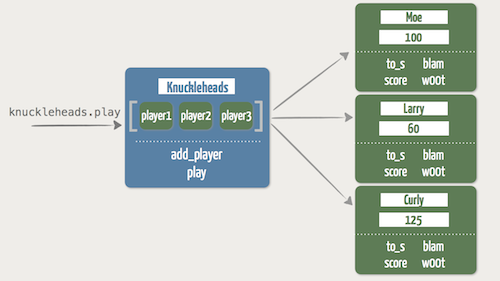
You currently have a Player class, and you've used it to create specific player objects. That's a good start, but we also need to model the concept of a game. A game is a collection of players and some behavior that lets us play the game. Here's visually what we want:

In this exercise, we'll implement this concept in a Game class. (Yes, we'll actually have something in our program called "game", finally!) In the end, we'll be able to play multiple games with a variety of players.

The objective is to be able to run this code:

knuckleheads = Game.new("Knuckleheads")  
knuckleheads.add\_player(player1)  
knuckleheads.add\_player(player2)  
knuckleheads.add\_player(player3)  
knuckleheads.play

and get this output:

There are 3 players in Knuckleheads:   
I'm Moe with a health of 100 and a score of 103.  
I'm Larry with a health of 60 and a score of 65.  
I'm Curly with a health of 125 and a score of 130.

So, take a moment to refill your tasty beverage, turn off the social media chatter, and we'll learn some more Ruby together.

1. Create a Game Class

1. Let's start by creating a new class called Game. Define it in your studio\_game.rb file below the Player class.

**class Game  
end**

1. Our game should always have a title, so write an initialize method that allows you to create a game object like so:
2. knuckleheads = Game.new("Knuckleheads")

**def initialize(title)  
 @title = title  
end**

1. For some practice creating attributes, make title a readable attribute. In other words, outside the class you should be able to do this:
2. puts knuckleheads.title

**attr\_reader :title**

### 2. Add Players

### Next we need a way to add player objects to the new game object. We'll do that in two steps. First, we'll keep an array of player objects in an instance variable in the Game class. Then we'll define an instance method on the Game class that appends the given player to the array.

1. In the initialize method of the Game class, define an instance variable called @players that starts off as an empty array.

**def initialize(title)  
 @title = title  
 @players = []  
end**

1. Next we need a method that adds player objects to the empty array we just created. Let's call that method add\_player. Sometimes it's easier to work out how to define a method if you have a specific example of how you'd call the method. Define the method so you can call it as follows:
2. knuckleheads.add\_player(player1)  
   knuckleheads.add\_player(player2)  
   knuckleheads.add\_player(player3)
3. Keep in mind that you may name your method parameter different than we do. That's ok—it's just a variable name.

**def add\_player(player\_one)   
 @players << player\_one  
end**

1. Finally, we need a way to actually *play* the game. When you play the game it should first print out each player's information. Then it should iterate through all the players and w00t or blam them (you decide how many w00ts or blams) and then print out their information. Here's an example of the output you're going for:

There are 3 players in Knuckleheads:   
I'm Moe with a health of 100 and a score of 103.  
I'm Larry with a health of 60 and a score of 65.  
I'm Curly with a health of 125 and a score of 130.  
Moe got blammed!  
Moe got w00ted!  
Moe got w00ted!  
I'm Moe with a health of 120 and a score of 123.  
Larry got blammed!  
Larry got w00ted!  
Larry got w00ted!  
I'm Larry with a health of 80 and a score of 85.  
Curly got blammed!  
Curly got w00ted!  
Curly got w00ted!  
I'm Curly with a health of 145 and a score of 150.

1. Implement this behavior in an instance method called play. If you're thinking you've done this somewhere before, you're right! You wrote similar code in the previous exercise. However, in that case the code lived in the top level of the program. The objective here is to tuck that code (with slight changes) in the play method so you can call it on any game object.

**def play  
 puts “There are #{@players.size} players in #{@title}:”  
 @players.each do |player|  
 puts player  
 end  
 @players.each do |player|  
 player.bam  
 player.w00t  
 player .w00t  
 puts player  
 end  
end**

1. Now that we have both a Game and a Player class, we can tidy up the code outside of the classes (at the bottom of the file). The only code we now need to run the game is the following:
2. player1 = Player.new("moe")  
   player2 = Player.new("larry", 60)  
   player3 = Player.new("curly", 125)  
      
   knuckleheads = Game.new("Knuckleheads")  
   knuckleheads.add\_player(player1)  
   knuckleheads.add\_player(player2)  
   knuckleheads.add\_player(player3)  
   knuckleheads.play
3. Check your work by running this code and making sure you get what you expect.
4. If for some reason you get an error, **don't panic!** Instead, calmly read the error message. Then read it again. Then try reading it out loud to yourself, or even to the person sitting nearest you in the coffee shop.
5. Seriously, error messages are your friend. They're Ruby's subtle way of trying to help you fix problems. Even the best programmers get error messages—often. The secret is that good programmers actually *read* them. And if you take the time to understand what Ruby's trying to tell you in an error message, after a while you'll be able to quickly recognize (and fix) common errors.

Bonus Round

Create a New Game

Remember, once you have a class you can use it to create many instances (objects) of it. For example, in addition to Knuckleheads, you could create a new game called Chipmunks with three different players: Alvin, Simon, and Theodore. You could even share players between games. (Though if you mix knuckleheads and chipmunks, it makes for a truly bizarre game!)

Create at least one more game object with a different set of players.

**player4 = Player.new (“Alvin”, 100)  
player5 = Player.new(“Simon”, 60)  
player6 = Player.new(“Theodore”, 125)**

Understanding how clusters of objects work together is the key to object-oriented programming. It's well worth practicing with a few additional examples. And if you're feeling really confident at this point, try opening a new program file and re-creating everything up to this point from scratch. Repeat until you don't have to look at any of the solutions. There's no substitute for practice!

Fundraising Program

In a similar way that we put movies in a Playlist class and you have players in a Game class, create a container class that holds on to your projects. Inside that class, here are some suggestions of things you could do:

* Give your collection of projects a name, such as "VC-Friendly Start-up Projects."
* Add your projects to an array so that you can deal with them all together.
* Add a method similar to the play method (you could call it request\_funding, for example) that iterates through the projects, prints out each project's information, adds or removes funds for each project, and then prints out the updated information.

class Project

attr\_accessor :name  
attr\_reader :funding, :target  
  
def initialize(name, target\_funding\_amount, funding=0)  
 @name = name  
 @target = target\_funding\_amount  
 @funding = funding  
end  
  
def to\_s  
 "#{@name} has $#{@funding} in funding towards a goal of $#{@target}."  
end

def remove\_funds  
 @funding -= 15  
 puts "#{@name} lost some funds!"  
end

def add\_funds  
 @funding += 25  
 puts "#{@name} got more funds!"  
end  
  
def total\_funding\_outstanding  
 @target - @funding  
end  
end

class FundRequest  
  
attr\_reader :title   
  
def initialize(title)  
 @title = title  
 @projects = []  
end

def add\_project(a\_project)  
 @projects.push(a\_project)  
end

def request\_funding  
 puts "There are #{@projects.size} projects that you could fund:"  
 @projects.each do |project|  
 puts project  
 end

@projects.each do |project|  
 project.add\_funds  
 project.remove\_funds  
 project.add\_funds  
 puts project  
 end  
end

end

project1 = Project.new("Project ABC", 5000, 1000)  
project2 = Project.new("Project LMN", 3000, 500)  
project3 = Project.new("Project XYZ", 75, 25)

projects = FundRequest.new("VC-Friendly Start-up Projects")

puts projects.title  
projects.add\_project(project1)  
projects.add\_project(project2)  
projects.add\_project(project3)

projects.request\_funding  
project4 = Project.new("Project TBD", 10000)  
projectrequest = FundRequest.new("Ask My Family For Money")

puts projectrequest.title  
  
projectrequest.add\_project(project1)  
projectrequest.add\_project(project2)  
projectrequest.add\_project(project3)  
projectrequest.add\_project(project4)

projectrequest.request\_funding