```java
/**
 * This is the root of Poker game
 * It validates input and saves input into data structure
 * It checks winner and print output
 */

import java.util.ArrayList;

public class Poker {
    public static void main(String[] args) {
        // If inputs are not available sets (set of 5), throw error
        if (args.length % 5 > 0 || args.length <= 0) {
            System.out.println("Error: wrong number of arguments; " +
                    "must be a multiple of 5");
            System.exit(1);
        }

        //players list for holding card sets (hands)
        ArrayList<Player> players = new ArrayList<Player>();

        //Assign cards to players
        ArrayList<Card> tempCard = new ArrayList<Card>();
        for (int i = 0; i < args.length; i++) {
            tempCard.add(new Card(args[i].toUpperCase()));
            if (tempCard.size() == 5) {
                //Create a new player and save to player list
                Hand tempHand = new Hand(tempCard);
                Player tempPlayer = new Player(i/5 + 1, tempHand);
                players.add(tempPlayer);

                //print out player information
                System.out.printf("Player %d: %s\n",
                        tempPlayer.getId(), tempPlayer.getHandDescription());

                //clear card list so it's ready for next set
                tempCard.clear();
            }
        }

        //find winner if have more than 1 players
        if(players.size() > 1) {
            ArrayList<Player> winners = getWinners(players);
            //print out result
            int size = winners.size();
            if (size > 1) {
                printDrawResult(size, winners);
            } else {
                System.out.printf("Player %d wins.\n", winners.get(0).getId());
            }
        }
    }

    /**
     * get list of winners
     * @param allPlayers
     * @return list of winners
     */
    private static ArrayList<Player> getWinners(ArrayList<Player> allPlayers){
        ArrayList<Player> winners = new ArrayList<Player>();
        for (int i = 0; i < allPlayers.size(); i++) {
            Player currentPlayer = allPlayers.get(i);

            //Set a default winner if no winner exists
            if (winners.size() == 0) {
                winners.add(currentPlayer);
            } else {
                Player winnerPlayer = winners.get(0);
```

```
68                      String result = currentPlayer.compete(winnerPlayer);
69                      //update winner based on compete result
70                      if (result == "Draw") {
71                          winners.add(currentPlayer);
72                      } else {
73                          if (result == "Win") {
74                              winners.clear();
75                              winners.add(currentPlayer);
76                          }
77                          //do nothing if challenge winner failed
78                      }
79                  }
80              }
81              return winners;
82          }
83
84          /**
85           * print draw result to console
86           * @param size
87           * @param winners
88           */
89          private static void printDrawResult(int size, ArrayList<Player> winners){
90              System.out.print("Players ");
91              for (int i = 0; i < size - 2; i++) {
92                  System.out.printf("%d, ", winners.get(i).getId());
93              }
94              System.out.printf("%d and %d draw.\n",
95                      winners.get(size - 2).getId(),
96                      winners.get(size - 1).getId());
97          }
98      }
99
100
101     /**
102      * This is the Class for Player
103      * It saves player id and its hand
104      * Provides function for compare player's hand
105      */
106
107     import java.util.ArrayList;
108
109     public class Player {
110         private int id;
111         private Hand hand;
112
113         /**
114          * initialize player
115          * @param id  player id
116          * @param hand  (card set)
117          */
118         public Player(int id, Hand hand) {
119             //player id
120             this.id = id;
121             this.hand = hand;
122         }
123
124         /**
125          * get player id, the order of card set
126          * @return id
127          */
128         public int getId() {
129             return this.id;
130         }
131
132         /**
133          * get hand description (explain hand and rank)
134          * @return description
```

```
135          */
136         public String getHandDescription() {
137             return this.hand.getDescription();
138         }
139 
140         /**
141          * decide winner(draw) between two cards
142          * @param second
143          * @return compare result
144          */
145         public String compete(Player second) {
146             //firstly, decide winner by hand category
147             if (this.hand.getCategory() < second.hand.getCategory()) {
148                 return "Win";
149             } else if (this.hand.getCategory() > second.hand.getCategory()) {
150                 return "Lose";
151             } else {
152                 //if they have same category, proceed to value compare
153                 ArrayList<Integer> list1 = this.hand.getRanksForCompare();
154                 ArrayList<Integer> list2 = second.hand.getRanksForCompare();
155 
156                 //Compare each rank until get a winner
157                 for (int i = 0; i < list1.size(); i++) {
158                     if (list1.get(i) > list2.get(i)) {
159                         return "Win";
160                     } else if (list1.get(i) < list2.get(i)) {
161                         return "Lose";
162                     }
163                 }
164                 //if still no winner, then draw
165                 return "Draw";
166             }
167         }
168 
169     }
170 
171 
172     /**
173      * This is the Class for Hand (set of 5 cards)
174      * It saves cards and hand information
175      * Including hand category and description
176      * Also detailed attribute like sequence, same suits and pairs etc
177      */
178 
179     import java.util.ArrayList;
180     import java.util.Collections;
181 
182     public class Hand {
183         //it's id represents a hand category
184         private int category;
185         //the description of hand category, includes rank
186         private String description;
187         //cards of this set
188         private ArrayList<Card> cards;
189 
190         //detailed attribute
191         //is all 5 cards a sequence
192         private boolean sequence = true;
193         //is all 5 cards have same suit
194         private boolean oneSuit = true;
195         //one card from a four of a kind set
196         private Card quadruple = null;
197         //one card from a three of a kind set
198         private Card triple = null;
199         //a list of pair, each represented by one card from the pair
200         private ArrayList<Card> pairs = new ArrayList<Card>();
201 
```

```
202         //hand category dictionary (the smaller, the winner)
203         private final String straightFlush = "0";
204         private final String fourOfAKind = "1";
205         private final String fullHouse = "2";
206         private final String flush = "3";
207         private final String straight = "4";
208         private final String threeOfAKind = "5";
209         private final String twoPair = "6";
210         private final String onePair = "7";
211         private final String highCard = "8";
212 
213         /**
214          * Initial Hand
215          * @param cards
216          */
217         public Hand(ArrayList<Card> cards) {
218             //sort the card by rank in descending order
219             Collections.sort(cards, Collections.reverseOrder());
220             this.cards = new ArrayList<Card>(cards);
221             organizeCards();
222 
223             //get hand type id and description
224             String[] handDetails = calcHandDetails();
225             this.category = Integer.parseInt(handDetails[0]);
226             this.description = handDetails[1];
227         }
228 
229         /**
230          * get category id
231          * @return category id
232          */
233         public int getCategory() {
234             return this.category;
235         }
236 
237         /**
238          * get hand description, in a sentence
239          * @return hand description
240          */
241         public String getDescription() {
242             return this.description;
243         }
244 
245         /**
246          * get the ranks for compare based on hand category
247          * @return list of rank values
248          */
249         public ArrayList<Integer> getRanksForCompare() {
250             //for each case
251             //get required check value first, then rest of values
252             //result is in the order to compare
253             ArrayList<Integer> compareList = new ArrayList<Integer>();
254             switch (this.category) {
255                 case 0: //straightFlush
256                 case 4: //straight
257                     compareList.add(this.cards.get(0).getValue());
258                     break;
259                 case 1: //fourOfAKind
260                     compareList.add(this.quadruple.getValue());
261                     compareList.addAll(this.getDifferentValues(
262                             this.quadruple.getValue(), -1));
263                     break;
264                 case 2: //fullHouse
265                     compareList.add(this.triple.getValue());
266                     compareList.add(this.pairs.get(0).getValue());
267                     break;
268                 case 3: //flush
```

```
                case 8: //highCard
                    compareList.addAll(this.getDifferentValues(-1, -1));
                    break;
                case 5: //threeOfAKind
                    compareList.add(this.triple.getValue());
                    compareList.addAll(this.getDifferentValues(
                            this.triple.getValue(), -1));
                    break;
                case 6: //twoPair
                    compareList.add(this.pairs.get(0).getValue());
                    compareList.add(this.pairs.get(1).getValue());
                    compareList.addAll(this.getDifferentValues(
                            this.pairs.get(0).getValue(),
                            this.pairs.get(1).getValue()));
                    break;
                case 7: //onePair
                    compareList.add(this.pairs.get(0).getValue());
                    compareList.addAll(this.getDifferentValues(
                            this.pairs.get(0).getValue(), -1));
                    break;
                default:
                    break;
            }
            return compareList;
        }

        /**
         * get hand's detail information
         * including is sequence, is same suit, full/three/two same ranks
         */
        private void organizeCards() {
            //compare card with it's next card
            for (int i = 0; i < this.cards.size() - 1; i++) {
                Card temp1 = this.cards.get(i);
                Card temp2 = this.cards.get(i + 1);

                if (!temp1.isAdjacent(temp2)) {
                    this.sequence = false;
                }

                if (!temp1.isSameSuit(temp2)) {
                    this.oneSuit = false;
                }

                if (temp1.isSameRank(temp2)) {
                    storeSameRankCards(temp1);
                }
            }
        }

        /**
         * save card to pair/triple/quadruple based on time of appearance
         * @param second
         */
        private void storeSameRankCards(Card second) {
            int existedPairIndex = -1;
            int newValue = second.getValue();

            //get pair index that have same rank
            for (int i = 0; i < this.pairs.size(); i++) {
                if (this.pairs.get(i).getValue() == newValue) {
                    existedPairIndex = i;
                }
            }
            //if a pair with same rank existed, then save it as triple
            if (existedPairIndex != -1) {
                this.pairs.remove(existedPairIndex);
```

```
                this.triple = second;
            } else {
                //if a triple with same rank existed, then save it as quadruple
                if (this.triple != null && this.triple.getValue() == newValue) {
                    this.quadruple = second;
                    this.triple = null;
                } else {
                    //if not a pair yet, save as pair
                    this.pairs.add(second);
                }
            }
        }

        /**
         * decide hand category and description based on hand properties
         * @return [hand category id, hand description]
         */
        private String[] calcHandDetails() {
            //cards.get(0) will get largest rank, because it's sorted
            //all card same suit
            if (this.oneSuit) {
                //if 5 cards are in sequence
                if (this.sequence) {
                    return new String[]{this.straightFlush,
                            this.cards.get(0).getRank() + "-high straight flush"};
                } else {
                    return new String[]{this.flush,
                            this.cards.get(0).getRank() + "-high flush"};
                }
            }

            //sequence but not same suit
            if (this.sequence) {
                return new String[]{this.straight,
                        this.cards.get(0).getRank() + "-high straight"};
            }

            //if four same rank
            if (this.quadruple != null) {
                return new String[]{this.fourOfAKind,
                        "Four " + this.quadruple.getRank() + "s"};
            }

            //if three same rank
            if (this.triple != null) {
                //if there is a pair
                if (this.pairs.size() > 0) {
                    return new String[]{this.fullHouse,
                            this.triple.getRank() + "s full of " +
                                    this.pairs.get(0).getRank() + "s"};
                } else {
                    return new String[]{this.threeOfAKind,
                            "Three " + this.triple.getRank() + "s"};
                }
            }

            //if there are pairs
            if (this.pairs.size() > 0) {
                if (this.pairs.size() > 1) {
                    return new String[]{this.twoPair,
                            this.pairs.get(0).getRank() + "s over " +
                                    this.pairs.get(1).getRank() + "s"};
                } else {
                    return new String[]{this.onePair,
                            "Pair of " + this.pairs.get(0).getRank() + "s"};
                }
            }
```

```
403
404             return new String[]{this.highCard,
405                     this.cards.get(0).getRank() + "-high"};
406         }
407
408         /**
409          * get values that different to provided value
410          * @param val1
411          * @param val2
412          * @return [up to 5 rank values]
413          */
414         private ArrayList<Integer> getDifferentValues(int val1, int val2) {
415             ArrayList compareList = new ArrayList();
416             this.cards.forEach(result -> {
417                 if (result.getValue() != val1 && result.getValue() != val2) {
418                     compareList.add(result.getValue());
419                 }
420             });
421             return compareList;
422         }
423     }
424
425
426
427     /**
428      * This is the Class for Cards
429      * It interprets card input to suit and rank (and its value)
430      * It provide function for compare cards
431      */
432
433     /**
434      * names of suits
435      */
436     enum Suits {
437         Clubs,
438         Diamonds,
439         Hearts,
440         Spades,
441     }
442
443     //implement comparable for sortable ArrayList
444     public class Card implements Comparable<Card> {
445         private Suits suit;
446         private String rank;
447         // rank value (integer version of rank)
448         private int value;
449
450         /**
451          * initialize Card info
452          * @param input
453          */
454         public Card(String input) {
455             //initial object with input
456             this.initialSuit(input.substring(1, 2));
457             this.initialRank(input.substring(0, 1));
458
459             //if failed to get suit or rank, throw error
460             if (this.suit == null || this.rank == null) {
461                 System.out.printf("Error: invalid card name '%s'\n", input);
462                 System.exit(1);
463             }
464         }
465
466         /**
467          * get value of rank
468          * @return rank value
469          */
```

```
470         public int getValue() {
471             return this.value;
472         }
473 
474         /**
475          * get name of rank
476          * @return rank
477          */
478         public String getRank() {
479             return this.rank;
480         }
481 
482         /**
483          * get name of suit
484          * @return suit
485          */
486         public Suits getSuit() {
487             return this.suit;
488         }
489 
490         /**
491          * override compareTo for using Collection ArrayList sort
492          * @param second, the other card for compare
493          * @return difference
494          */
495         @Override
496         public int compareTo(Card second) {
497             return this.value - second.getValue();
498         }
499 
500         /**
501          * check if two cards have same suit
502          * @param second, the other card for compare
503          * @return compare result
504          */
505         public boolean isSameSuit(Card second) {
506             return this.suit == second.getSuit();
507         }
508 
509         /**
510          * check if two cards have same rank
511          * @param second, the other card for compare
512          * @return compare result
513          */
514         public boolean isSameRank(Card second) {
515             return this.value == second.getValue();
516         }
517 
518         /**
519          * check if two cards is adjacent
520          * @param second, the other card for compare
521          * @return compare result
522          */
523         public boolean isAdjacent(Card second) {
524             return Math.abs(this.value - second.getValue()) == 1;
525         }
526 
527         /**
528          * interpret suit name with input, return null if no match
529          * @param info
530          */
531         private void initialSuit(String info) {
532             this.suit = null;
533             switch (info) {
534                 case "C":
535                     this.suit = Suits.Clubs;
536                     break;
```

```
                case "D":
                    suit = Suits.Diamonds;
                    break;
                case "H":
                    suit = Suits.Hearts;
                    break;
                case "S":
                    suit = Suits.Spades;
                    break;
                default:
                    this.suit = null;
                    break;
            }
        }

        /**
         * interpret input to rank name and value, return null if no match
         * @param info
         */
        private void initialRank(String info) {
            switch (info) {
                case "2":
                case "3":
                case "4":
                case "5":
                case "6":
                case "7":
                case "8":
                case "9":
                    this.rank = info;
                    this.value = Integer.parseInt(info);
                    break;
                case "T":
                    this.rank = "10";
                    this.value = 10;
                    break;
                case "J":
                    this.rank = "Jack";
                    this.value = 11;
                    break;
                case "Q":
                    this.rank = "Queen";
                    this.value = 12;
                    break;
                case "K":
                    this.rank = "King";
                    this.value = 13;
                    break;
                case "A":
                    this.rank = "Ace";
                    this.value = 14;
                    break;
                default:
                    this.rank = null;
                    break;
            }
        }
    }

```