### Objective

In this exercise, we'll be visited by the Health Commission and Larry's mama! In order to keep them both happy, we'll want to be able to do three things from outside the Player class:

* Use a readable attribute to get access to a player's health
* Allow a player's name to be both read and written (changed)
* Access a player's score, which is computed based on name and health

In the end, we want to be able to run this code:

player2 = Player.new("larry", 60)  
puts player2.name  
player2.name = "Lawrence"  
puts player2.name  
puts player2.health  
puts player2.score  
puts player2

and get this output:

Larry  
Lawrence  
60  
68  
I'm Lawrence with a health of 60 and a score of 68.

### 1. Make a Readable Attribute

The Health Commission stopped by today and requested a list of our players' health values. (Apparently there's a public safety issue when players get blammed too many times.) Unfortunately, as our code now stands the only output we can give them is something like "I'm Moe with a health of 100." And associating his name with his health would violate Moe's privacy!

So in addition to being able to print out *all* of a player's information, we also want to get access to a player's health value from outside the Player class. To do that, we need to define an instance method that returns the value of the player's @healthinstance variable. In other words, we'll expose part of the object's internal state through a method. Outside of the class, we refer to the external state of an object as its *attributes*.

1. Add a read-only attribute to the Player class so that a player's health can be accessed from outside the class.

**class Player  
 Attr\_reader :health  
end**

1. Then, outside of the class, use the attribute to print out the (now anonymous) values for each player's health.

**puts player1.health  
puts player2.health  
puts player3.health**

1. The Health Commission thanks you for your outstanding public service!
2. Go ahead and make name a readable attribute as well. It's really no secret that player's have a name, and it's likely we'll need to access a player's name later on in the game.

**class Player  
 attr\_reader :name, :health  
end**

### 2. Make a Writable Attribute

### Not long afterwards, someone else shows up at your door. It's Larry's mama, and she doesn't like her son being called "Larry"—she prefers "Lawrence". So in addition to being able to read a player's name, we also need to write (change) a player's name from outside the class.

1. Change the Player class to expose the name attribute as both readable **and** writable.

**Class Player  
 attr\_accessor :name  
 attr\_reader :health  
end**

1. Now (outside of the class) change Larry's name to "Lawrence" using the name attribute and print out his correctly spelled name for Mama to confirm.

**player2.name = “Lawrence”  
puts player2.name**

### 

### 3. Make a Virtual Attribute

### Our Player class is shaping up nicely, but it's missing something important: a score. Every player needs a score, right? Of course! That's how we figure out who wins the game.

### Our first reaction may be to define a @score instance variable and make it a readable attribute. After all, instance variables hold on to an object's state. But in this case we simply want a player's score to be the sum of their health value and the number of characters in their name. So there's really no need for another instance variable.

### Instead, we'll write a *virtual attribute* to access the score from outside the Player class. The upshot of using an attribute here is that it keeps our code flexible. In an upcoming exercise, we'll change how a player's score is calculated to include points (rather than the length of their name). To do that, we'll change the implementation of the score method. But because the method *encapsulates* how the score is computed, changing it won't impact any callers of that method.

1. For now, write a virtual accessor (a method) called score that returns the player's health plus the number of characters in the player's name.

**class Player  
 def score  
 @health + @name.length  
 end  
end**

1. (Don't worry, we'll make the score a tad more sophisticated later.)
2. Then change the to\_s method to include the player's score, something like: "I'm Moe with a health of 100 and a score of 103."

**Def to\_s  
 “I’m #{@name} with a health of #{@health} and a score of #{score}**

1. Re-run the program to make sure the score is included in each player's information.

Bonus Round

name= Method

You may have noticed that changing a player's name using the name attribute doesn't automatically capitalize the name. For example, suppose you set a player's name using lowercase letters and print it out, like so:

player2.name = "lawrence"  
puts player2.name

The result will be "lawrence". As the Player class stands, the name is only capitalized in the initialize method. How could we automatically capitalize the name when writing to the name attribute?

Here's the secret: When you assign a value to the name attribute, for example, you're actually sending a message to the name=method. In other words, these two lines do exactly the same thing:

player2.name = "lawrence"  
  
player2.name=("lawrence")

The second line looks more like a method call because we used parentheses, but both lines call the name= method and pass in the value on the right-hand side of the equal sign. The first form is just a shortcut for the second form.

Ruby generated the default name= method for you when you used attr\_accessor :name, but you can easily override it. For example, if we want to capitalize the name before it's assigned, all we need to do is explicitly define a name= method in the Player class:

**def** **name**=(new\_name)  
 @name = new\_name.capitalize  
**end**

Now try running this code again:

player2.name = "lawrence"  
puts player2.name

This time it will call your version of the name= method and the name should get printed as "Lawrence". Pretty cool!

Fundraising Program

Now that you've learned about attributes, what state would you expose from outside your Project class? What virtual attribute would you like to have easily accessible?

Here are a few ideas:

* Allow access to a project's funding and target funding amount
* Allow a project's name to be changed (read and written both)
* Calculate the total funding still needed for a project (computed as the target funding amount minus current funding)

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

project1 = Project.new("Project ABC", 10000, 4000)  
puts project1  
puts project1.total\_funding\_outstanding

project2 = Project.new("Project LMN", 5000, 800)  
puts project2

project3 = Project.new("Project XYZ", 125, 25)  
puts project3

project4 = Project.new("Project TBD", 10000)  
puts project4

puts "\*\*\*"

puts project4.name

project4.name = "Project123"

puts project4.name

project1.remove\_funds

project2.remove\_funds

project3.add\_funds

project4.add\_funds

puts "\*\*\*"

puts project1

puts project1.total\_funding\_outstanding

puts project2

puts project3

puts project4