Introduction

Hi, and welcome to the final lesson in this series! We are going to apple what we've learned about classes to the Play Your Cards Right game.

Deck.rb

Create a file called *deck.rb* with a couple of classes: one to model cards and one to model the deck of cards.

```
class Card
  def initialize(name,suit)
    @name = name
    @suit = suit
  end

  def description
    "The #{ @name } of #{ @suit }"
  end

  def value
    case @name[0]
      when "J" then 11
      when "Q" then 12
      when "K" then 13
      else @name.to_i
    end
  end
end
```

The `Card` class has an `initialize` method that will be run every time we create a new card.

A card has two parameters: `name` and `suit`. We take those parameters andwe use them to set these two instance variables. They will be used to keep track of those facts about the cards throughout the class. Remember, instance variables can be accessed in any method inside of this class.

The methods that we give to the `Card` class are going to be `description` and `value`. The first one just returns a string to describe a card. The second one reuses some of the logic that we used earlier to get an actual numerical value of each card based on it's name.

Deck Class

Now let's take a look at the `Deck` class:

```
class Deck
  def initialize
    @cards = []
    suits = %w[ Hearts Diamonds Clubs Spades ]
    names = %w[ Ace 2 3 4 5 6 7 8 9 10 Jack Queen King ]
    suits.each do |suit|
      names.each do |name|
        @cards << Card.new(name,suit)
      end
    end
  end

  def shuffle
    @cards.shuffle!
  end

  def draw
    @cards.pop
  end
end
```

It models an actual deck of cards and uses the card objects from the `Card` class we've just created.

`initialize` does not take any parameters. It creates one array for suits and one for names and then stores cards inside the `@cards` array.

Shuffle Method

`shuffle` method just piggybacks on the shuffle method that the array uses, but we employ the bang method. So it makes sure once the cards are shuffled in the deck, they stay shuffled.

`draw` takes a card from the deck. Again, we piggyback on the array's `pop` method that will return the last item in an array.

So these methods, even though they basically just use array methods, just make them sound a bit more what we'd expect when we're using a deck of cards.

Refactoring Your Code

Now create a new *play_your_cards_rightv3.rb* file:

```
require 'sinatra'
require './deck'
enable :sessions

helpers do

  def set_up_game
    session[:deck] = Deck.new
    session[:deck].shuffle
    session[:guesses] = -1
  end

  def player_loses
    (params[:guess] == 'higher' and @card.value < session[:value]) or (params[:guess] == 'lower' a
  end

  def game_over
    "Game Over! The card was the #{ @card.description }. You managed to make #{session[:guesses]}
  end

  def update_session
    session[:value] = @card.value
    session[:guesses]  += 1
  end

  def ask_about_card
    "The card is the #{ @card.description }. Do you think the next card will be <a href='/play/hig
  end
end

get '/' do
  set_up_game
  redirect to('/play/cards')
end

get '/play/:guess' do
  @card = session[:deck].draw

  if player_loses
    game_over
  else
    update_session
    ask_about_card
```

```
  end
end
```

By extracting the logic into a separate classes, we drastically cut down on the amount of code that we use in the game's logic.

The first thing we do is requiring the file that we've just created. We don't need the *.rb* extension here.

We use `Deck.new` to create a new deck and then call `shuffle`.

Note that cards now have their own methods, so we take advantage of them.

The `/play/cards` route handler has some changes too. The biggest one is making the `card` an instance variable. This will mean we can use it throughout our Sinatra application, because actually, the Sinatra application runs off its own class.

Using Instance Variables as Views

So any instance variables that are used can then be referred to in all the different methods that we use in our Sinatra application. That's why instance variables can be used in views as well - it's a clever trick. They work in the same way as we were using them throughout class, because the Sinatra application is its own class.