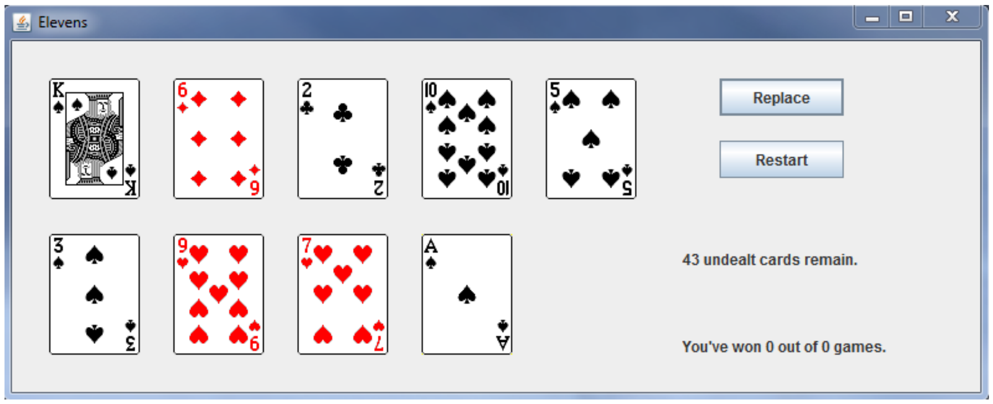
AP ACTIVITY **4.2.1**

Design and Create a Card Class (AP)

INTRODUCTION

The activities in this lesson are related to a simple solitaire game called Elevens. You will learn the rules of Elevens, which are similar to the rules of Sevens from the last lesson. You will be able to play it by using the supplied User Interface (UI) shown below. You will learn about the design and the object-oriented principles that suggested the design. You will also implement much of the code.



Materials

- Computer with BlueJ IDE

Procedure

In this activity, you will complete a `Card` class that will be used to create card objects. Think about the Sevens game from AP Lesson 4.1 and other card games you have played. What kinds of information do these games require a card object to "know"? What kinds of operations do these games require a `card` object to provide?

Now, think about implementing a class to represent a playing card, such as the ones pictured in the introduction. What instance variables should the class have? What methods should it provide?

1. Discuss your ideas for the `Card` class with your classmates.
2. Using source code that your teacher provides, open and create an *ElevensActivity1* project in BlueJ:
   - Create a directory called `ElevensActivity1` and move the *ElevensActivity1* java files into it.
   - In BlueJ, select **Project** > **Open Non BlueJ ...** In the navigation window, navigate to your `ElevensActivity1` directory.
   - The folder will look empty. Click **Open in BlueJ**. BlueJ will discover all java files and create a BlueJ project. Project information will be stored in a new `package.bluej` file in this folder.

Review the Card class and the Javadocs. Notice, similar to the Concentration project, the Object class's `toString` method is being overridden. Recall that every object in Java is a subclass of the `Object` class. And one of its most useful methods is the `toString()` method. If an overridden method such as `toString()` does not have the correct method signature, then the Java compiler will give an error message.

Let's look at some in method signatures of overridden methods. Programmers new to Java often encounter problems matching signatures of overridden methods to the superclass's original method signature. For example, in the `Weight` class below, the `toString` method is intended to be invoked when `toString` is called for a `Weight` object.

```
public class Weight {
    private int pounds;
    private int ounces;
    ...
    @Override
    public String tostring(String str) {
        return this.pounds + " lb. " + this.ounces + " oz.";
    }
    ...
}
```

Unfortunately, this doesn't work; the `tostring` method given above has a different name and a different signature from the `Object` class's `toString` method. The `@Override` annotation would cause an error message for this implementation to alert the programmer of the errors. The following is the correct version, which has the correct signature, specifically the correct name `toString`, the correct return type, and no parameter:

```
@Override
    public String toString() {
        return this.pounds + " lb. " + this.ounces + " oz.";
    }
```

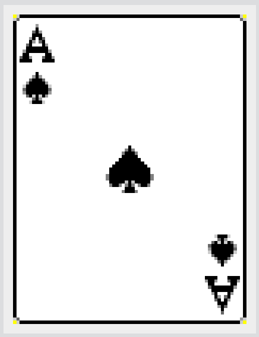
Another misconception related to the `toString` method is that it *prints out* a string, for example with `System.out`. This is not only incorrect use, but it also breaks encapsulation. Using a specific output method makes the object unusable in other UIs, such as an Android app, network apps in the cloud, and the UI for Elevens in this lesson. Remember `toString` *returns* a string value and *displays* nothing at all.

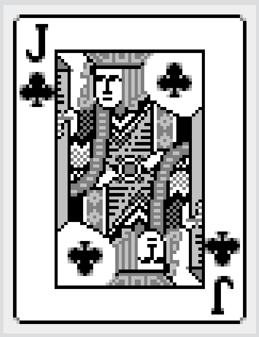
3 Complete the implementation of the provided `Card` class. You will be required to complete:

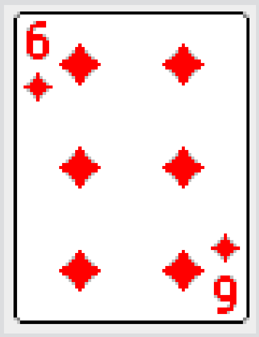
a. The constructor that takes two `String` parameters representing the card's rank and suit, and an `int` parameter representing the point value of the card.

Playing Cards: If you are unfamiliar with playing cards, this quick overview will help you with the Elevens project.

- **rank**: the number or letter on the card; 2, 3, 4, 5, 6, 7, 8 9, 10, Jack, Queen, King, or Ace
- **suit**: one of clubs, diamonds, hearts, or spades


Ace Of Spades


Jack of Clubs


Six of Diamonds

Cards often have a point value associated with them. For example, In the game of Twenty-One, card point values are added to reach a total of 21:

- Aces can be either 1 or 11, card holders choice
- Face cards, the Jack, Queen, and King are 10 points
- All other cards have their rank as their point value (a 6 has a point value of 6)

b. Accessor methods for the card's rank, suit, and point value.

c. The overridden `toString` method to return a `String` that contains the rank, suit, and point value of the card object. The string should be in the following format:

```
rank of suit (point value = pointValue)
```

4. Once you have completed the Card class, open the `CardTester.java`.
5. Create three `Card` objects and test each method for each `Card` object.

CONCLUSION

1. Think about a deck of cards.
   a. How would you describe a deck of cards?
   b. When you play card games, what kinds of operations do these games require a deck to provide?