```
/*This project implements the classification of A poker hand.
   A poker hand consists fives cards,which means five input args for this programme.
   Each card has two attributes,rank and suit.
   Rank,in ascending order,includes 2,3,4,5,6,7,8,9,T(10),J(Jack),Q(Queen),K(King),A(Ace).
   Suit consists of C(Clubs),D(Diamonds),H(Hearts),S(Spades).
   There are nine classifications for hands,in descending order to be:
   Straight flush,Four of a kind,Full house,Flush,Straight,Three of a kind,Two pair,One pair,High card.
   Poker.java mainly judges whether the input is valid and sorts the valid rank and suit.
*/
import java.util.Arrays;

public class Poker {
	public static void main(String[] args) {
		Poker input_copy=new Poker();
		input_copy.IsCorrectInput(args);
		input_copy.ArraySort(args);
	}

	/**
	 * Determine if the input is a valid card by checking whether rank and suit are in desired range.
	 * @param args:includes five args.Each one has two parts,rank and suit.
	 */
	private void IsCorrectInput(String []args){
		final int A = 10;
		final int T = 29;
		final int J = 19;
		final int Q = 26;
		final int K = 20;
		final int C=12;
		final int D=13;
		final int H=17;
		final int S=28;

		if(args.length>5){
			System.out.println("NOT UNDERTAKEN");
			System.exit(1);
		}
		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);
		}

		for (String arg : args) {
			int rank = Character.getNumericValue(arg.charAt(0));
```

```
            int suit = Character.getNumericValue(arg.charAt(1));
            switch (rank) {
                case A:
                case T:
                case J:
                case Q:
                case K:
                case 2:
                case 3:
                case 4:
                case 5:
                case 6:
                case 7:
                case 8:
                case 9:
                    break;
                default:
                    System.out.printf("Error: invalid card name '%s'\n",arg);
                    System.exit(1);
            }
            switch (suit){
                case C:
                case D:
                case H:
                case S:
                    break;
                default:
                    System.out.printf("Error: invalid card name '%s'\n",arg);
                    System.exit(1);
            }
        }
}

/**
 * Divide a valid input(card) into two arrays(RankArray,SuitArray). And sort them separately.
 * @param args:includes five args.Each one has two parts,rank and suit.
 */
private void ArraySort(String []args){
        int[] RankArray =new int[5];
        String[] SuitArray = new String[5];

        Player card_copy=new Player();

        for(int index=0;index<5;index++){
```

```
            String card=args[index];
            char[] ch=card.toCharArray();
            RankArray[index]=card_copy.RankDecoder(ch[0]);
            SuitArray[index]=String.valueOf(ch[1]).toUpperCase();
        }
        Arrays.sort(RankArray);
        Arrays.sort(SuitArray);

        new Player(RankArray, SuitArray);
    }


}
```

```
/*Poker.java calls Player.java. And Player.java determines the type of hand for input.
 */
class Player {
    Player(){}

    Player(int []RankArray, String []SuitArray ){
        CardClassificationOutput(RankArray, SuitArray);
    }

    /**
     * Determine which of the nine categories a hand poker belongs to.
     * After that, programme will jump to the corresponding classification output according to the return value.
     * @param RankArray:Rank value in ascending order.
     * @param SuitArray:Suit value in ascending order.
     * @return a value which helps programme jump to corresponding output in CardClassificationOutput.
     */
    private int CardClassification(int[] RankArray, String[] SuitArray) {
        if (IsStraightFlush(RankArray,SuitArray)) { return 1; }
        else if (IsFourOfAKind(RankArray)) { return 2; }
        else if (IsFullHouse(RankArray)) { return 3; }
        else if (IsFlush(SuitArray)) { return 4; }
        else if (IsStraight(RankArray)) { return 5; }
        else if (IsThreeOfAKind(RankArray)) { return 6; }
        else if (IsTwoPair(RankArray)) { return 7; }
        else if (IsOnePair(RankArray)) { return 8; }
        else { return 9; }
    }

    /**
     * print the hand classification result.
     * @param RankArray:Rank value in ascending order.
     * @param SuitArray:Suit value in ascending order.
     */
    private void CardClassificationOutput(int[] RankArray, String[] SuitArray) {
        String[] RankArrayCoder = RankCoder(RankArray);
        switch (CardClassification(RankArray, SuitArray)) {
            case 1:
                System.out.printf("Player 1: %s-high straight flush\n", RankArrayCoder[4]);
                break;
            case 2:
                if (RankArray[0] == RankArray[3]) {
                    System.out.printf("Player 1: Four %ss\n", RankArrayCoder[0]);
                } else {
                    System.out.printf("Player 1: Four %ss\n", RankArrayCoder[4]);
```

```
        }
        break;
    case 3:
        if (RankArray[0] == RankArray[2]) {
            System.out.printf("Player 1: %ss full of %ss\n", RankArrayCoder[0], RankArrayCoder[4]);
        } else {
            System.out.printf("Player 1: %ss full of %ss\n", RankArrayCoder[4], RankArrayCoder[0]);
        }
        break;
    case 4:
        System.out.printf("Player 1: %s-high flush\n", RankArrayCoder[4]);
        break;
    case 5:
        System.out.printf("Player 1: %s-high straight\n", RankArrayCoder[4]);
        break;
    case 6:
        if (RankArray[0] == RankArray[2]) {
            System.out.printf("Player 1: Three %ss\n", RankArrayCoder[0]);
        } else if (RankArray[2] == RankArray[4]) {
            System.out.printf("Player 1: Three %ss\n", RankArrayCoder[4]);
        } else {
            System.out.printf("Player 1: Three %ss\n", RankArrayCoder[2]);
        }
        break;
    case 7:
        if(RankArray[0]==RankArray[1]){
            if(RankArray[2]==RankArray[3]){
                System.out.printf("Player 1: %ss over %ss\n",RankArrayCoder[2],RankArray[0]);
            }
            else{
                System.out.printf("Player 1: %ss over %ss\n",RankArrayCoder[3],RankArray[0]);
            }
        }
        else{
            System.out.printf("Player 1: %ss over %ss\n",RankArrayCoder[3],RankArray[1]);
        }
        break;
    case 8:
        for(int index=0;index<4;index++){
            int index_next=index+1;
            if(RankArray[index]==RankArray[index_next]){
                System.out.printf("Player 1: Pair of %ss\n",RankArrayCoder[index]);
                break;
            }
```

```
                }
                break;
            case 9:
                System.out.printf("Player 1: %s-high\n",RankArrayCoder[4]);
                break;
        }
}

/**
 * If the rank entered is a character, convert it to a number.
 * If the rank is a number, it will output directly.
 * @param rank
 * @return the value converted from the rank input.
 */
int RankDecoder(char rank){
        int rank_decoder = 0;

        if (!Character.isDigit(rank)) {
            char rank_upcase = Character.toUpperCase(rank);
            if (rank_upcase == 'T') {
                rank_decoder = 10;
            } else if (rank_upcase == 'J') {
                rank_decoder = 11;
            } else if (rank_upcase == 'Q') {
                rank_decoder = 12;
            } else if (rank_upcase == 'K') {
                rank_decoder = 13;
            } else if (rank_upcase == 'A') {
                rank_decoder = 14;
            }
            return rank_decoder;
        } else {
            rank_decoder = Integer.parseInt(String.valueOf(rank));
        }
        return rank_decoder;
}
/**
 * @param RankArray: Rank in ascending order.
 * @return character or number converted from corresponding rank value
 */
private String[] RankCoder(int[] RankArray){
        String[] RankArrayCoder = new String[5];
        for (int index = 0; index < 5; index++) {
            if (RankArray[index] < 11) {
```

```
                RankArrayCoder[index] = String.valueOf(RankArray[index]);
            }
            else if (RankArray[index] == 11) {
                RankArrayCoder[index] = "Jack";
            }
            else if (RankArray[index] == 12) {
                RankArrayCoder[index] = "Queen";
            }
            else if (RankArray[index] == 13) {
                RankArrayCoder[index] = "King";
            }
            else if (RankArray[index] == 14) {
                RankArrayCoder[index] = "Ace";
            }
        }
        return RankArrayCoder;
}


//The following 9 methods determine which of the nine categories a hand poker belongs to.
private boolean IsStraightFlush(int[] RankArray, String[] SuitArray){
        return IsStraight(RankArray) && IsFlush(SuitArray);
}

private boolean IsFourOfAKind(int[] RankArray){
        int []count=IsNOfAKind(RankArray);
        return count[0] == 4 || count[1] == 4;
}

private boolean IsFullHouse(int[] RankArray){
        int []count=IsNOfAKind(RankArray);
        return count[0] + count[1] == 5;
}

/**
  * One of the core three methods.It aims to determine if it is flush.
  * @param SuitArray:Suit value in ascending order.
  * @return
  */
private boolean IsFlush(String[] SuitArray){
        return SuitArray[0].equals(SuitArray[4]);
}

/**
```

```
 * One of the core three methods.It aims to determine if it is straight.
 * @param RankArray:Rank value in ascending order.
 * @return
 */
private boolean IsStraight(int[] RankArray){
	int index = 0;
	for (; index < 4; index++) {
		int index_next = index + 1;
		if (!(RankPlusOne(RankArray[index]) == RankArray[index_next])) {
			break;
		}
	}
	return index == 4;
}

private boolean IsThreeOfAKind(int[] RankArray){
	int []count=IsNOfAKind(RankArray);
	return count[0] == 3 || count[1] == 3 ||count[2] == 3;
}

private boolean IsTwoPair(int[] RankArray){
	int []count=IsNOfAKind(RankArray);
	return count[0] + count[1] +count[2]== 5;
}

private boolean IsOnePair(int[] RankArray){
	int []count=IsNOfAKind(RankArray);
	int index = 0;
	for (; index < 4; index++) {
		if (count[index]==2) {
			return true;
		}
	}
	return false;
}

/**
 * One of the core three methods.It aims to determine how many cards in the hand are the same.
 * @param RankArray:  Rank value in ascending order.
 * @return
 */
private int []IsNOfAKind(int []RankArray){
	int []count = {1,1,1,1,1};
	int i=0;
```

```
        for (int j = 0; j < 4; j++) {
            for (int k = j + 1; k < 5; k++) {
                if ((RankArray[j] == RankArray[k])) {
                    count[i]++;
                }
                else { break; }
            }
            int bound = j + count[i];
            if (bound > 3) { break; }
            else {
                j = bound - 1;
                i++;
            }
        }
        return count;
    }

    //small tool,rank value plus one.
    private int RankPlusOne(int Rank){
        return Rank + 1;
    }
}
```