Object-oriented Programming

Juan “Harek” Urrios
@xharekx33

What is Object-oriented Programming?

Instead of the traditional view of a program as a logical procedure that takes input data, processes it, and produces output data, OOP is a programming paradigm based around the concept of **objects**.

The focus is in the objects that we want to manipulate rather than in the logic required to manipulate them

These objects are representations of entities (real or abstract) and they will interact with other objects to make up a computer program.

What are objects?

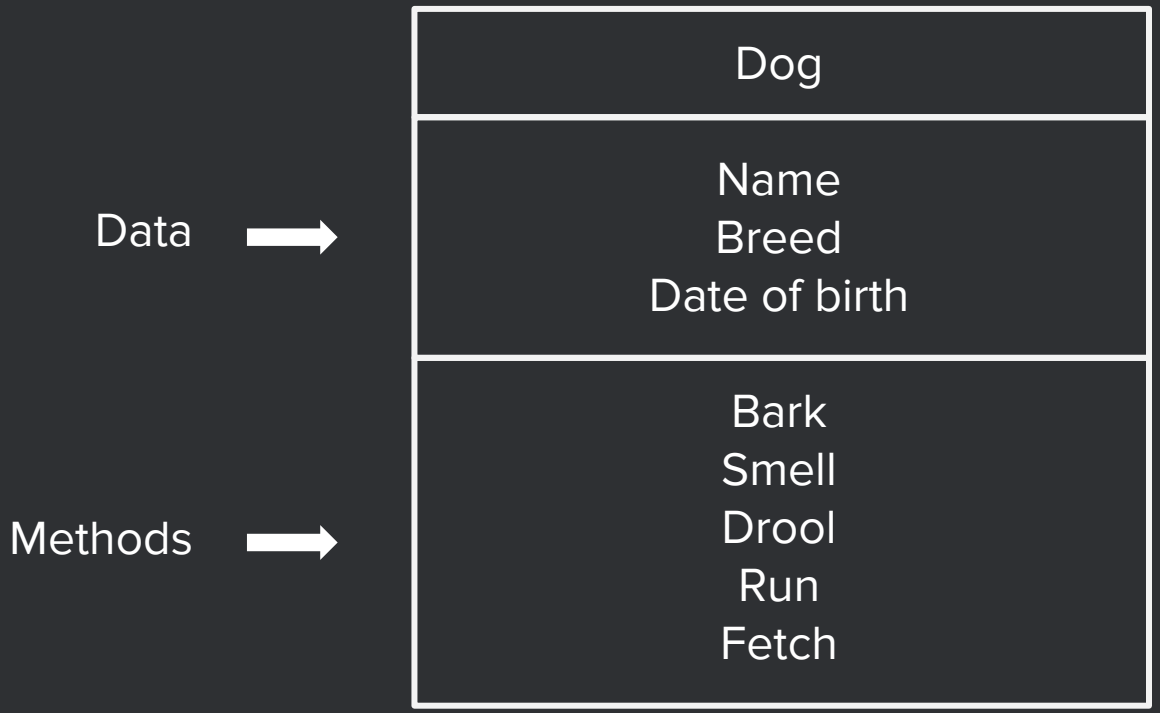
Objects are abstractions. Representations of entities (real or abstract).

They are often defined as a "data with methods".

The data in an object is stored in its **variables** and will describe its state.

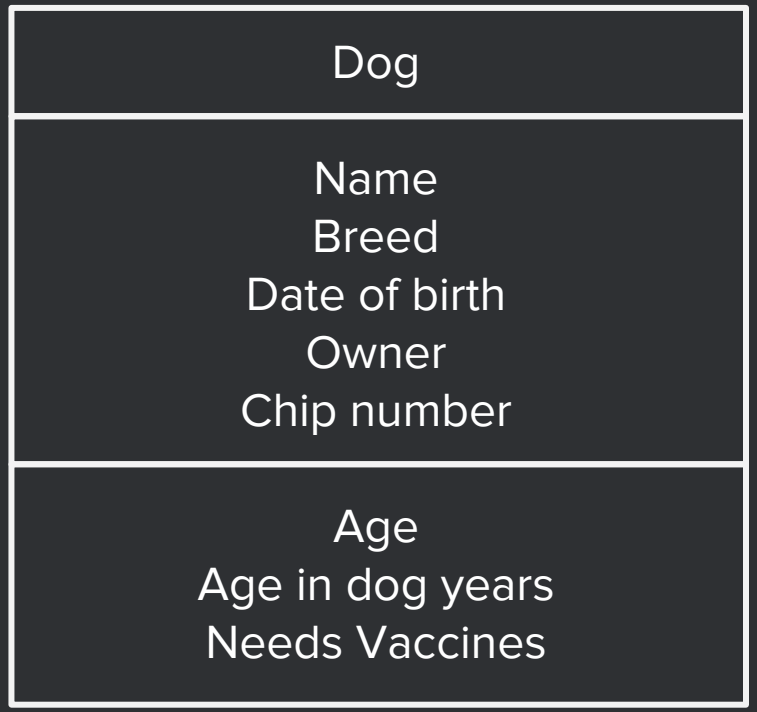
The methods will let us interact with the object and its data. They can be invoked by other objects or by itself.

Example: A dog as an object

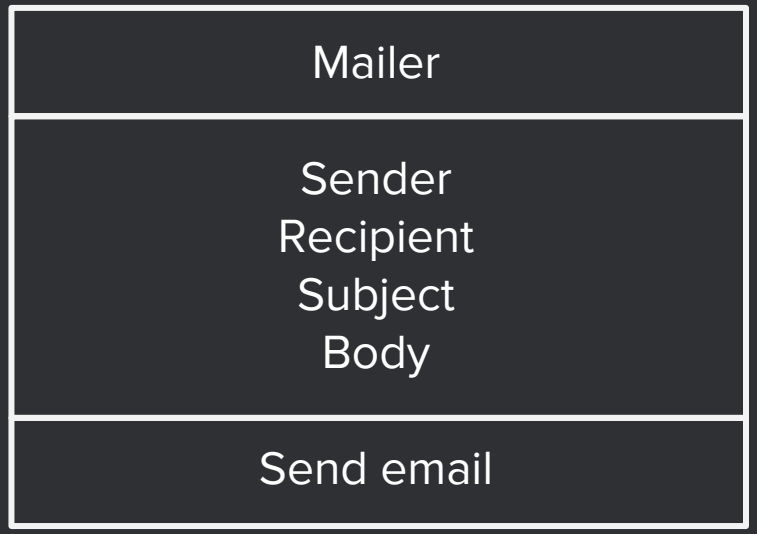


IRON HACK

Example: A more "useful" dog



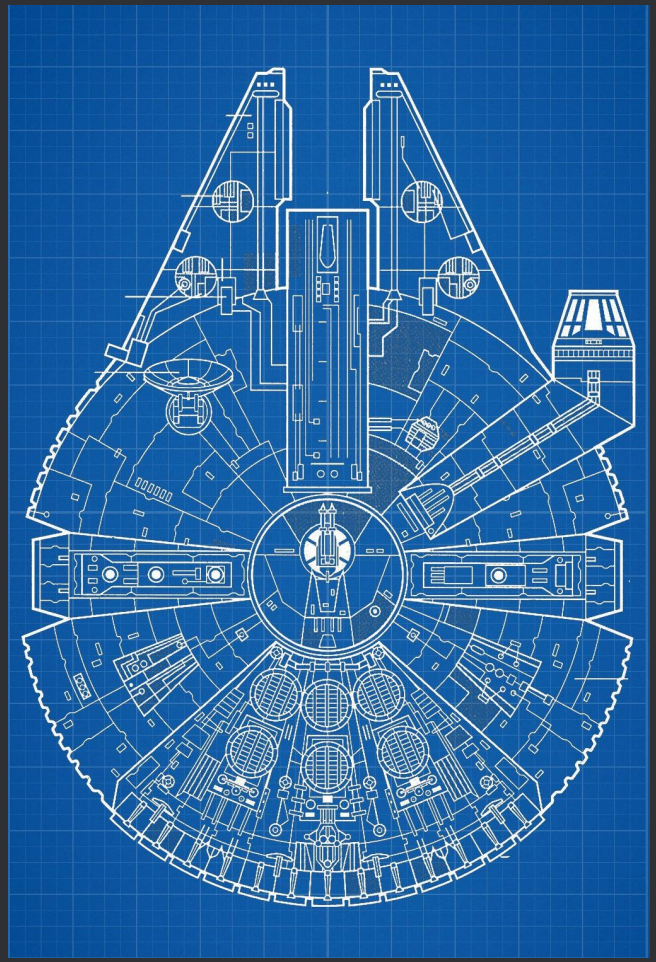
Example: A more abstract object



Classes and instances

Normally when we talk about an object we are talking about an **instance** of a class.

Classes contain the definition of the object, its data and its methods. They are like blueprints from which instances are created (instantiation).

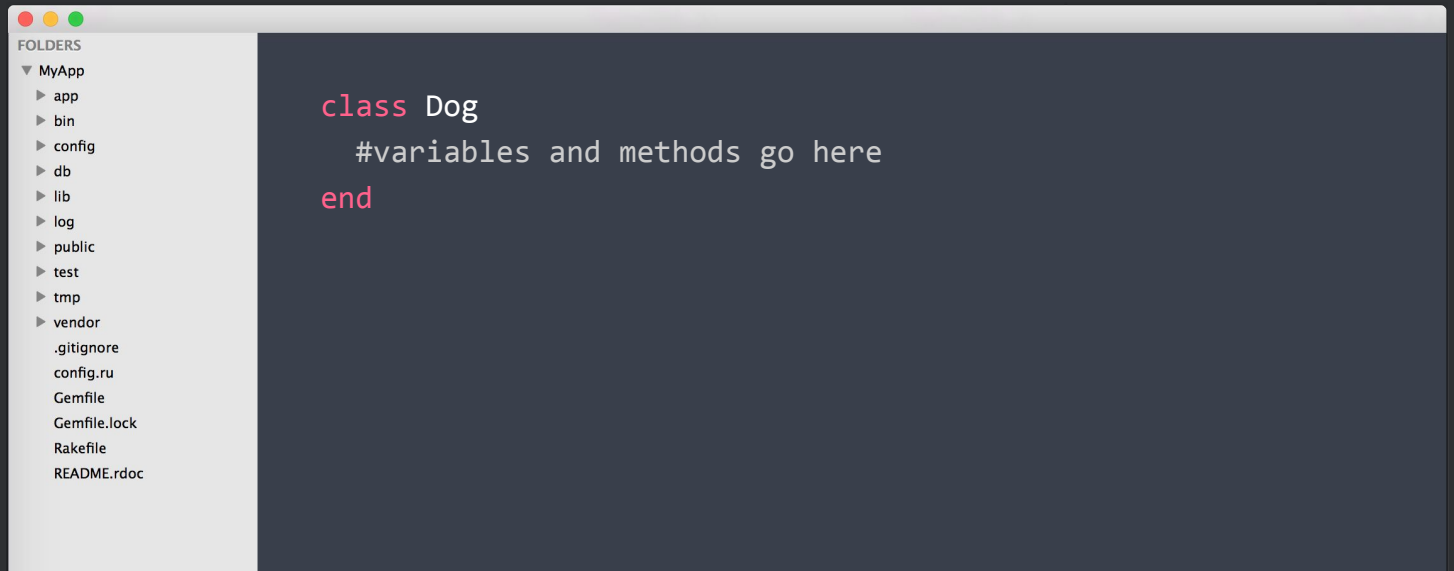


IRON HACK

@xharekx33

Classes in Ruby

Classes in Ruby are created using the class keyword, followed by the name of the class. In them we'll add our variables and methods.

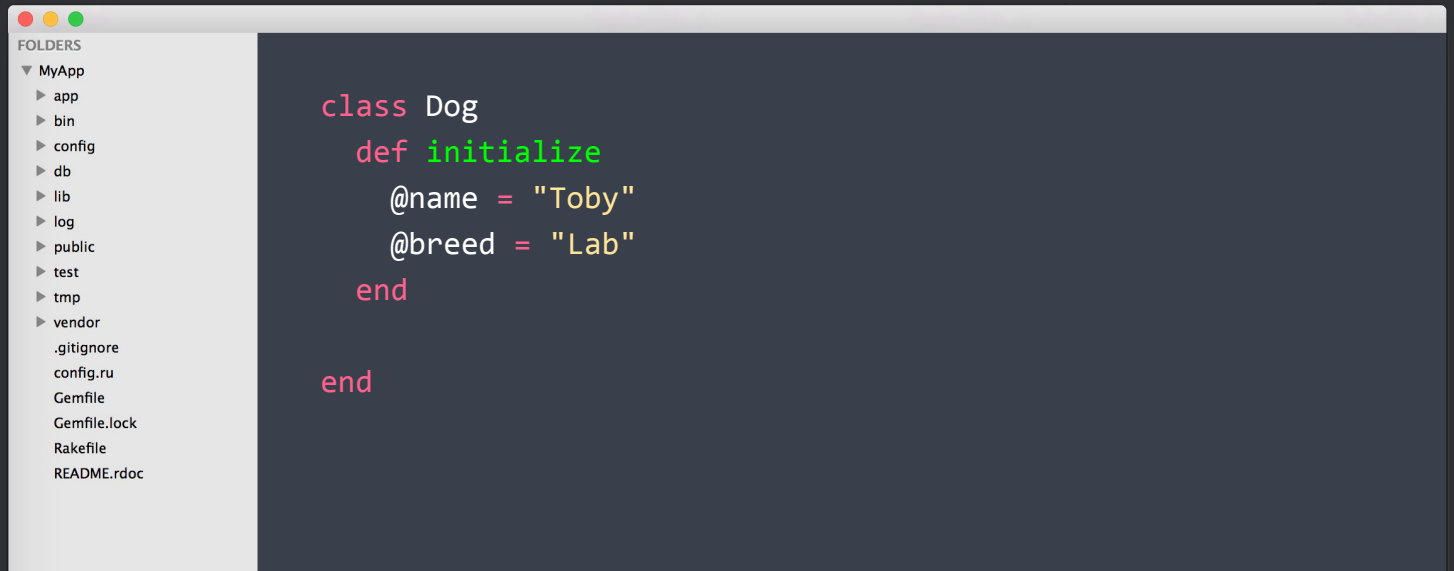


IRON HACK

@xharekx33

Constructor method

Whenever Ruby creates a new object, it looks for a method named `initialize` and executes it, with it we can initialize variables for our object with default values



IRON HACK

@xharekx33

Exercise

Add a method to let our dog bark by printing "Woof!" to the console. Instantiate the class and make your dog bark.

Variable Scope

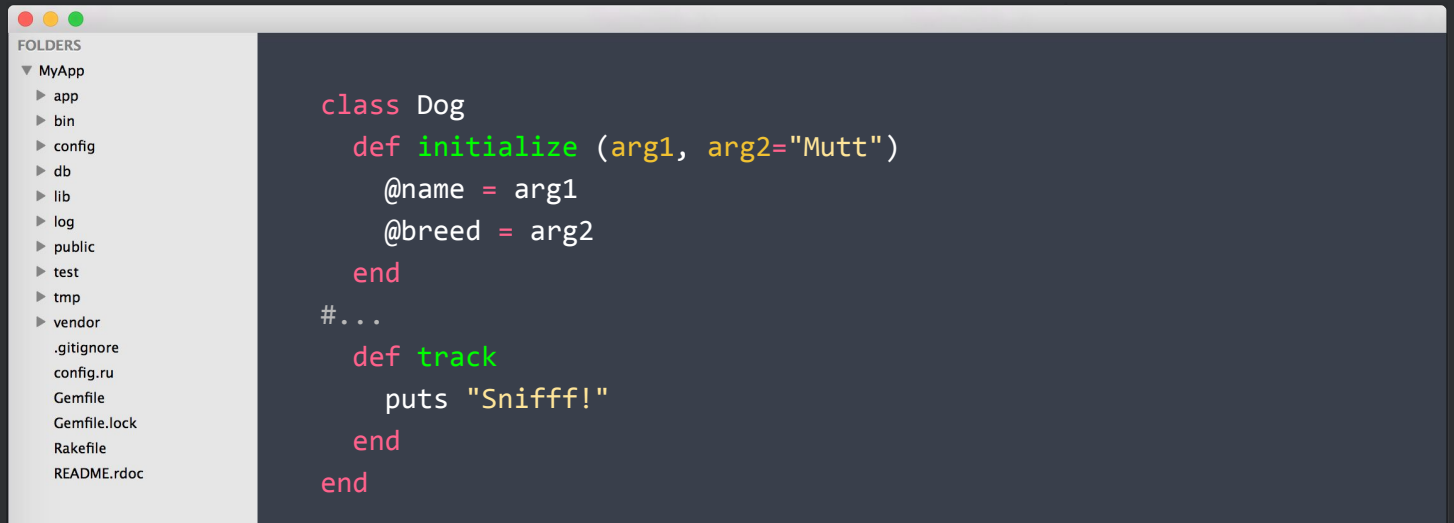
Variables inside our classes will have different scopes depending on where they will be accessible.

Ruby has four types of variable scope: **local**, **instance**, **class** and **global**

- **local:** Not available outside their method or construct. Name begins with [a-z] or _
- **instance:** Available across methods for an object. Name begins with @
- **class:** Available across all instances of a class. Name begins with @@
- **global:** Available anywhere in the Ruby program. Name begins with $

Constructor method: adding arguments

If we want to pass arguments on object creation we can add them to our initialize method and accept values for our variables. If we don't provide values for them we'll get an error.



IRON HACK

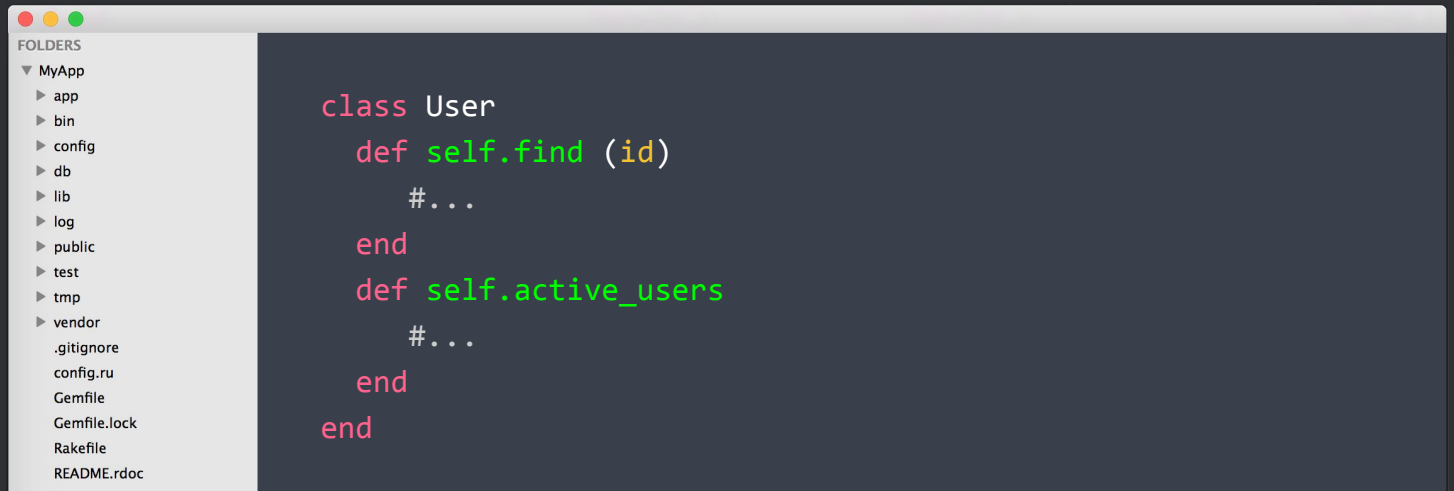
Exercise

Modify our Dog class and let different dogs have their own defining bark.

Create one that "Woof!"s and one that "Hooooooowl!"s

Class methods

By default all methods are **instance methods**, but we can create **class methods** that are not tied to any particular single instance. by prepending their name with `self.` The name of the class is used to call them: `Class.method`

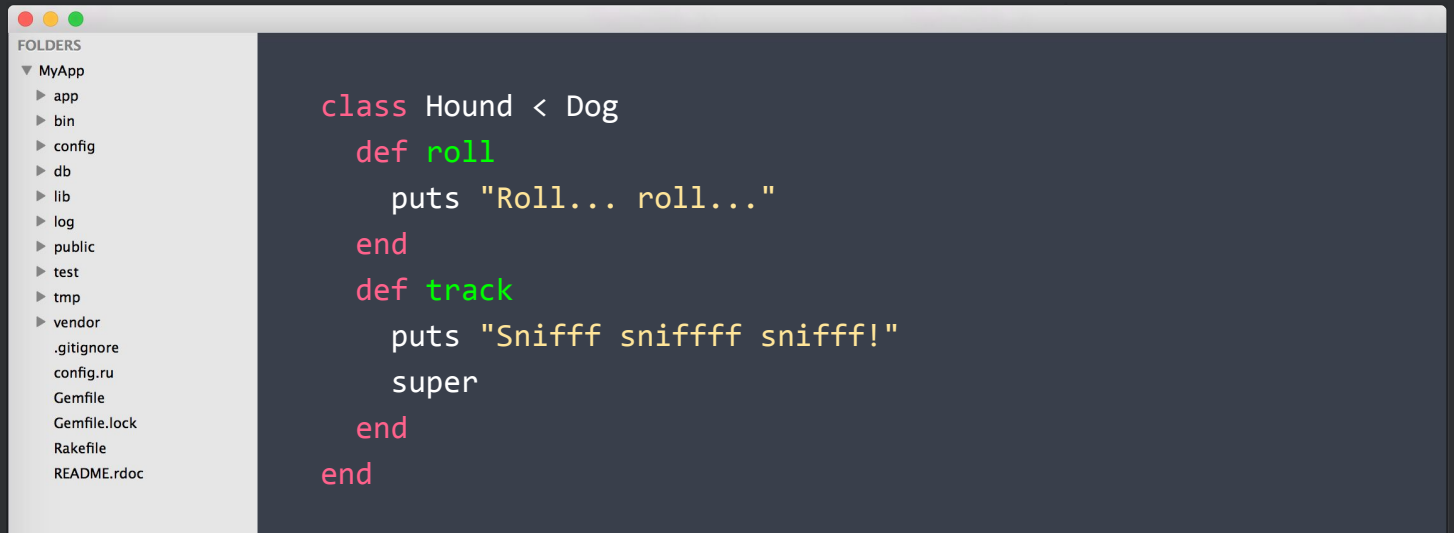


IRON HACK

@xharekx33

Inheritance

Classes can be based in other classes and share their implementations through inheritance. A child class inherits all the features of its parent class and can extend or replace them.



IRON HACK

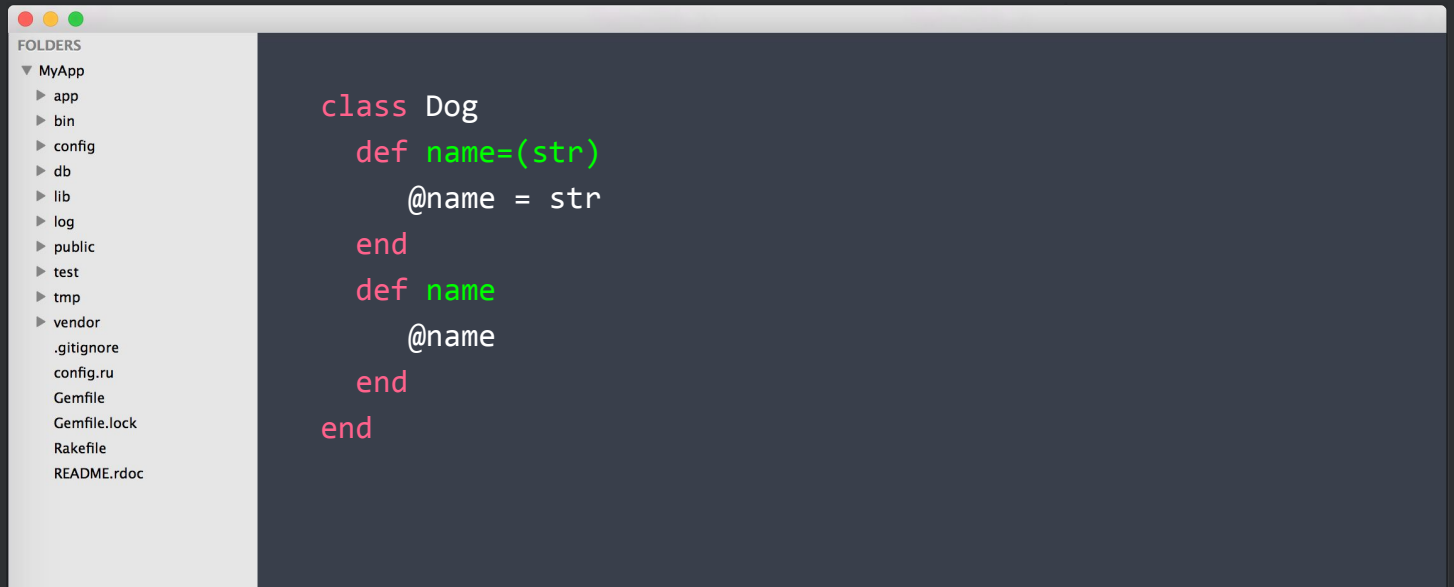
@xharekx33

Exercise

Extend our Dog class by creating a PetDog class that has an owner attribute and that can play fetch with different toys.

Attribute Accessors

To be able to write and read attributes in an object methods called **attribute accessors** are needed (a.k.a. "getters" and "setters")

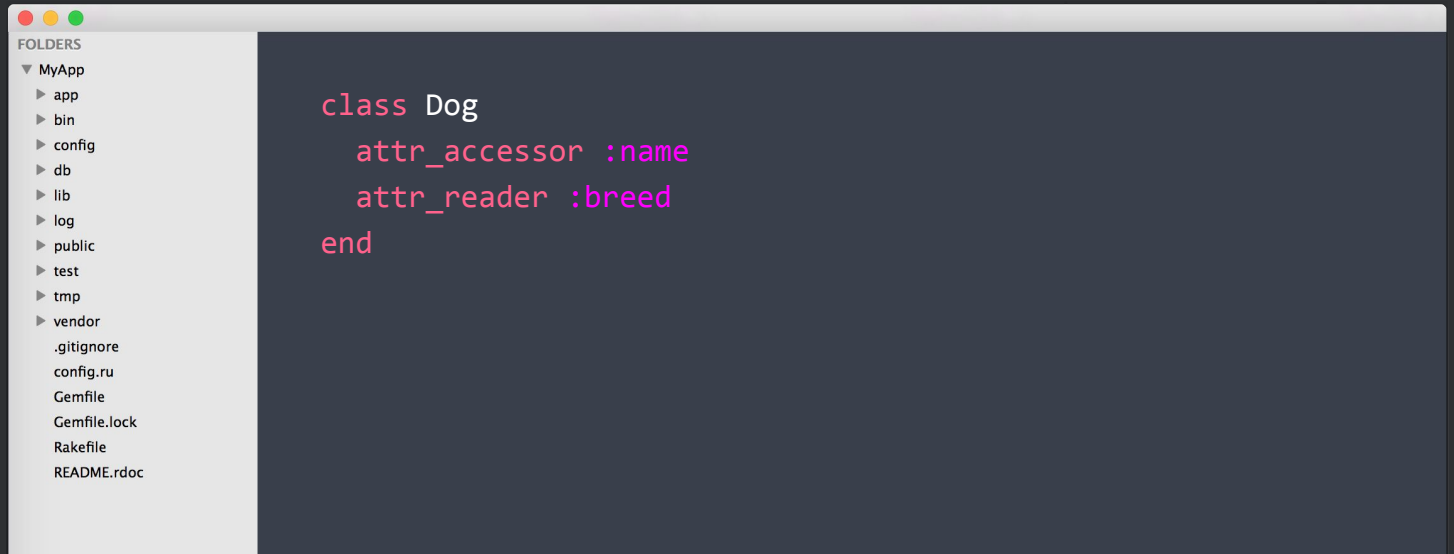


IRON HACK

@xharekx33

Attribute Accessors

Defining getters and setters for every attribute is a pain, so Ruby gives us some syntactic sugar for exposing them: The `attr_reader`, `attr_writer` and `attr_accessor` methods.



IRON HACK

@xharekx33

Encapsulation

Objects should only expose those parts (methods and variables) of themselves that are necessary for the "outside world" to use and manipulate, while keeping the rest safe and hidden.

They should behave as black boxes, hiding their internal details. What matters is WHAT they do, not HOW they do it.

Encapsulation reduces system complexity and increases robustness

Exercise

Our Dog class need to be able to "smell" someone and remember it.

It will also have a method to let us know all the people they've smelled by returning an array of names.

Exercise

Now change it so instead of saving the names of people they've smelled is saved to a file and not simply into an array variable.

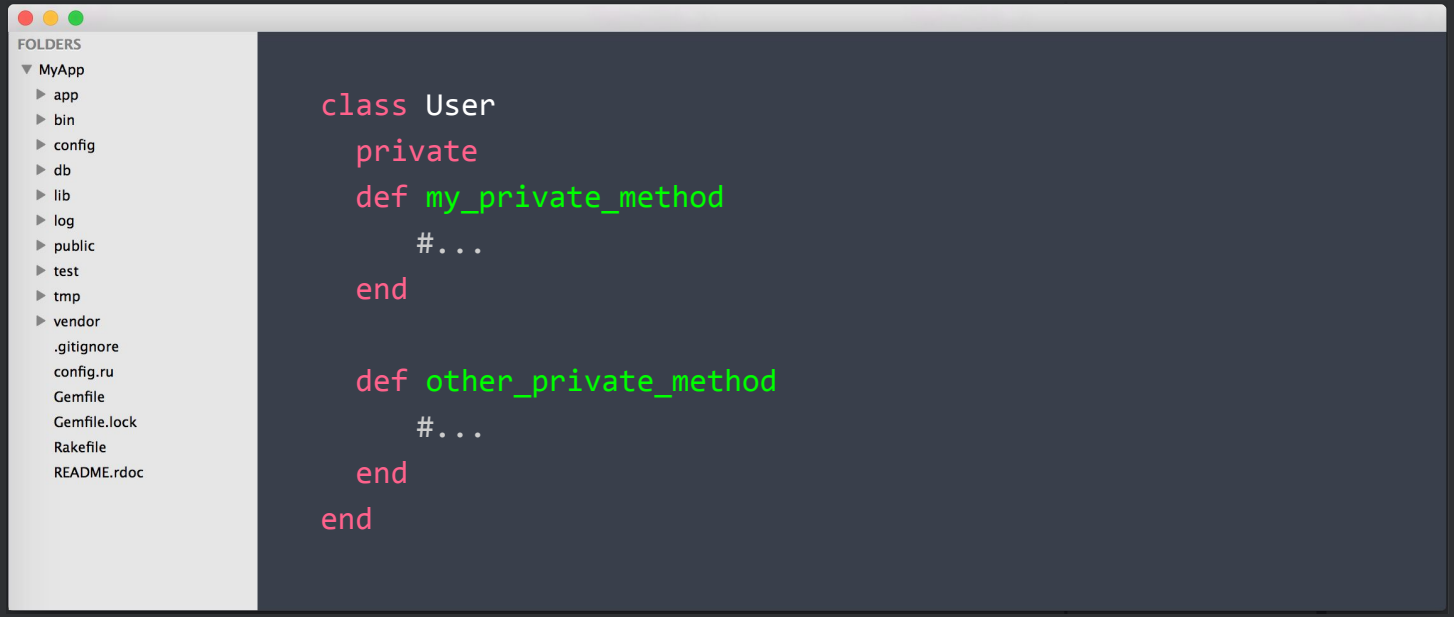
Method Scope

Methods can also have different scopes depending on where they will be accessible.

Ruby has three types of method scope: **public**, **private** and **protected**

- **public:** Available outside of the class. Interface of the object. Default for all methods.
- **private:** only accessible by the object's own instance methods
- **protected:** Available from instances of that class or its descendants

Method scope



IRON HACK

@xharekx33

Procedural programming VS OOP

Procedural programming uses a list of instructions to tell the computer what to do step-by-step, subdividing the code in procedures (subroutines) that operate on data.

In OOP data and the procedures that operate on them are bundled together into Objects. These objects are responsible for their data and no knowledge of its implementation is necessary for its use.

This helps with software reusability, maintainability and data integrity.