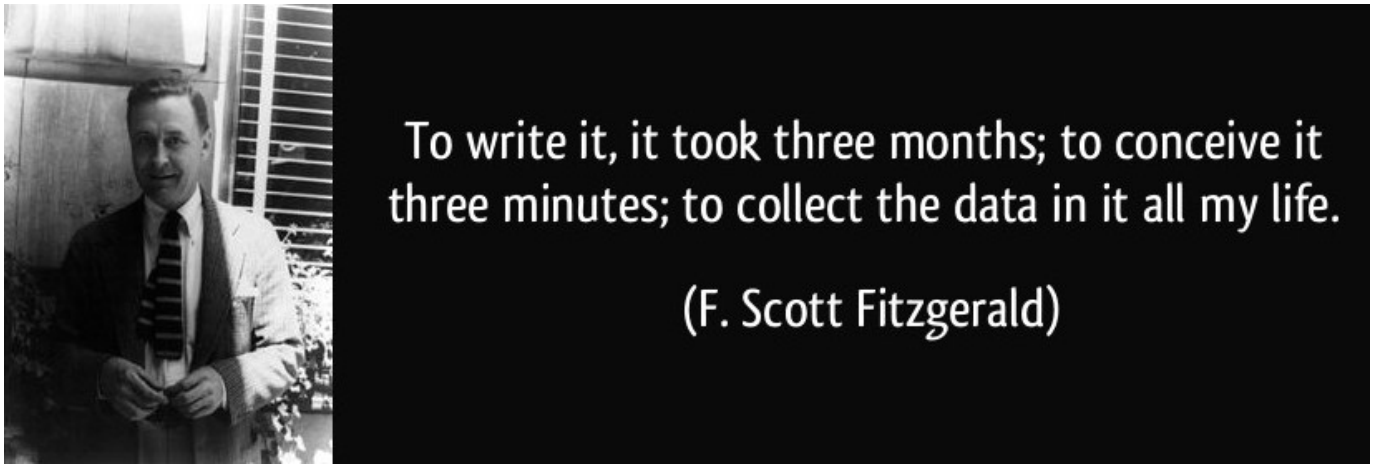
21 Address Bloc: Reading CSVs



Introduction

Populating Address Bloc with Data

Open the command line and navigate to the Address Bloc project:

~

```
$ cd ~/address-bloc
```

Git

Create a new Git feature branch for this checkpoint. See **Git Checkpoint Workflow: Before Each Checkpoint for details**.

Test

We'll use test-driven development to write the part of Address Bloc that will pull in data. First we'll write a test for `import_from_csv`.

Ultimately, we want the method to create entries in our `AddressBook` class. Let's assume that `AddressBook` will have five initial entries:

spec/address_book_spec.rb

```
  require_relative '../models/address_book'

  RSpec.describe AddressBook do
  # #1
+   let(:book) { AddressBook.new }

  # #2
  describe "attributes" do
    it "responds to entries" do
-     book = AddressBook.new
      expect(book).to respond_to(:entries)
    end

    it "initializes entries as an array" do
-     book = AddressBook.new
      expect(book.entries).to be_a(Array)
    end

    it "initializes entries as empty" do
-     book = AddressBook.new
      expect(book.entries.size).to eq 0
    end
  end

  describe "#add_entry" do
    it "adds only one entry to the address book" do
-     book = AddressBook.new
      book.add_entry('Ada Lovelace', '010.012.1815', 'augusta.king@lovelace.com')

      expect(book.entries.size).to eq 1
    end

    it "adds the correct information to entries" do
-     book = AddressBook.new
      book.add_entry('Ada Lovelace', '010.012.1815', 'augusta.king@lovelace.com')
      new_entry = book.entries[0]

      expect(new_entry.name).to eq 'Ada Lovelace'
      expect(new_entry.phone_number).to eq '010.012.1815'
      expect(new_entry.email).to eq 'augusta.king@lovelace.com'
    end
  end
```

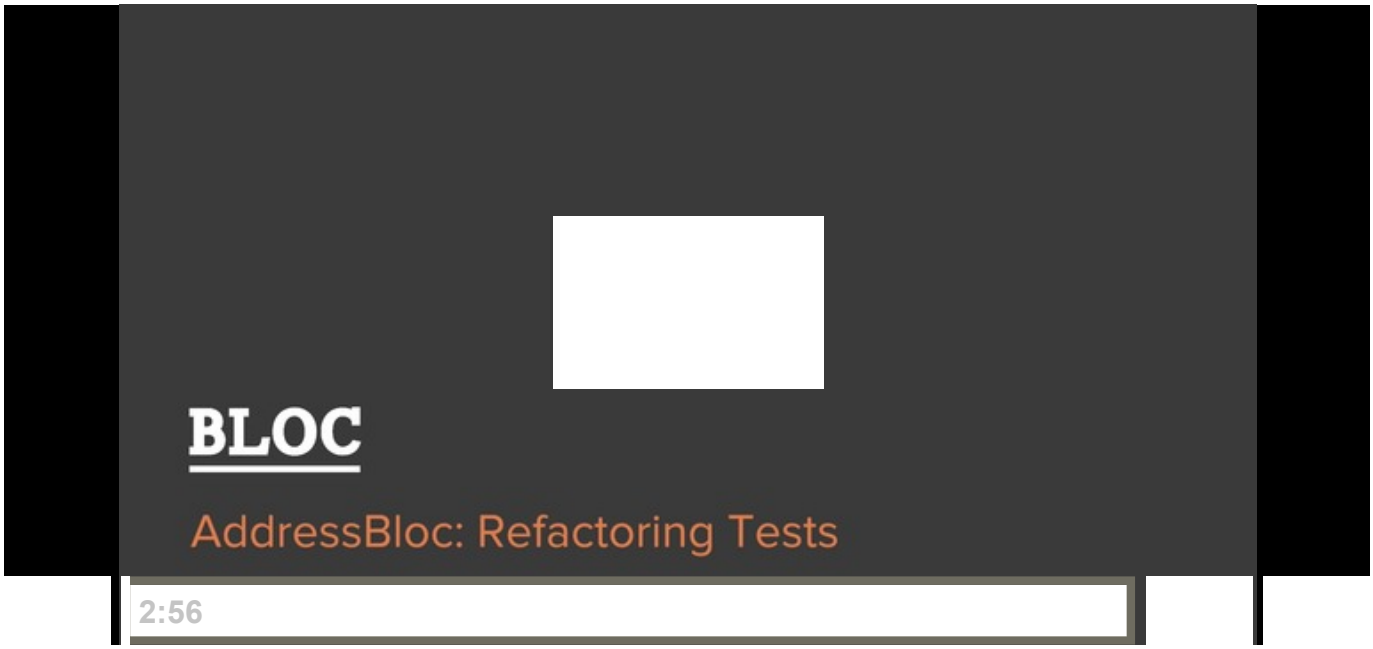
```
+   # Test that AddressBook's .import_from_csv() method is working as expected
+   describe "#import_from_csv" do
+     it "imports the correct number of entries" do
  # #3
+       book.import_from_csv("entries.csv")
+       book_size = book.entries.size
+
+       # Check the size of the entries in AddressBook
+       expect(book_size).to eq 5
+     end
+
+   end
  end
```

At **#1**, we create new instance of the AddressBook model and assign it to the variable named `book` using the `let` syntax provided by RSpec. This lets us use `book` in all our tests, removing the duplication of having to instantiate a new `AddressBook` for each test.

At **#2**, we see `describe` and `it` statements which are an RSpec paradigm to explain what we are testing. `it` explains the functionality of the method we're testing in a human readable form. RSpec will take the content from `describe` and `it` and output them nicely to the command line when we execute the test. **Read more about the differences between them**.

At **#3**, after the `describe` and `it` statements, we call the `import_from_csv` method on the `book` object which is of type `AddressBook` (our data model). We pass `import_from_csv` the string `entries.csv` as a parameter. `CSV` files are a fairly typical way of dealing with data and you can read more about them `here`. On the next line we reference the `AddressBook.entries` variable to get its size. This variable will be an **array**. Next, we save the size of the `AddressBook.entries` to our local variable `book_size`.

Watch the following video to see us walk through the refactoring steps above:



Run the spec. We should see it fail:

```
$ rspec spec/address_book_spec.rb
.....F

Failures:

  1) AddressBook#import_from_csv imports the correct number of entries
     Failure/Error: book.import_from_csv("entries.csv")

     NoMethodError:
       undefined method `import_from_csv' for #<AddressBook:0x007ff393940618 @entrie
     # ./spec/address_book_spec.rb:40:in `block (3 levels) in <top (required)>'

Finished in 0.00545 seconds (files took 0.16404 seconds to load)
6 examples, 1 failure

Failed examples:

rspec ./spec/address_book_spec.rb:39 # AddressBook#import_from_csv imports the corre
```

Let's stub `import_from_csv` to get rid of the `NoMethodError`.

Stub

Open `AddressBook` and add the following:

models/address_book.rb

```
  require_relative 'entry'
+ require "csv"

  class AddressBook
    attr_reader :entries

    def initialize
      @entries = []
    end

    def add_entry(name, phone, email)
      index = 0
      entries.each do |entry|
        if name < entry.name
          break
        end
        index += 1
      end
      entries.insert(index, Entry.new(name, phone, email))
    end

+   def import_from_csv(file_name)
+     # Implementation goes here
+   end
  end
```

Change the working directory back to the project root if you're not already in it:

Terminal

```
$ cd ~/address-bloc
```

Run the spec we just created:

Terminal

```
$ rspec spec/address_book_spec.rb
.....F

Failures:

  1) AddressBook#import_from_csv imports the correct number of entries
     Failure/Error: expect(book_size).to eq 5

       expected: 5
            got: 0

       (compared using ==)
     # ./spec/address_book_spec.rb:44:in `block (3 levels) in <top (required)>'

Finished in 0.02073 seconds (files took 0.11983 seconds to load)
6 examples, 1 failure

Failed examples:

rspec ./spec/address_book_spec.rb:39 # AddressBook#import_from_csv imports the corre
```

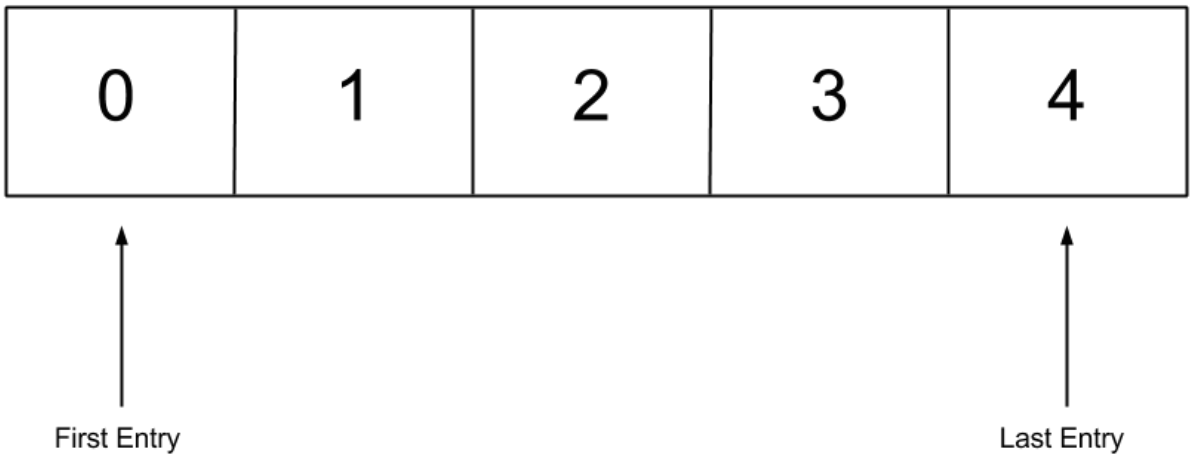
We see that the spec fails. This is logical since `import_from_csv` is stubbed out but has no implementation. Let's add another test:

spec/address_book_spec.rb

```
  require_relative '../models/address_book'

  RSpec.describe AddressBook do
    ...

    # Test that AddressBook's .import_from_csv() method is working as expected
    describe "#import_from_csv" do
      it "imports the correct number of entries" do
        book.import_from_csv("entries.csv")
        book_size = book.entries.size

        # Check the size of the AddressBook.entries
        expect(book_size).to eq 5
      end
  # #4
+     it "imports the 1st entry" do
+       book.import_from_csv("entries.csv")
+       # Check the first entry
+       entry_one = book.entries[0]

  # #5
+       expect(entry_one.name).to eq "Bill"
+       expect(entry_one.phone_number).to eq "555-555-4854"
+       expect(entry_one.email).to eq "bill@blocmail.com"
+     end

    end
  end
```

At **#4**, we access the first entry in the array of entries that our `AddressBook` stores.

> Remember, arrays in Ruby use zero based numbering. The first element is located at index zero, the second element at index one, and so on.



At **#5**, we've added three `expect`s to verify that the first entry has the name "Bill", the phone number "555-555-4854", and the email address "bill@blocmail.com". If we run this test, it will still fail. Using the same pattern above, we can add four similar tests:

spec/address_book_spec.rb

```
require_relative '../models/address_book'

RSpec.describe AddressBook do
  ...

  # Test that AddressBook's .import_from_csv() method is working as expected
  describe "#import_from_csv" do
    it "imports the correct number of entries" do
      book.import_from_csv("entries.csv")
      book_size = book.entries.size

      # Check the size of the AddressBook.entries
      expect(book_size).to eq 5
    end

    it "imports the 1st entry" do
      book.import_from_csv("entries.csv")
      # Check the first entry
      entry_one = book.entries[0]
      expect(entry_one.name).to eq "Bill"
      expect(entry_one.phone_number).to eq "555-555-4854"
      expect(entry_one.email).to eq "bill@blocmail.com"
    end
```

```
+     it "imports the 2nd entry" do
+       book.import_from_csv("entries.csv")
+       # Check the second entry
+       entry_two = book.entries[1]
+       expect(entry_two.name).to eq "Bob"
+       expect(entry_two.phone_number).to eq "555-555-5415"
+       expect(entry_two.email).to eq "bob@blocmail.com"
+     end
+
+     it "imports the 3rd entry" do
+       book.import_from_csv("entries.csv")
+       # Check the third entry
+       entry_three = book.entries[2]
+       expect(entry_three.name).to eq "Joe"
+       expect(entry_three.phone_number).to eq "555-555-3660"
+       expect(entry_three.email).to eq "joe@blocmail.com"
+     end
+
+     it "imports the 4th entry" do
+       book.import_from_csv("entries.csv")
+       # Check the fourth entry
+       entry_four = book.entries[3]
+       expect(entry_four.name).to eq  "Sally"
+       expect(entry_four.phone_number).to eq "555-555-4646"
+       expect(entry_four.email).to eq "sally@blocmail.com"
+     end
+
+     it "imports the 5th entry" do
+       book.import_from_csv("entries.csv")
+       # Check the fifth entry
+       entry_five = book.entries[4]
+       expect(entry_five.name).to eq "Sussie"
+       expect(entry_five.phone_number).to eq "555-555-2036"
+       expect(entry_five.email).to eq "sussie@blocmail.com"
+     end

   end
 end
```

We've added tests to test for four more entries. Our test now expects our data to have five total entries with varying names, numbers, and email addresses.

Our tests have a large amount of **redundancy** (code duplication). Let's use a helper method to check each entry and reduce the clutter:

spec/address_book_spec.rb

```
  require_relative '../models/address_book'

  RSpec.describe AddressBook do
    let(:book) { AddressBook.new }

  # #6
+  def check_entry(entry, expected_name, expected_number, expected_email)
+    expect(entry.name).to eq expected_name
+    expect(entry.phone_number).to eq expected_number
+    expect(entry.email).to eq expected_email
+  end

  ...

    describe "#import_from_csv" do
      it "tests the csv import process" do
        book.import_from_csv("entries.csv")
        book_size = book.entries.size

        # Check the size of the AddressBook.entries
        expect(book_size).to eq 5
      end

      it "imports the 1st entry" do
        book.import_from_csv("entries.csv")
        # Check the first entry
        entry_one = book.entries[0]
+       check_entry(entry_one, "Bill", "555-555-4854", "bill@blocmail.com")
-       expect(entry_one.name).to eq "Bill"
-       expect(entry_one.phone_number).to eq "555-555-4854"
-       expect(entry_one.email).to eq "bill@blocmail.com"
      end

      it "imports the 2nd entry" do
        book.import_from_csv("entries.csv")
        # Check the second entry
        entry_two = book.entries[1]
+       check_entry(entry_two, "Bob", "555-555-5415", "bob@blocmail.com")
-       expect(entry_two.name).to eq "Bob"
```

```
-       expect(entry_two.phone_number).to eq "555-555-5415"
-       expect(entry_two.email).to eq "bob@blocmail.com"
      end

      it "imports the 3rd entry" do
        book.import_from_csv("entries.csv")
        # Check the third entry
        entry_three = book.entries[2]
+       check_entry(entry_three, "Joe", "555-555-3660", "joe@blocmail.com")
-       expect(entry_three.name).to eq "Joe"
-       expect(entry_three.phone_number).to eq "555-555-3660"
-       expect(entry_three.email).to eq "joe@blocmail.com"
      end

      it "imports the 4th entry" do
        book.import_from_csv("entries.csv")
        # Check the fourth entry
        entry_four = book.entries[3]
+       check_entry(entry_four, "Sally", "555-555-4646", "sally@blocmail.com")
-       expect(entry_four.name).to eq  "Sally"
-       expect(entry_four.phone_number).to eq "555-555-4646"
-       expect(entry_four.email).to eq "sally@blocmail.com"
      end

      it "imports the 5th entry" do
        book.import_from_csv("entries.csv")
        # Check the fifth entry
        entry_five = book.entries[4]
+       check_entry(entry_five, "Sussie", "555-555-2036", "sussie@blocmail.com")
-       expect(entry_five.name).to eq "Sussie"
-       expect(entry_five.phone_number).to eq "555-555-2036"
-       expect(entry.email).to eq "sussie@blocmail.com"
      end

    end
  end
```

At **#6**, we create a helper method named `check_entry` which consolidates the redundant code. We can now pass in the particular name, number, and email address we want into this reusable helper method. We have our basic tests set up. The next step is to build the implementation of the `import_from_csv` method.

Implement

Let's add the code to `AddressBook`:

models/address_book.rb

```
  require_relative 'entry'
  require "csv"

  class AddressBook
    attr_reader :entries

    def initialize
      @entries = []
    end

    def add_entry(name, phone, email)
      index = 0
      entries.each do |entry|
        if name < entry.name
          break
        end
        index += 1
      end
      entries.insert(index, Entry.new(name, phone, email))
    end

  # #7
    def import_from_csv(file_name)
+     csv_text = File.read(file_name)
+     csv = CSV.parse(csv_text, headers: true, skip_blanks: true)
  # #8
+     csv.each do |row|
+       row_hash = row.to_hash
+       add_entry(row_hash["name"], row_hash["phone_number"], row_hash["email"])
+     end
+
-     # Implementation goes here
    end
  end
```

Let's break down the code above.

At **#7**, we defined `import_from_csv`. The method starts by reading the file, using `File.read`. The file will be in a `CSV` format. We use the `CSV` class to **parse** the file. The result of `CSV.parse` is an object of type `CSV::Table`.

At **#8**, we **iterate** over the `CSV::Table` object's rows. On the next line we create a **hash** for each row. We convert each `row_hash` to an `Entry` by using the `add_entry` method which will also add the Entry to the AddressBook's entries.

Create a Data Source for Address Bloc

As we have already alluded to, we'll use a `CSV` file in our Address Bloc Ruby app. We have a functional test and a functional `import_from_csv` method. If you run the test we just created, it will fail since `entries.csv` does not exist and the code attempts to use this as its `CSV` file. Let's create the `entries.csv` file.

Place the `entries.csv` file in the same directory as the `address_bloc.rb` file. Open a text editor and enter the following into the file:

```
name,phone_number,email
Bill,555-555-4854,bill@blocmail.com
Bob,555-555-5415,bob@blocmail.com
Joe,555-555-3660,joe@blocmail.com
Sally,555-555-4646,sally@blocmail.com
Sussie,555-555-2036,sussie@blocmail.com
```

Run the `address_book_spec.rb`:

Terminal

```
$ rspec spec/address_book_spec.rb
.

Finished in 0.00514 seconds (files took 0.10853 seconds to load)
11 examples, 0 failures
```

The test passes so we know that the implementation is reading `entries.csv` and storing its values properly.

Recap

Concept	Description
TDD	We stubbed out `import_from_csv` to act as a placeholder. Then we defined the expected behavior of `import_from_csv` using tests. We built the tests first so that they could constrain the structure of `import_from_csv`. This forced us to build the method in a way that fulfilled our desired outcome. We call this practice **Test Driven Development**. With our tests in place, we built the implementation of `import_from_csv` until the tests passed.
Data Source	Lastly, we connected the actual data to our application by creating the `CSV` file and ran the test using the real data.

Assignment | Discussion | Submission

Create a new Git feature branch for this assignment. See **Git Checkpoint Workflow: Before Each Assignment** for details.

We may want to parse another `CSV` file at some point.

- Add tests to `address_book_spec.rb` that will use data from a new `CSV` file named `entries_2.csv` (do not delete the existing tests in `address_book_spec.rb`).
- Add a `CSV` file named `entries_2.csv` with three entries (do not delete `entries.csv`).
- Ensure the new test passes with the new data.

Commit your assignment in Git. See **Git Checkpoint Workflow: After Each**

Assignment for details. Submit your commit to your mentor.

Solution

Do not watch this video until after you've attempted to complete the assignment. If you struggle to complete the assignment, submit your best effort to your mentor *before watching a solution video*.

Media Queries Solution

assignment completed

?