights.java

```
//According to Rule 6 we need to check whether the game is over.
public Boolean gameOver() {
    return p1.getHand().empty() || p2.getHand().empty();
}
```

Eights.java

```
//According to Rule 5 if the draw pile ever runs out, the discard pile should
//    be shuffled except the top card and becomes the new draw pile.
//We encapsulate this rule into the draw method, which will be invoked by the
//    Player class.
public Card draw() {
    if (drawPile.empty()) {
        Card temp = discardPile.popCard();
        discardPile.shuffle();
        discardPile.dealAll(drawPile);
        discardPile.addCard(temp);
    }
    return drawPile.popCard();
}
//(Turn to Player.java to create the Play method.)
```

Player.java

```
//According to Rule 3, when a player places a single card on the discard pile,
//    the card must match the rank or suit of the top card on the discard pile,
//    or it is an eight, which is a "wild card".
//According to Rule 4, when a player does not have a matching card or an eight,
//    he/she must draw new cards until they get one.
//We encapsulate these rules into the autoPlay method.
public Card autoPlay(Eights eights, Card prev) {
    Card card = searchForMatch(prev);
    if (card == null) {
        card = drawForMatch(eights, prev);
    }
    return card;
}
```

Player.java

```
public Card searchForMatch(Card prev) {
    for (int i = 0; i < hand.size(); i++) {
        Card card = hand.getCard(i);
        if (cardMatch(card, prev)) {
            return hand.popCard(i);
        }
    }
    return null;
}
```

Player.java

```
public boolean cardMatch(Card card, Card prev) {
    if (card.getSuit() == prev.getSuit() ||
            card.getRank() == prev.getRank() ||
            card.getRank() == 8) {
        return true;
    }
    return false;
}
```

Player.java

```
public Card drawForMatch(Eights eights, Card prev) {
    while (true) {
        Card card = eights.draw();
        System.out.println(name + " draws " + card);
        if (cardMatch(card, prev)) {
            return card;
        }
        hand.addCard(card);
    }
}
```

Player.java

```
//According to Rule 6, As soon as the game ends, all players get penalty points
//for their remaining cards. Eights are worth 20, face cards
//(Jacks, Queens and Kings) are worth 10, and all others are worth their rank.
//The winner get penalty point of 0.
```

Player.java

```
public int score() {
    int score = 0;
    for (int i = 0; i < hand.size(); i++) {
        if (hand.getCard(i).getRank() > 10) {
            score += 10;
        } else {
            if (hand.getCard(i).getRank() == 8) {
                score += 20;
            } else {
                score += hand.getCard(i).getRank();
            }
        }
    }
    return score;
}
```

Player.java

```
	//The Player class is completed. Go back to Eights.java to create the takeTurn
	//method.
}
```

Eights.java

```
//According to Rule 3 we should make the players take turns.
//Now it is time to create a method to start the game.
```

Eights.java

```
public void displayState() {
    System.out.println("-----------------State of the game--------------------");
    p1.getHand().display();
    p2.getHand().display();
    System.out.println("Discard pile: " + discardPile.size() + " cards with this"
            + " one as the last: " + discardPile.last());
}
```

Eights.java

```
    public void autoGame() {
        Player player = p2;
        Card prev;
        Card next;

        while(!gameOver()) {
            displayState();
            System.out.println("It's " + player.getName() + "'s turn to play.");

            prev = discardPile.last();
            next = player.autoPlay(this, prev);
            System.out.println(player.getName() + " plays " + next);
            System.out.println();
            discardPile.addCard(next);
```

Eights.java

```
            if (player == p1) {
                player = p2;
            } else {
                player = p1;
            }
        }

        System.out.println("Game Over.");
        System.out.println(p1.getName() + " get a score of " + p1.score());
        System.out.println(p2.getName() + " get a score of " + p2.score());
    }

    //The Eights.java is completed. Return to Chap14.java to start a game.
}
```

14.6 The Eights class

```
                out.println("Let's start a game.");
                out.println();
                Eights crazyEight = new Eights();
                crazyEight.autoGame();
                out.println();

                //The autoGame() is played by the computer with itself.
                //Now create a method game() in the Eights.java and
                //a method userPlay(Eights eights, Card prev) then start a real game.
                Eights realGame = new Eights();
                realGame.game();
        }
}
```