Activity 1: Design and Create a `Card` Class

Introduction:

In this activity, you will complete a `Card` class that will be used to create card objects.

Think about card games you've 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?

Exploration:

Now think about implementing a class to represent a playing card. What instance variables should it have? What methods should it provide? Discuss your ideas for this `Card` class with classmates.

Read the partial implementation of the `Card` class available in the **Activity1 Starter Code** folder. As you read through this class, you will notice the use of the `@Override` annotation before the `toString` method. The Java `@Override` annotation can be used to indicate that a method is intended to override a method in a superclass. In this example, the `Object` class's `toString` method is being overridden in the `Card` class. If the indicated method doesn't override a method, then the Java compiler will give an error message.

Here's a situation where this facility comes in handy. Programmers new to Java often encounter problems matching headings of overridden methods to the superclass's original method heading. 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;
       ...
   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 correct version below has the correct name `toString` and no parameter:

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

The `@Override` annotation would cause an error message for the first `tostring` version to alert the programmer of the errors.

Exercises:

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

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

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

   c. a method to test equality between two card objects; and

   d. the `toString` method to create 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*`)`

2. Once you have completed the `Card` class, find the `CardTester.java` file in the **Activity1 Starter Code** folder. Create three `Card` objects and test each method for each `Card` object.