AP PROJECT **4.2.9**

Implement Elevens (AP)

INTRODUCTION

In AP Activity 4.2.8, you refactored (reorganized) the original `ElevensBoard` class into a new `Board` class and a much smaller `ElevensBoard` class. The purpose of this change was to allow you to reuse code in new games such as Tens and Thirteens. Now, you will complete the implementation of the methods in the refactored `ElevensBoard` class.

Materials

- Computer with BlueJIDE

RESOURCES

Free Response Question: Bird
Resources available online

Procedure

Part I

This project emphasizes public and private access to methods. Public methods usually change the state of the object, such as a board gains a card, it matches, and can be removed. Programmers will often write private *helper* methods in a class when the code gets large or complex. Private helper methods can *help* keep code clean and organized.

> **Reminder**: A private method cannot be used outside of the class.

1. Open and create a BlueJ project for *ElevensActivity9* with the provided java files.
2. Complete the `ElevensBoard` class by implementing the following methods, both `public` abstract methods and `private` helper methods.

Abstract methods in the `Board` class

a. `isLegal`— related comments below. The implementation should check the number of cards selected and utilize the ElevensBoard helper methods.

```
/**
 * 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.
 */
@Override
public boolean isLegal(List<Integer> selectedCards)
```

b. `anotherPlayIsPossible` — this method should also utilize the helper methods. It should be very short.

```
/**
 * 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.
 */
@Override
public boolean anotherPlayIsPossible()
```

Helper methods in the `ElevensBoard` class

c. `containsPairSum11`— this method determines if the selected elements of cards contain a pair of `cards` whose point values add to 11.

```
/**
 * Check for an 11-pair in the selected cards.
 * @param selectedCards selects a subset of this board. It is this
   list
 *                      of indexes into this board that are searched
 *                      to find an 11-pair.
 * @return true if the board entries indexed in selectedCards
 *              contain an 11-pair; false otherwise.
 */
private boolean containsPairSum11(List<Integer> selectedCards)
```

d. `containsJQK`— this method determines if the selected elements of `cards` contains a jack, a queen, and a king in some order.

```
/**
 * Check for a JQK in the selected cards.
 * @param selectedCards selects a subset of this board. It is this
   list
 *                      of indexes into this board that are searched
 *                      to find a JQK-triplet.
 * @return true if the board entries indexed in selectedCards
 *              include a jack, a queen, and a king; false otherwise.
 */
private boolean containsJQK(List<Integer> selectedCards)
```

3. When you have completed these methods, run the `main` method found in `ElevensGUIRunner.java`. Make sure that the Elevens game works correctly. Note that the `cards` directory must be in the same directory with your `.class` files.

Part II: Further Exploration

AP Focus: **FRQ: Bird**

CONCLUSION

1. The size of the board is one of the differences between Elevens and Thirteens. Why is size not an abstract method?
2. Why are there no abstract methods dealing with the selection of the cards to be removed or replaced in the array cards?
3. Another way to create "IS-A" relationships is by implementing interfaces. Suppose that instead of creating an `abstract Board` class, you created the following `Board` interface (shown below), and had `ElevensBoard` implement it. Would this new scheme allow the `Elevens` GUI to call `isLegal` and `anotherPlayIsPossible` polymorphically? Would this alternate design work as well as the `abstract Board` class design? Why or why not?

```
public interface Board
{
    boolean isLegal(List<Integer> selectedCards);

    boolean anotherPlayIsPossible();
}
```