Final Elevens Game: GUI #03

(designed to accompany the AP® Computer Science A Elevens Lab)

- To complete the game, there are many methods still needed. Follow the steps in this lab and you have a game!

1. Copy and paste all code from the Elevens_Lab12_GUI03_CODE (download) document into the GUI Class.

2. Notice that there will be many errors! This is because the GUI code for the game requires more methods. Examine all the errors and see if you can discover what methods still need to be written to make this code work!

3. The first method that appears to be missing is the anotherPlayIsPossible() method which determines if there are cards on the board that allow the user to continue playing. Add this method to the Board Class:

```
public boolean anotherPlayIsPossible() {
    ArrayList<Integer> cIndexes = cardIndexes();
    return containsPairSum11(cIndexes) || containsJQK(cIndexes);
}
```

4. After adding the anotherPlayIsPossible method, an error in the GUI class will be solved; however, there will now be two more lines of errors in the Board Class. This is because the anotherPlayIsPossible method calls 3 methods that have not been written yet. Look at the code, do you see the three methods that now need to be written?

5. Add the following three methods to the Board Class (can you understand the main idea behind what they do?):

```
// Method #1
public ArrayList<Integer> cardIndexes() {
   ArrayList<Integer> selected = new ArrayList<Integer>();
   for (int k = 0; k < cardsOnBoard.length; k++) {
      if (cardsOnBoard[k] != null) {
         selected.add(new Integer(k));
      }
   }
   return selected;
}

// Method #2
private boolean containsPairSum11(ArrayList<Integer> selectedCards) {
        for (int sk1 = 0; sk1 < selectedCards.size(); sk1++) {
      int k1 = selectedCards.get(sk1).intValue();
      for (int sk2 = sk1 + 1; sk2 < selectedCards.size(); sk2++) {
                int k2 = selectedCards.get(sk2).intValue();
                if (cardAt(k1).pointValue() + cardAt(k2).pointValue() == 11) {
          return true;
                }
      }
        }
        return false;
}
```

```
//Method #3
private boolean containsJQK(ArrayList<Integer> selectedCards) {
        boolean foundJack = false;
        boolean foundQueen = false;
        boolean foundKing = false;
        for (Integer kObj : selectedCards) {
      int k = kObj.intValue();
      if (cardAt(k).rank().equals("jack")) {
                foundJack = true;
      }
      else if (cardAt(k).rank().equals("queen")) {
         foundQueen = true;
      }
      else if (cardAt(k).rank().equals("king")) {
                foundKing = true;
      }
        }
        return foundJack && foundQueen && foundKing;
}
```

6. Return to the GUI Class. Are some of the previous errors disappearing? The next error has to do with a method named isLegal( ). This method is missing in the Board Class. Do you see what it does? Add it to the Board Class.

```
public boolean isLegal(ArrayList<Integer> selectedCards) {
        if (selectedCards.size() == 2) {
            return containsPairSum11(selectedCards);
        }
         else if (selectedCards.size() == 3) {
            return containsJQK(selectedCards);
        }
        else {
            return false;
        }
}
```

7. More methods are still needed in the Board Class. One method is needed that will replace cards and while another is needed to deal a new card to a specific location on the game board. Study the following two new methods and add them to the Board Class.:

```
public void replaceSelectedCards(ArrayList<Integer> selectedCards) {
        for (Integer k : selectedCards) {
             deal(k.intValue());
        }
}

public void deal(int k) {
        cardsOnBoard[k] = gameDeck.deal();
}
```

8. Yet another method is needed that checks to see if the game board is empty … an important feature needed to tell if the player has own the game or not. Study and add the following method to the Board class:

```
public boolean isEmpty() {
    for (int k = 0; k < cardsOnBoard.length; k++) {
        if (cardsOnBoard[k] != null) {
            return false;
        }
    }
    return true;
}
```

9. One final method!!! When the user clicks the button to start a new game, a method is needed that generates a new game! Study and add the following method to the Board class:

```
public void newGame() {
        gameDeck = new Deck();
        gameDeck.shuffle();
        dealCards();
}
```

10. And now for the GRAND FINALE … the moment of truth!!! Run the following GUITester code and hope that this game is all good to go!

```
public class GUITester {
   public static void main(String[] args) {
      Board setBoard = new Board();
      GUI newGame = new GUI(setBoard);
      newGame.displayGame();
   }
}
```

11. Game On! It only took me 23 games to win on my first go-around … probably just bad luck (I hope)!