“No object is an island”

Wendy Kuhn Follow
Oct 6, 2017 · 5 min read

Image source: https://www.merriam-webster.com

Collaborator objects in OOP

Launch School’s 120: Object Oriented Programming curriculum states that *using collaborator objects is at the core of OO programming*. Considering this, I was surprised by how challenging it was to track down a concise definition that could tell me exactly *what is a collaborator object?*

If you Google *definition collaborator object oop*, the first result returned is a blog referencing Launch School’s curriculum, and the next top results all jump straight to detailed discussion of CRC cards, but lack a clear and consistent definition for collaborator objects. Interestingly, the creators of CRC cards stated that the tools were developed precisely because of the difficulty that learners have in grasping object oriented concepts. And they were right to develop this tangible learning aid because the definition given by these same authors for collaborator objects is about as clear as mud to a beginner like me: *We name as collaborators objects which will send or be sent messages in the course of satisfying responsibilities* (Beck & Cunningham).

Both Launch School and search engine results provided a number of examples of collaborator objects, but while specific examples are helpful, they don't by themselves provide an overarching framework, or mental model, in which to place the concept of collaborator objects.

So, I did some further digging and asking questions in Launch School's 120 Discussion forum in an attempt to gain clarity on the characteristics of collaborator objects. I've organized my findings into three underlying questions:

What is collaboration?

Collaboration is a way of modeling relationships between different objects.

There are a number of different types of relationships discussed with regard to OOP, and the number varies depending on which source you consult. I will just focus on two types of relationships for context in this discussion:

Inheritance can be thought of as an *is-a* relationship. For example, a dictionary is a book.
Association can be thought of as a *has-a* relationship. For example, a library has books, so there is an associative relationship between objects of class Library and objects of class Book.

Take away: A collaborative relationship is a relationship of association — not of inheritance. Collaboration is a *has-a* relationship rather than a *is-a* relationship.

What are collaborator objects and how can you spot them in the wild?

First, collaborator objects can be of any type: custom class object, Array, Hash, String, Integer, etc. The class of object really depends on the program that you are designing.

Second, a collaborator object is part of another object's state. For example, by assigning a collaborator object to an instance variable in another class' constructor method definition, you are associating the two objects with one another.

One way to spot them in code is when an object is assigned to an instance variable of another object inside a class definition. As you can see from Example 1 below, the `Person` object `joe` *has a* name — a `String` object with value “Bob” — as part of its state. So, object `"Bob"` assigned to the instance variable `@name` is a collaborator object of `joe`.

Example 1:

```
class Person
  attr_reader :name

  def initialize(name)
    @name = name
  end
end

joe = Person.new("Bob")
p joe              # => #<Person:0x00000001f86c80 @name="Bob">
joe.name           # => "Bob"
```

However, it’s not always the case that the instantiation of the collaborator object occurs in the definition itself. Sometimes, the class definition may just define a setter or other instance method, but the collaborator object does not become part of the primary object’s state until the setter or instance method is invoked elsewhere, outside of the class definition.

See another way to spot a collaborator in code in Example 2. In this example, you can see that the `Book` object `book_1` is not added to the state of `my_library` until the `Library#add_book` method is invoked on `my_library`. Note: You can also see from this example that `book_1` *has a* title; `@title` is a `String` object collaborator of objects of the `Book` class, which is made explicit in the `Book#initialize` method.

Example 2:

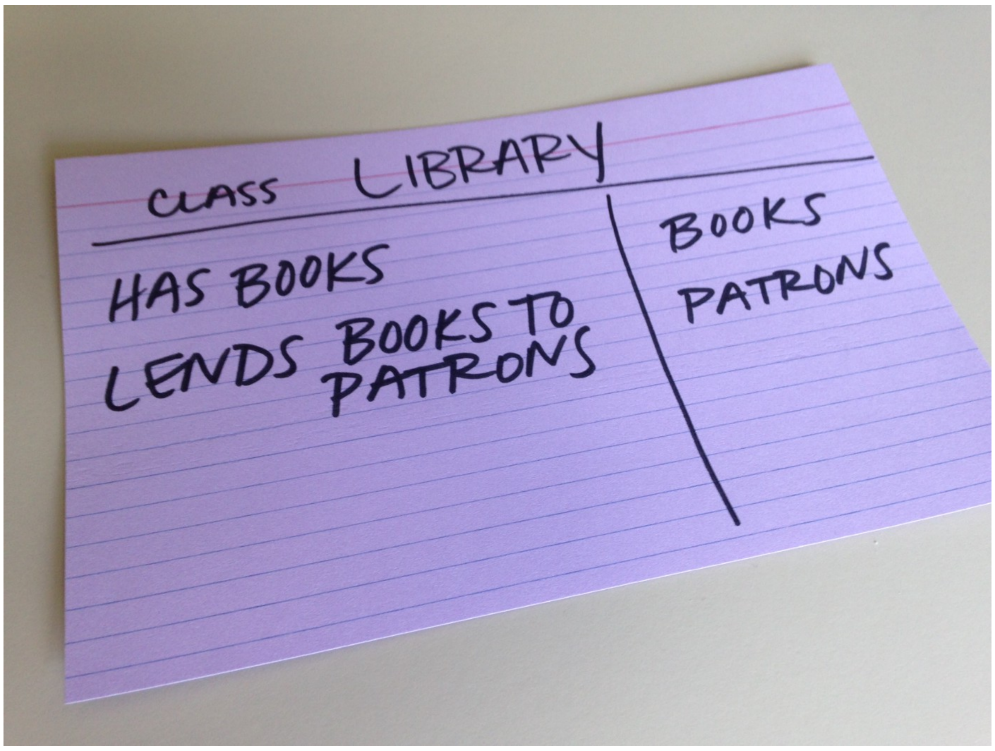
```
class Library
  def initialize
    @books = []
  end

  def add_book(book)
    @books << book
  end
end

class Book
  def initialize(title)
    @title = title
  end
```

```
end

my_library = Library.new
p my_library    # => #<Library:0x00000000c76050 @books=[]>

book_1 = Book.new('The Grapes of Wrath')
my_library.add_book(book_1)

p my_library    # => #<Library:0x00000001cedff0 @books=[#
<Book:0x00000001cede10 @title="The Grapes of Wrath">]>
```

Take away: A collaborator object is part of another object's state and can be an object of any class. The type of object depends on the context of your program.

When does collaboration occur?

So, as you can see from Example 2, it seems that collaboration with an actual object that occupies a place in memory doesn't always happen immediately upon instantiating your primary object. The `Book` object doesn't become part of the state of `my_library` until the `Library#add_book` method is invoked and adds `book_1` to the state of `my_library`. However, the collaborative relationship between `Library` and `Book` exists within the definition for the `Library` class. The CRC card for `Library` class might look something like this:



My first attempt at a CRC card

Take away: Collaboration doesn’t just occur when code is executed and objects are occupying space in memory, but it exists from the design phase of your program.

Final mental model (so far)

Top highlight

With regard to actual objects in memory, *collaboration* occurs when one object is added to the state of another object (i.e., when a method is invoked on an object). However, a more helpful mental model is: *the collaborative relationship exists in the design (or intention) of our code.*

Acknowledgement: I would like to thank Karl Lingiah for his contribution to my understanding of collaborator objects and for the Library-Books example code. I have *quoted* him a few times throughout this post.

Technical note: In reference to Example 2, the `@books` object is technically an `Array`. However, the array is just a way to organize and store the `Book` objects that will fill it, providing us with a number of `Array` methods with which to interact with our collection of books. It is the relationship between `Library` and `Books` that is meaningful in terms of the design of our program, not the relationship between `Library` and `Array`.

Programming Object Oriented Launch School How To Study Wwcode