23 AddressBloc: Menu Continued

> “The closer one gets to the speed of light, the slower time travels. The exact thing is true when completing a book.”
>
> — **Daniel Ionson**

Introduction

We're almost done! We need to write the rest of the methods in `menu_controller.rb` to connect the user interface of Address Bloc to the methods in `AddressBook`.

Git

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

Import

Start by writing the body of the stubbed `read_csv` method:

controllers/menu_controller.rb

```
    def read_csv
  # #1
+     print "Enter CSV file to import: "
+     file_name = gets.chomp
+
  # #2
+     if file_name.empty?
+       system "clear"
+       puts "No CSV file read"
+       main_menu
+     end
+
  # #3
+     begin
+       entry_count = address_book.import_from_csv(file_name).count
+       system "clear"
+       puts "#{entry_count} new entries added from #{file_name}"
+     rescue
+       puts "#{file_name} is not a valid CSV file, please enter the name of a valid
+       read_csv
+     end
    end
```

At **#1**, we prompt the user to enter a name of a `CSV` file to import. We get the filename from `STDIN` and call the `chomp` method which removes `newlines`.

At **#2**, we check to see if the file name is empty. If it is then we return the user back to the main menu by calling `main_menu`.

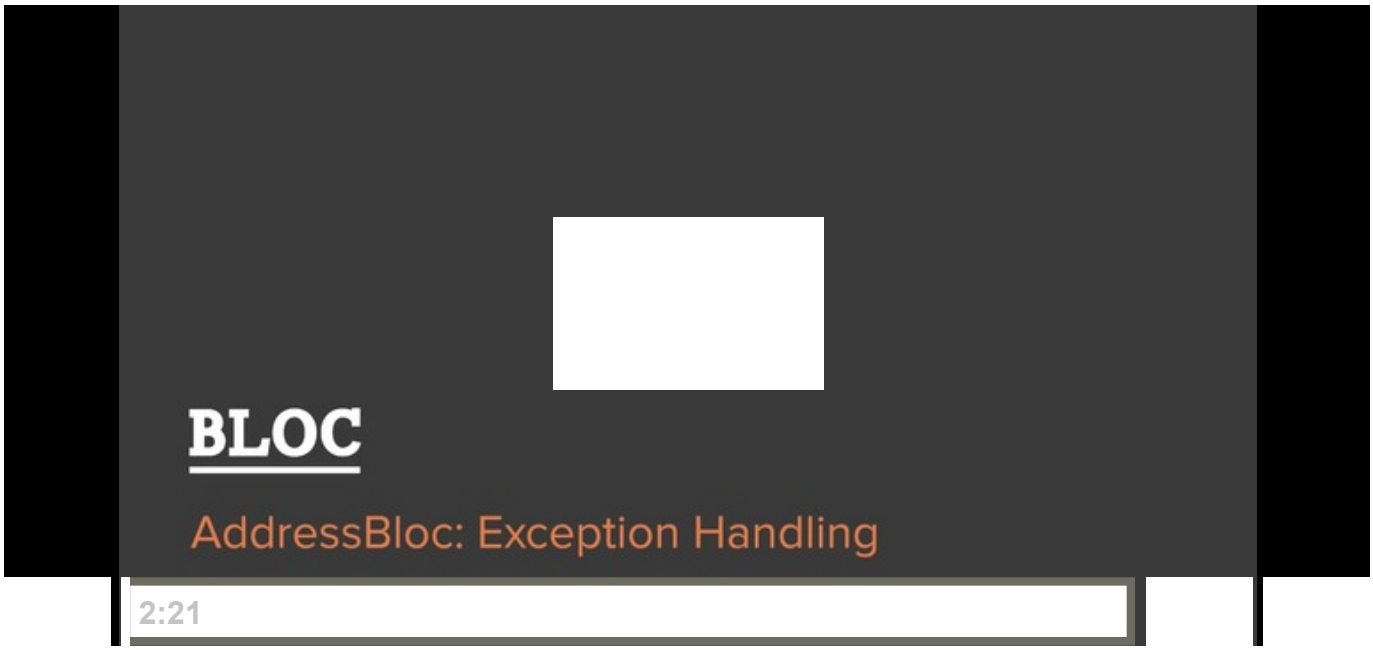
At **#3**, we import the specified file with `import_from_csv` on `address_book`. We then clear the screen and print the number of entries that were read from the file. All of these commands are wrapped in a `begin/rescue` block. `begin` will protect the program from crashing if an exception is thrown.

> An exception is like a yellow card in soccer. When a player commits a foul, the referee gives the player a warning by issuing a yellow card, but the player is allowed to continue to play.

In Ruby, if the program performs an operation that is illegal (such as dividing a rational number by zero) then Ruby will throw an exception, but the program is allowed to continue executing at the `rescue` statement. Our `begin` and `rescue` block catches potential exceptions and handles them by printing an error message and calling `import_from_csv`

again.

Let's explore Ruby exception handling:



Delete

Let's add the ability to delete an entry:

controllers/menu_controller.rb

```
+   def delete_entry(entry)
+     address_book.entries.delete(entry)
+     puts "#{entry.name} has been deleted"
+   end
```

We remove `entry` from `address_book` and print out a message to the user that says `entry` has been removed. Let's add the ability to edit an entry.

Edit

controllers/menu_controller.rb

```
+   def edit_entry(entry)
  # #4
+     print "Updated name: "
+     name = gets.chomp
+     print "Updated phone number: "
+     phone_number = gets.chomp
+     print "Updated email: "
+     email = gets.chomp
  # #5
+     entry.name = name if !name.empty?
+     entry.phone_number = phone_number if !phone_number.empty?
+     entry.email = email if !email.empty?
+     system "clear"
  # #6
+     puts "Updated entry:"
+     puts entry
+   end
```

At **#4**, we perform a series of `print` statements followed by `gets.chomp` assignment statements. Each `gets.chomp` statement gathers user input and assigns it to an appropriately named variable.

At **#5**, we use `!attribute.empty?` to set attributes on entry only if a valid attribute was read from user input.

At **#6**, we print out `entry` with the updated attributes.

Connect the Delete and Edit Methods

We wrote `delete_entry` and `edit_entry` but we still need to connect them to `main_menu`. An entry can only be deleted or edited directly from the submenu. Let's connect them directly from `entry_submenu`:

controllers/menu_controller.rb

```
  def entry_submenu(entry)
    puts "\nn - next entry"
    puts "d - delete entry"
    puts "e - edit this entry"
    puts "m - return to main menu"

    selection = gets.chomp

    case selection
    when "n"
    when "d"
  # #7
+    delete_entry(entry)
    when "e"
  # #8
+     edit_entry(entry)
+     entry_submenu(entry)
    when "m"
      system "clear"
      main_menu
    else
      system "clear"
      puts "#{selection} is not a valid input"
      entry_submenu(entry)
    end
  end
```

At **#7**, when a user is viewing the submenu and they press `d`, we call `delete_entry`. After the entry is deleted, control will return to `view_all_entries` and the next entry will be displayed.

At **#8**, we call `edit_entry` when a user presses `e`. We then display a sub-menu with `entry_submenu` for the entry under edit.

Search

The last task is to write the remainder of `search_entries`.

controllers/menu_controller.rb

```
    def search_entries
  # #9
+     print "Search by name: "
+     name = gets.chomp
  # #10
+     match = address_book.binary_search(name)
+     system "clear"
  # #11
+     if match
+       puts match.to_s
+       search_submenu(match)
+     else
+       puts "No match found for #{name}"
+     end
    end
```

At **#9**, we get the name that the user wants to search for and store it in `name`.

At **#10**, we call `search` on `address_book` which will either return a match or `nil`, it will never return an empty string since `import_from_csv` will fail if an entry does not have a name.

At **#11**, we check if `search` returned a match. This expression evaluates to *false* if `search` returns `nil` since `nil` evaluates to *false* in Ruby. If `search` finds a match then we call a helper method called `search_submenu`. `search_submenu` displays a list of operations that can be performed on an `Entry`. We want to give the user the ability to delete or edit an entry and return to the main menu when a match is found. Let's write `search_submenu`:

controllers/menu_controller.rb

```
+   def search_submenu(entry)
  # #12
+     puts "\nd - delete entry"
+     puts "e - edit this entry"
+     puts "m - return to main menu"
  # #13
+     selection = gets.chomp
+
  # #14
+     case selection
+     when "d"
+       system "clear"
+       delete_entry(entry)
+       main_menu
+     when "e"
+       edit_entry(entry)
+       system "clear"
+       main_menu
+     when "m"
+       system "clear"
+       main_menu
+     else
+       system "clear"
+       puts "#{selection} is not a valid input"
+       puts entry.to_s
+       search_submenu(entry)
+     end
+   end
```

At **#12**, we print out the submenu for an entry.

At **#13**, we save the user input to `selection`.

At **#14**, we use a `case` statement and take a specific action based on user input. If the user input is `d` we call `delete_entry` and after it returns we call `main_menu`. If the input is `e` we call `edit_entry`. `m` will return the user to the main menu. If the input does not match anything (see the `else` statement) then we clear the screen and ask for their input again by calling `search_submenu`.

Recap

Concept	Description
`begin` and `rescue`	`begin` and `rescue` blocks are Ruby's implementation of try/catch blocks in other languages. **Exception handling** is a very important part of any programming language and allows your program to dynamically rebound from any sort of unexpected error that gets *raised* during execution. `begin/rescue` blocks should not be abused. They should only be used when we know that there is a chance that a particular exception may occur.
`View`	A `View` "requests information from the model that it uses to generate an output representation to the user" in `MVC`. In Address Bloc, the command line menu we built operates as our `View` in the `MVC` pattern.

23. AddressBloc: Menu Continued

Assignment | **Discussion** | **Submission**

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

- Create a menu option to delete all entries. Name it something extreme like `detonate`, `demolish` or `nuke`.

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*.

Menu Continued Solution

assignment completed

?