```java
// EXAM #2 Sample Solution Card
// Instructions
// ******************************************************************
/* Complete the definition of the following MUTABLE class named Card.
    Each instance of the class (i.e. each object) will represent a poker card with a
    suit and a rank.

Your job consists of completing the following tasks:

Complete the definition of the cardInCommon method
Complete the definition of the removeJokers instance method
Complete the definition of findHighCard instance method
Complete the definitions of the isInDeck instance method
Complete the definitions of the cardValue instance method
*/

// Sample Solution
// ******************************************************************

/**
 * Card Class
 */
public class Card {

    /**
     * Enum types for the card variables
     * - Suit
     * - Rank
     */
    enum Suit { CLUBS, DIAMONDS, HEARTS, SPADES }
    enum Rank { JOKER, A, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHTH,
        NINE, TEN, J, Q, K }

    Suit suit;
    Rank rank;

    /**
     * Main card class
     * @param s
     * @param r
     */
    public Card(Suit s, Rank r) {
        suit = s;
        rank = r;
    }

    /**
     * Checks if two cards are the same.
     * @param arg0
     */
    @Override
    public boolean equals(Object arg0) {
        if(!(arg0 instanceof Card)) { return false; }
        Card c = (Card) arg0;
        return suit == c.suit && rank == c.rank;
    }

    /**
     * Compares two cards.
     * Returns == to 0 if both cards are equal.
     * Returns > to 0 if the target object is bigger than the given card.
     * Returns < to 0 if the target object is smaller than the given card.
     */
    public int compareTo(Card o) {
        if (rank.compareTo(o.rank) == 0)
            return this.suit.compareTo(suit);
        else
            return this.rank.compareTo(o.rank);

    }

    /**
     * Exercise 1
     * Checks true if there are any cards in common between two decks.
     * @param deck1
     * @param deck2
     * @return
     */
    public static boolean cardInCommon(Card[] deck1, Card[] deck2) {
        for(int i = 0; i < deck1.length; i++) {
            for(int j = 0; j < deck2.length; j++) {
                if(deck1[i].equals(deck2[j])) {
                    return true;
                }
            }
        }
        return false;
    }

    /**
     * Exercise 2
     * Returns a new deck with all the Joker cards removed
     * from the original deck given how many joker cards are
     * in the given array.
     * @param deck
     * @param jokerCount
     * @return
     */
    public static Card[] removeJokers(Card[] deck, int jokerCount) {
        Card[] newDeck = new Card[deck.length-jokerCount];
        int index = 0;
        for(Card c: deck) {
            if(c.rank != Rank.JOKER) {
                newDeck[index++] = c;
            }
        }
        return newDeck;
    }

    /**
     * Exercise 3
     * Returns the High Card (Card with the Highest Value)
     * if available. (Ignore Poker Rules)
     * Hint: Use the method CompareTo() Method.
     * @param deck
     * @return
     */
    public static Card findHighCard(Card[] deck) {
        if(deck.length == 0) return null;
        Card highest = deck[0];
        for(int i = 1; i < deck.length; i++) {
            if(highest.compareTo(deck[i]) < 0) {
                highest = deck[i];
            }
        }
        return highest;
    }

    /**
     * Exercise 4
     * Returns true only if the target object is present inside
     * the given array.
```

```java
         * @param deck
         * @return
         */
        public boolean isInDeck(Card[] deck) {
            for(Card c: deck) {
                if(this.equals(c)) {
                    return true;
                }
            }
            return false;
        }

        /**
         * Exercise 5
         * Returns the value of a card based on the rank.
         *
         * Cards with rank Two to Ten have the same value as its name.
         * Rank Joker is zero.
         * Rank J, Q, K is 13, 14 and 15 respectively.
         * Rank A is 21.
         *
         * Hint: The enum types that is called .ordinal() which returns the
         * value assigned based on its position in the enum type list.
         * @return
         */
        public int cardValue() {
            if(this.rank == Rank.A) return 21;
            if(this.rank.ordinal() >= 11) return this.rank.ordinal()+2;
            return this.rank.ordinal();
        }
    }

    // Tests
    // ******************************************************************

    import static org.junit.Assert.*;

    import org.junit.Before;
    import org.junit.Test;

    public class CardTester {

        Card jokerDiamondCard;
        Card jokerClubsCard;
        Card AceHeartsCard;
        Card twoDiamondCard;
        Card twoDiamondCard2;
        Card fiveSpadesCard;
        Card nineHeartsCard;
        Card tenClubsCard;
        Card jHeartsCard;
        Card qHeartsCard;
        Card kSpadesCard;

        @Before
        public void setUp() {
            jokerDiamondCard = new Card(Card.Suit.DIAMONDS, Card.Rank.JOKER);
            jokerClubsCard = new Card(Card.Suit.CLUBS, Card.Rank.JOKER);
            AceHeartsCard = new Card(Card.Suit.HEARTS, Card.Rank.A);
            twoDiamondCard = new Card(Card.Suit.DIAMONDS, Card.Rank.TWO);
            twoDiamondCard2 = new Card(Card.Suit.DIAMONDS, Card.Rank.TWO);
            fiveSpadesCard = new Card(Card.Suit.SPADES, Card.Rank.FIVE);
            nineHeartsCard = new Card(Card.Suit.HEARTS, Card.Rank.NINE);
            tenClubsCard = new Card(Card.Suit.CLUBS, Card.Rank.TEN);
            jHeartsCard = new Card(Card.Suit.HEARTS, Card.Rank.J);
            qHeartsCard = new Card(Card.Suit.HEARTS, Card.Rank.Q);
            kSpadesCard = new Card(Card.Suit.SPADES, Card.Rank.K);
        }

        @Test
        public void cardInCommonTest() {
            // Decks
            Card[] emptyDeck = new Card[0];
            Card[] JokerD2D2D9H = { jokerDiamondCard, twoDiamondCard , twoDiamondCard2, 
            nineHeartsCard };
            Card[] QH2D10C = { qHeartsCard, twoDiamondCard, tenClubsCard };
            Card[] KSJockerC5S = {kSpadesCard, jokerClubsCard, fiveSpadesCard };

            // Empty Arrays
            assertFalse("The first array is empty.", Card.cardInCommon(emptyDeck, 
            JokerD2D2D9H));
            assertFalse("The second array is empty.", Card.cardInCommon(QH2D10C, emptyDeck));

            // Cards In Common
            assertTrue("The same array have cards in common.", Card.cardInCommon(QH2D10C, 
            QH2D10C));
            assertTrue("They have the 2 of Diamonds in Common.", Card.cardInCommon(
            JokerD2D2D9H, QH2D10C));

            // No Cards In Common
            assertFalse("They have no cards in common.", Card.cardInCommon(JokerD2D2D9H, 
            KSJockerC5S));
            assertFalse("They have no cards in common.", Card.cardInCommon(KSJockerC5S, 
            JokerD2D2D9H));

        }

        @Test
        public void removeJokerTest() {
            Card[] emptyDeck = new Card[0];
            Card[] KSJockerC5S = { kSpadesCard, jokerClubsCard, fiveSpadesCard };
            Card[] TenCJockerC9HJokerD = { tenClubsCard, jokerClubsCard, nineHeartsCard, 
            jokerDiamondCard };
            Card[] JockerDeck = {jokerClubsCard, jokerDiamondCard };
            Card[] JokerD2DJokerCJokerD = { jokerDiamondCard, twoDiamondCard, jokerClubsCard
            , jokerDiamondCard };

            // Empty Deck
            Card[] emptyDeckResult = Card.removeJokers(emptyDeck, 0);
            assertEquals("Result should be empty", 0, emptyDeckResult.length);

            // One Joker
            Card[] KSJockerC5SResult = Card.removeJokers(KSJockerC5S, 1);
            assertEquals("Result should have two items", 2, KSJockerC5SResult.length);
            assertEquals("Item 1 should be King of Spades", kSpadesCard, KSJockerC5SResult[0
            ]);
            assertEquals("Item 2 should be Five of Spades", fiveSpadesCard, 
            KSJockerC5SResult[1]);

            // Two Jokers
            Card[] TenJockerC9HJokerDResult = Card.removeJokers(TenCJockerC9HJokerD, 2);
            assertEquals("Result should have two items", 2, TenJockerC9HJokerDResult.length);
            assertEquals("Item 1 should be Ten of Clubs", tenClubsCard, 
            TenJockerC9HJokerDResult[0]);
            assertEquals("Item 2 should be Nine of Hearts", nineHeartsCard, 
            TenJockerC9HJokerDResult[1]);

            // Only Jokers
            Card[] JockerDeckResult = Card.removeJokers(JockerDeck, 2);
            assertEquals("Result should be empty", 0, JockerDeckResult.length);

            // Three Jokers
            Card[] JokerD2DJokerCJokerDResult = Card.removeJokers(JokerD2DJokerCJokerD, 3);
            assertEquals("Result should have one items", 1, JokerD2DJokerCJokerDResult.
            length);
            assertEquals("Item 1 should be Two of Diamonds", twoDiamondCard,
            JokerD2DJokerCJokerDResult[0]);
        }

        @Test
        public void findHighCardTest() {
            Card[] emptyDeck = new Card[0];
            Card[] oneCardDeck = { qHeartsCard };
            Card[] NineHKSJockerC5S = {nineHeartsCard, jokerClubsCard, kSpadesCard,
            fiveSpadesCard };
            Card[] JokerD2DJokerC2DJokerD = { jokerDiamondCard, twoDiamondCard,
            jokerClubsCard, twoDiamondCard2, jokerDiamondCard };

            assertTrue("There are no cards in the deck.", Card.findHighCard(emptyDeck)==null
            );
            assertEquals("It should return the only card.", qHeartsCard, Card.findHighCard(
            oneCardDeck));
            assertEquals("It should return the highest card.", kSpadesCard, Card.
            findHighCard(NineHKSJockerC5S));
            assertEquals("It should return the first high card.", twoDiamondCard, Card.
            findHighCard(JokerD2DJokerC2DJokerD));
        }

        @Test
        public void isInDeck() {
            Card[] emptyDeck = new Card[0];
            Card[] oneCardDeck = { qHeartsCard };
            Card[] NineHKSJockerC5S = {nineHeartsCard, jokerClubsCard, kSpadesCard,
            fiveSpadesCard };
            Card[] JokerD210CD2D9H = { jokerDiamondCard, twoDiamondCard, tenClubsCard,
            twoDiamondCard2, nineHeartsCard };

            assertFalse("The deck has no cards", qHeartsCard.isInDeck(emptyDeck));
            assertFalse("The deck has no Ten of Clubs", tenClubsCard.isInDeck(
            NineHKSJockerC5S));

            assertTrue("The deck has the Five of Spades", fiveSpadesCard.isInDeck(
            NineHKSJockerC5S));
            assertTrue("The deck has the Two of Diamonds", twoDiamondCard.isInDeck(
            JokerD210CD2D9H));

        }

        @Test
        public void cardValue() {

            assertEquals("Card is a Joker should return 0", jokerDiamondCard.cardValue(), 0);
            assertEquals("Card is an Ace should return 21", AceHeartsCard.cardValue(), 21);
            assertEquals("Card is an Two should return 2", twoDiamondCard.cardValue(), 2);
            assertEquals("Card is an Five should return 5", fiveSpadesCard.cardValue(), 5);
            assertEquals("Card is an Ten should return 10", tenClubsCard.cardValue(), 10);
            assertEquals("Card is an J should return 13", jHeartsCard.cardValue(), 13);
            assertEquals("Card is an Q should return 14", qHeartsCard.cardValue(), 14);
            assertEquals("Card is an k should return 15", kSpadesCard.cardValue(), 15);

        }

    }


    // *****************************************************************
```