```
void main() {
    Deck d = getNewDeckOfCards();
    printDeck(d);
    shuffleDeck(d);
    printDeck(d);
    sortDeck(d);
    printDeck(d);
    shuffleDeck(d);

    for(int i = 0 ; i < d.cards.length - 5; i = i+ 5) {
        Hand h = getHand(d.cards[i], d.cards[i+1], d.cards[i+2], d.cards[i+3], d
.cards[i+4]);
        printHand(h);
    }

    testHands();
}

enum Rank {
    Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King, Ace
;
}

enum Suit {
    Heart, Spade, Club, Diamond;
}

class Card {
    Rank rank;
    Suit suit;
}

void printCard( Card c ) {
    print(c.rank);
    print(" of ");
    print(c.suit);
    print("s");
}

class Deck {
    Card[] cards;
}

Deck getNewDeckOfCards() {
    Card[] cards = new Card[52];
    int i = 0;
    for(Suit s = Suit.Heart; s != null; s = enumSucc(s) ) {
        for(Rank r = Rank.Two; r != null; r = enumSucc(r) ) {
            Card c;
            c.rank = r;
            c.suit =s;
            cards[i] = c;
            i = i+1;
        }
    }

    Deck d;
    d.cards = cards;
    return d;
}

void printDeck(Deck d) {
```



```
    for(int i = 0; i < d.cards.length ; i++ ){
        printCard(d.cards[i]);
        if(i < d.cards.length -1) {
            print(", ");
        }
    }
    println();
}

void shuffleDeck(Deck d) {
    Card[] cards = d.cards;

    for(int i = 0; i < cards.length; i++ ) {
        int j = i + floor(random() * (cards.length - i));
        swapCards(cards, i, j);
    }
}

void swapCards(Card[] cards, int i, int j) {
    Card temp = cards[i];
    cards[i] = cards[j];
    cards[j] = temp;
}

enum HandClassifier {
    HighCard, Pair, TwoPair, ThreeOfAKind, Straight, Flush, FullHouse, FourOfAKi
nd, StraightFlush, RoyalFlush;
}

class Hand {
    //class invariant: there are exactly 5 cards
    Card[] cards;
    HandClassifier classification;
}

void printHand(Hand h) {
    print("Hand is: ");
    print(h.classification);
    print("( ");
    for(int i = 0 ; i < h.cards.length ; i++ ) {
        printCard(h.cards[i]);
        if(i < h.cards.length - 1) {
            print(", ");
        }
    }
    print(")");
    println();
}

Hand getHand(Card a, Card b, Card c, Card d, Card e) {
    Card[] handCards = {a,b,c,d,e};
    sortCards(handCards);

    Hand hand;
    hand.cards = handCards;
    int[] straightCounts = getStraightCounts(hand.cards);

    if(isStraightFlush(handCards)) {
        if(handCards[4].rank == Rank.Ace) {
            hand.classification = HandClassifier.RoyalFlush;
        } else {
            hand.classification = HandClassifier.StraightFlush;
```

```
        }
    } else if(isFourOfAKind(straightCounts, handCards)) {
        hand.classification = HandClassifier.FourOfAKind;
    } else if(isFullHouse(straightCounts, handCards)) {
        hand.classification = HandClassifier.FullHouse;
    } else if(isFlush(handCards)) {
        hand.classification = HandClassifier.Flush;
    } else if(isStraight(handCards)) {
        hand.classification = HandClassifier.Straight;
    } else if(isThreeOfAKind(straightCounts, handCards)) {
        hand.classification = HandClassifier.ThreeOfAKind;
    } else if(isTwoPair(straightCounts, handCards)) {
        hand.classification = HandClassifier.TwoPair;
    } else if(isPair(straightCounts, handCards)) {
        hand.classification = HandClassifier.Pair;
    } else /* Its a high card */ {
        hand.classification = HandClassifier.HighCard;
    }

    return hand;
}


void sortCards( Card[] cards ) {
    sortCards(cards, 0, cards.length);
}

void sortCards(Card[] cards, int low, int high) {
    if( low == high) {
        return;
    }

    int pivot = (low + high) / 2;
    swapCards(cards, low, pivot);
    int newHigh = partition(cards, low+1, high, low);
    swapCards(cards, newHigh - 1, low);

    sortCards(cards, low, newHigh - 1);
    sortCards(cards, newHigh, high);

}

int partition(Card[] cards, int low, int high, int pivot) {
    while(low < high) {
        if(compareCards(cards[low], cards[pivot]) == Ord.GreaterThan) {
            high = high - 1;
            swapCards(cards, low, high);
        } else {
            low = low + 1;
        }
    }
    return high;
}

Ord compareCards(Card a, Card b) {
    int aOrd = ordinal(a.rank);
    int bOrd = ordinal(b.rank);
    if(aOrd < bOrd) {
        return Ord.LessThan;
    } else if( aOrd == bOrd) {
        return Ord.Equal;
    } else{
```



```
        return Ord.GreaterThan;
    }
}

enum Ord {
    LessThan, GreaterThan, Equal;
}

int ordinal(Rank r) {
    int rval = 0;
    for(Rank rr = Rank.Two; rr != r; rr = enumSucc(rr)) {
        rval = rval + 1;
    }
    return rval;
}

void sortDeck(Deck d) {
    sortCards(d.cards);
}

/* Properties of these methods: cards are sorted
if boolean returns true; card order may have been altered.
Private methods only to getHand
*/

void testIsStraightFlush() {
    Deck d = getNewDeckOfCards();
    {
        Card[] cards = { d.cards[0], d.cards[1], d.cards[2], d.cards[3], d.cards
[4] };
        assert isStraightFlush(cards);
    }
    {
        Card[] cards = { d.cards[8], d.cards[9], d.cards[10], d.cards[11], d.car
ds[12] };
        assert isStraightFlush(cards);
    }
    {
        Card[] cards = { d.cards[0], d.cards[0], d.cards[0], d.cards[0], d.cards
[0] };
        assert !isStraightFlush(cards);
    }
}

boolean isStraightFlush(Card[] cards){
    return (isStraight(cards) && isFlush(cards));
}

boolean isFourOfAKind(int[] straightCounts, Card[] cards) {
    Rank foursRank = null;
    Rank onesRank = null;
    Rank currentRank = Rank.Two;
    for(int i = 0 ; i < straightCounts.length ; i++) {
        switch(straightCounts[i]) {
            case 0:{ break; }
            case 1: {
                if(onesRank == null) {
                    onesRank = currentRank;
                } else {
                    return false;
                }
                break;
```

```
            }
            case 4: {
                if(foursRank == null) {
                    foursRank = currentRank;
                } else {
                    return false;
                }
                break;
            }
            default: {
                return false;
            }
        }
        currentRank = enumSucc(currentRank);
    }

    for(int i = 0 ; i < cards.length ; i++ ) {
        if(cards[i].rank == onesRank) {
            swapCards(cards,0,i);
            return true;
        }
    }

    assert false : "Should be Unreachable!";
    return false;

}

boolean isFullHouse(int[] straightCounts, Card[] cards) {
    Rank threesRank = null;
    Rank twosRank    = null;

    Rank currentRank = Rank.Two;

    for(int i = 0 ; i < straightCounts.length ; i++ ) {
        switch(straightCounts[i]) {
            case 0: {
                /* ok */
                break;
            }
            case 2: {
                if(twosRank == null ) {
                    twosRank = currentRank;
                } else {
                    return false;
                }
                break;
            }
            case 3: {
                if(threesRank == null){
                    threesRank = currentRank;
                } else {
                    return false;
                }
                break;
            }
            default : {
                return false;
            }
        }
        currentRank = enumSucc(currentRank);
    }
```



```
    int l = 0;
    int r = 5;
    while(l < r) {
        if(cards[l].rank == threesRank) {
            r = r - 1;
            swapCards(cards,l,r);
        } else {
            l = l + 1;
        }
    }
    return true;
}

boolean isFlush(Card[] cards) {
    Suit expSuit = cards[0].suit;
    for(int i = 1; i < 5 ; i++ ) {
        if( cards[i].suit != expSuit) {
            return false;
        }
    }
    return true;
}

boolean isStraight(Card[] cards) {
    Rank expRank = enumSucc(cards[0].rank);
    if(expRank == null) {
        expRank = Rank.Two;
    }

    for(int i = 1; i < 5 ; i++ ) {
        if( expRank == null ||
            cards[i].rank != expRank) {
            return false;
        }
        expRank = enumSucc(expRank);
    }

    return true;
}

boolean isThreeOfAKind(int[] straightCounts, Card[] cards) {
    Rank threesRank = null;
    Rank currentRank = Rank.Two;
    for(int i = 0 ; i < straightCounts.length ; i++ ) {
        if(straightCounts[i] == 3) {
            threesRank = currentRank;
        }
        currentRank = enumSucc(currentRank);
    }

    if(threesRank == null) {
        return false;
    }

    if(cards[4].rank != threesRank) {
        swapCards(cards, 1, 4);
    }
    if(cards[3].rank != threesRank) {
        swapCards(cards, 0, 3);
    }
```

```
    return true;
}

boolean isTwoPair(int[] straightCounts, Card[] cards) {
    Rank[] pairRanks = new Rank[2];
    int cRank = 0;

    Rank currentRank = Rank.Two;
    for(int i = 0; i < straightCounts.length ; i++ ) {
        if(straightCounts[i] == 2) {
            pairRanks[cRank] = currentRank;
            cRank = cRank + 1;
        }
        currentRank = enumSucc(currentRank);
    }

    if(cRank != 2) {
        return false;
    }

    if(cards[4].rank != pairRanks[1]) {
        swapCards(cards, 4, 2 );
    }

    if(cards[2].rank != pairRanks[0]) {
        swapCards(cards, 0, 2);
    }

    return true;
}

boolean isPair(int[] straightCounts, Card[] cards) {
    Rank pairRank = null;
    Rank currentRank = Rank.Two;
    for(int i = 0 ; i < straightCounts.length ; i++ ) {
        if(straightCounts[i] == 2) {
            pairRank = currentRank;
        }
        currentRank = enumSucc(currentRank);
    }

    if(pairRank == null) {
        return false;
    }

    int pairStart = -1;

    for(int i = 0 ; i < cards.length ; i++ ){
        if(cards[i].rank == pairRank) {
            pairStart = i;
            break;
        }
    }

    for(int i = pairStart ; i < 3 ;i++ ) {
        swapCards(cards, i, i+2);
    }

    return true;
}

int[] getStraightCounts(Card[] cards) {
```



```
    int[] out = new int[13];
    for(int i = 0 ; i < cards.length ; i++) {
        int idx = ordinal(cards[i].rank);
        out[idx] = out[idx] + 1;
    }
    return out;
}


void testHands() {
    check( getHand(card(Rank.Ace, Suit.Club ),
        card(Rank.King, Suit.Club),
        card(Rank.Queen, Suit.Club),
        card(Rank.Jack, Suit.Club),
        card(Rank.Ten, Suit.Club)), HandClassifier.RoyalFlush);

    check( getHand(card(Rank.Nine, Suit.Heart ),
        card(Rank.Eight, Suit.Heart ),
        card(Rank.Seven, Suit.Heart ),
        card(Rank.Six, Suit.Heart ),
        card(Rank.Five, Suit.Heart )), HandClassifier.StraightFlush);

    check( getHand(card(Rank.Eight, Suit.Heart ),
        card(Rank.Eight, Suit.Diamond ),
        card(Rank.Eight, Suit.Club),
        card(Rank.Eight, Suit.Spade),
        card(Rank.Two, Suit.Club )), HandClassifier.FourOfAKind);

    check( getHand(card(Rank.Jack, Suit.Heart ),
        card(Rank.Jack, Suit.Diamond ),
        card(Rank.Jack, Suit.Spade ),
        card(Rank.King, Suit.Club ),
        card(Rank.King, Suit.Heart )), HandClassifier.FullHouse);

    check( getHand(card(Rank.Ace, Suit.Heart ),
        card(Rank.Jack, Suit.Heart ),
        card(Rank.Nine, Suit.Heart ),
        card(Rank.Eight, Suit.Heart ),
        card(Rank.Two, Suit.Heart )), HandClassifier.Flush);

    check( getHand(card(Rank.Nine, Suit.Heart ),
        card(Rank.Eight, Suit.Diamond ),
        card(Rank.Seven, Suit.Club ),
        card(Rank.Six, Suit.Heart ),
        card(Rank.Five, Suit.Spade )), HandClassifier.Straight);

    check( getHand(card(Rank.Nine, Suit.Heart ),
        card(Rank.Nine, Suit.Diamond ),
        card(Rank.Nine, Suit.Club ),
        card(Rank.Six, Suit.Heart ),
        card(Rank.Ace, Suit.Heart )), HandClassifier.ThreeOfAKind);

    check( getHand(card(Rank.Nine, Suit.Heart ),
        card(Rank.Nine, Suit.Diamond ),
        card(Rank.Six, Suit.Club),
        card(Rank.Six, Suit.Heart),
        card(Rank.Three, Suit.Spade )), HandClassifier.TwoPair);

    check( getHand(card(Rank.Nine, Suit.Heart ),
        card(Rank.Nine, Suit.Diamond ),
        card(Rank.Three, Suit.Club),
        card(Rank.Six, Suit.Heart),
```

```
        card(Rank.Ace, Suit.Diamond)), HandClassifier.Pair);

    check( getHand(card(Rank.Nine, Suit.Heart ),
        card(Rank.Two, Suit.Diamond ),
        card(Rank.Three, Suit.Club),
        card(Rank.Six, Suit.Heart),
        card(Rank.Queen, Suit.Club )), HandClassifier.HighCard);
}

Card card(Rank rank, Suit suit) {
    Card c;
    c.rank = rank;
    c.suit = suit;
    return c;
}

void check(Hand a, HandClassifier b) {
    if(a.classification != b) {
        println("" + b + " is not equal to: ");
        printHand(a);
    } else {
        println("Ok!");
        printHand(a);
    }
}
```