```java
import java.util.List;
import java.util.ArrayList;

/**
 * The ElevensBoard class represents the board in a game of Elevens.
 */
public class ElevensBoard extends Board {

    /**
     * The size (number of cards) on the board.
     */
    private static final int BOARD_SIZE = 9;

    /**
     * The ranks of the cards for this game to be sent to the deck.
     */
    private static final String[] RANKS =
        {"ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "jack", "queen", "king"};

    /**
     * The suits of the cards for this game to be sent to the deck.
     */
    private static final String[] SUITS =
        {"spades", "hearts", "diamonds", "clubs"};

    /**
     * The values of the cards for this game to be sent to the deck.
     */
    private static final int[] POINT_VALUES =
        {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 0, 0};

    /**
     * Creates a new ElevensBoard instance.
     */
    public ElevensBoard() {
        super(BOARD_SIZE, RANKS, SUITS, POINT_VALUES);
    }

    /**
     * Determines if the selected cards form a valid group for removal.
     * In Elevens, the legal groups are (1) a pair of non-face cards
     * whose values add to 11, and (2) a group of three cards consisting of
     * a jack, a queen, and a king in some order.
     * @param selectedCards the list of the indices of the selected cards.
     * @return true if the selected cards form a valid group for removal;
     *         false otherwise.
     */
    public boolean isLegal(List<Integer> selectedCards) {
        /* *** TO BE IMPLEMENTED IN ACTIVITY 9 *** */
        if( selectedCards.size() == 2 )
            return containsPairSum11(selectedCards);
        else if( selectedCards.size() == 3 )
            return containsJQK(selectedCards);
        return false;
    }

    /**
     * Determine if there are any legal plays left on the board.
     * In Elevens, there is a legal play if the board contains
     * (1) a pair of non-face cards whose values add to 11, or (2) a group
     * of three cards consisting of a jack, a queen, and a king in some order.
     * @return true if there is a legal play left on the board;
     *         false otherwise.
     */
    public boolean anotherPlayIsPossible() {
        /* *** TO BE IMPLEMENTED IN ACTIVITY 9 *** */
        return containsPairSum11( cardIndexes() ) || containsJQK( cardIndexes() );
    }
    
    /**
     * Check for an 11-pair in the selected cards.
     * @param selectedCards selects a subset of this board.  It is list
     *                      of indexes into this board that are searched
     *                      to find an 11-pair.
     * @return true if the board entries in selectedCards
     *                 contain an 11-pair; false otherwise.
     */
    private boolean containsPairSum11(List<Integer> selectedCards) {
        for( int o = 0; o < selectedCards.size() - 1; o++ )
            for( int i = o + 1; i < selectedCards.size(); i++ )
                if( bCards[selectedCards.get(o)].getPointValue() + bCards[selectedCards.get(i)].getPointValue() == 11 )
                    return true;
        return false;
    }

    /**
     * Check for a JQK in the selected cards.
     * @param selectedCards selects a subset of this board.  It is list
     *                      of indexes into this board that are searched
     *                      to find a JQK group.
     * @return true if the board entries in selectedCards
     *                 include a jack, a queen, and a king; false otherwise.
     */
    private boolean containsJQK(List<Integer> selectedCards) {
        /* *** TO BE IMPLEMENTED IN ACTIVITY 7 (revised) *** */
        for( int o = 0; o < selectedCards.size() - 1; o++ )
            for( int m = 0; m < selectedCards.size() - 1; m++ )
                for( int i = 0; i < selectedCards.size() - 1; i++ ) {
                    if( bCards[selectedCards.get(o)].getPointValue() + bCards[selectedCards.get(m)].getPointValue() + bCards[selectedCards.get(i)].getPointValue() != 0 )
                        return false;
                    if( bCards[selectedCards.get(0)].equals( bCards[selectedCards.get(1)] ) || bCards[selectedCards.get(0)].equals( bCards[selectedCards.get(2)] ) || bCards[selectedCards.get(1)].equals( bCards[selectedCards.get(2)] ) )
                        return false;
                }
        return true;
    }
}
```