20 Address Bloc: Menu

> “I don't want to hear the specials. If they're so special, put 'em on the menu.”
>
> — Jerry Seinfeld

Introduction

With models to store and retrieve data, we have the foundation we need to make Address Bloc an *interactive* experience. We will give users a command-line menu that allows them to view entries, create entries, search for a specific entry, import entries from a file, and exit the program. To do this, we'll create a `MenuController` to process user selections, update the models, and present information to the user.

Git

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

Create MenuController

Create a directory to store `MenuController`:

Terminal

```
$ cd address-bloc
$ mkdir controllers
$ touch controllers/menu_controller.rb
```

`MenuController` will need to connect to `AddressBook`. It will also need methods to display the main menu and process user input:

controllers/menu_controller.rb

```
  # #1
+ require_relative '../models/address_book'
+
+ class MenuController
+   attr_reader :address_book
+
+   def initialize
+     @address_book = AddressBook.new
+   end
+
+   def main_menu
  # #2
+     puts "Main Menu - #{address_book.entries.count} entries"
+     puts "1 - View all entries"
+     puts "2 - Create an entry"
+     puts "3 - Search for an entry"
+     puts "4 - Import entries from a CSV"
+     puts "5 - Exit"
+     print "Enter your selection: "
+
  # #3
+     selection = gets.to_i
+     puts "You picked #{selection}"
+   end
+ end
```

At **#1**, include `AddressBook` using `require_relative`. At **#2**, display the main menu options to the command line. At **#3**, retrieve user input from the command line using `gets`. `gets` reads the next line from **standard input**.

Let's watch a video that explains the difference between `puts` and `gets`:

Use `MenuController` in the driver program we created:

address_bloc.rb

```
+ require_relative 'controllers/menu_controller'

  # #4
+ menu = MenuController.new
  # #5
+ system "clear"
  puts "Welcome to AddressBloc!"
  # #6
+ menu.main_menu
```

At **#4**, create a new `MenuController` when `AddressBloc` starts. At **#5**, use `system "clear"` to clear the command line. At **#6**, call `main_menu` to display the menu.

Give `AddressBloc` a quick test run:

Terminal

```
$ ruby address_bloc.rb
Welcome to AddressBloc!
Main Menu - 0 entries
1 - View all entries
2 - Create an entry
3 - Search for an entry
4 - Import entries from a CSV
5 - Exit
Enter your selection:
```

Handling User Input

`MenuController` asks for user input, and then exits. Update `main_menu` to process user input and stub out the methods we'll need:

controllers/menu_controller.rb

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

  class MenuController
    attr_reader :address_book

    def initialize
      @address_book = AddressBook.new
    end

    def main_menu
      puts "Main Menu - #{address_book.entries.count} entries"
      puts "1 - View all entries"
      puts "2 - Create an entry"
      puts "3 - Search for an entry"
      puts "4 - Import entries from a CSV"
      puts "5 - Exit"
      print "Enter your selection: "

      selection = gets.to_i
-     puts "You picked #{selection}"
  # #7
+     case selection
+     when 1
+       system "clear"
+       system "clear"
+       view_all_entries
+       main_menu
+     when 2
+       system "clear"
+       create_entry
+       main_menu
+     when 3
+       system "clear"
+       search_entries
+       main_menu
+     when 4
+       system "clear"
+       read_csv
+       main_menu
+     when 5
+       puts "Good-bye!"
  # #8
+       exit(0)
  # #9
+     else
+       system "clear"
+       puts "Sorry, that is not a valid input"
+       main_menu
+     end
    end
+
  # #10
+   def view_all_entries
+   end
+
+   def create_entry
+   end
+
+   def search_entries
+   end
+
+   def read_csv
+   end
  end
```

At **#7**, use a `case` **statement operator** to determine the proper response to the user's input. At **#8**, terminate the program using `exit(0)`. `0` signals the program is exiting without an error. At **#9**, use an `else` to catch invalid user input and prompt the user to retry. At **#10**, stub the rest of the methods called in `main_menu`.

Run `AddressBloc` again. Confirm that you can make selections and that the program will continue to run until you tell it to exit.

Create an Entry

An address book is only useful if we can create new entries. Let's give our users a way to add entries to `AddressBloc`:

controllers/menu_controller.rb

```
     def create_entry
   # #11
+      system "clear"
+      puts "New AddressBloc Entry"
   # #12
+      print "Name: "
+      name = gets.chomp
+      print "Phone number: "
+      phone = gets.chomp
+      print "Email: "
+      email = gets.chomp
+
   # #13
+      address_book.add_entry(name, phone, email)
+
+      system "clear"
+      puts "New entry created"
     end
```

At **#11**, clear the screen for before displaying the create entry prompts. At **#12**, use `print` to prompt the user for each `Entry` attribute. `print` works just like `puts`, except that it doesn't add a newline. At **#13**, add a new entry to `address_book` using `add_entry` to ensure that the new entry is added in the proper order.

View Entries

Now that we can add entries, we want to be able to view them as well:

controllers/menu_controller.rb

```
    def view_all_entries
  # #14
+     address_book.entries.each do |entry|
+       system "clear"
+       puts entry.to_s
  # #15
+       entry_submenu(entry)
+     end
+
+     system "clear"
+     puts "End of entries"
    end
```

At **#14**, iterate through all entries in `AddressBook` using `each`. At **#15**, we call `entry_submenu` to display a submenu for each entry. Let's add this method at the bottom of `MenuController`.

controllers/menu_controller.rb

```
+   def entry_submenu(entry)
  # #16
+     puts "n - next entry"
+     puts "d - delete entry"
+     puts "e - edit this entry"
+     puts "m - return to main menu"
+
  # #17
+     selection = gets.chomp
+
+     case selection
  # #18
+     when "n"
  # #19
+     when "d"
+     when "e"
  # #20
+     when "m"
+       system "clear"
+       main_menu
+     else
+       system "clear"
+       puts "#{selection} is not a valid input"
+       entry_submenu(entry)
+     end
+   end
```

#16, display the submenu options. **#17**, `chomp` removes any trailing whitespace from the string `gets` returns. This is necessary because `"m "` or `"m\n"` won't match `"m"`. **#18**, when the user asks to see the next entry, we can do nothing and control will be returned to `view_all_entries`. At **#19**, we'll handle deleting and editing in another checkpoint, for now the user will be shown the next entry. At **#20**, we return the user to the main menu.

Run `AddressBloc` and test adding and viewing users.

Git

Commit your checkpoint work in Git. See **Git Checkpoint Workflow: After Each Checkpoint** for details.

Recap

Concept	Description
Controller	**Controllers** process user input, update the model, and presents model information.
Standard input	Standard input is data going into a program. By default standard input is expected from the same keyboard which started the program.
`case` **statement operator**	Ruby's `case` statement operator is used to manage more complicated control flow. It can be used as a cleaner alternative to multiple `if` statements.

20. Address Bloc: Menu

Assignment	**Discussion**	**Submission**

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

Modify `main_menu` to give users the ability to view a specific entry by number:

- Add a new option to the main menu: "View Entry Number n".
- Once the user selects the new option, ask for the entry number and display that entry to the user.
- Handle invalid input by prompting the user to enter a valid entry number.

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