Terminal App

Presented by Benjamin Stuart

Design:

Goal:

- A cafe point of sale (POS) terminal application that allows users to log in and record orders for a table.

- At the end of the customer experience the employee can then tabulate their orders in a printable layout to display the orders, order price and bill total.

Target Audience:

- Cafe owners and cafe managers to make their point of sale more efficient.

Gems:

TTY-Prompt
TTY-Table
Coloriz/se
Json

Features

Menu interface: Can be used to navigate the entire pos

Business Setup: Used to create and re-create the setup for business POS. (e.g. add all employees, add all menu-items etc)

Login Menu: prevents malicious use of the POS.

Users can track tables: Can tabulate and view the bill for a table as well as additional features such as add menu-items to a tables order.

Manager Permissions: Controls access to POS settings where features like remove/add staff (and more) exist

Save/Load Functionality: Saves and loads the program.

Parent Classes

Menu Class - > Represents the generic menu prompt that is inherited by all Menu subclasses

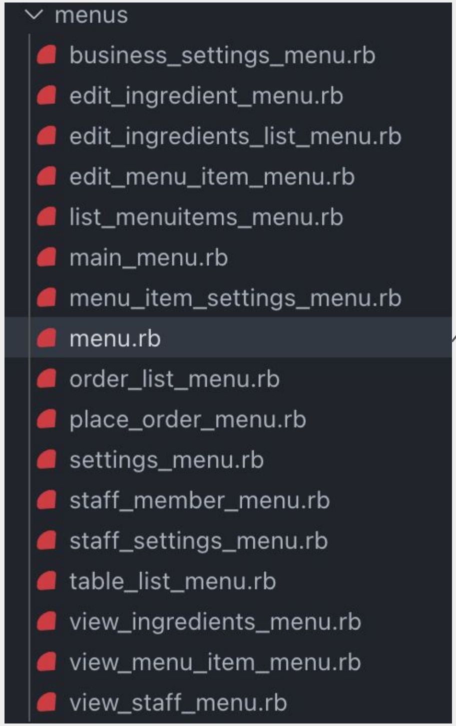
Staff Class - > Represents the individual employees that belong to the business

MenuItem - > Represents the items on the businesses menu

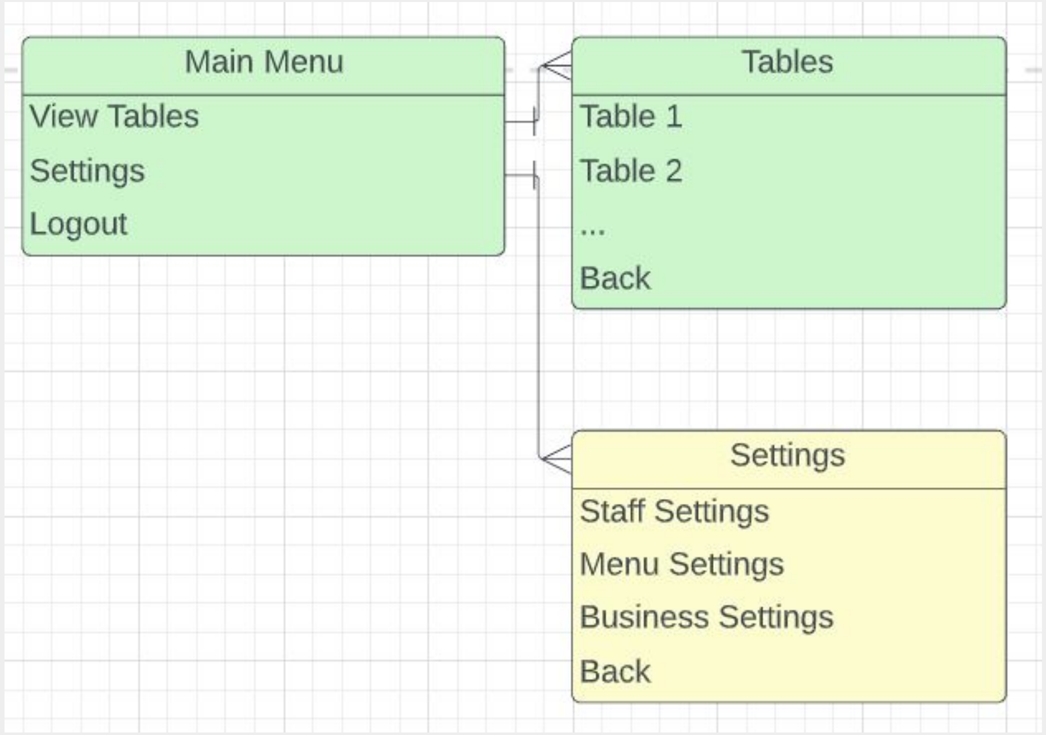
Table Class - > Represents the tables in the cafe

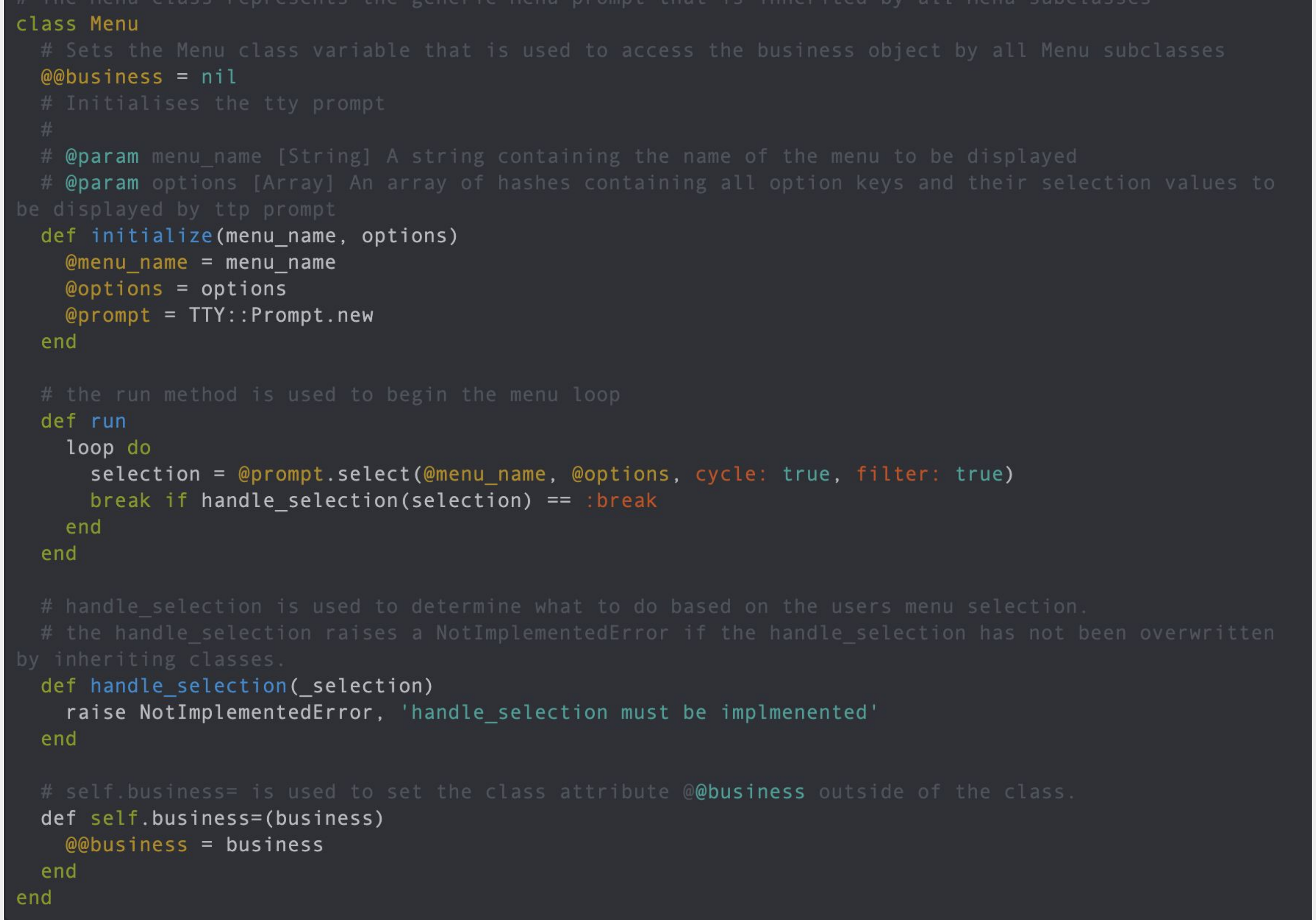
Business - > Represents the cafe

Feature 1: Menu Interface

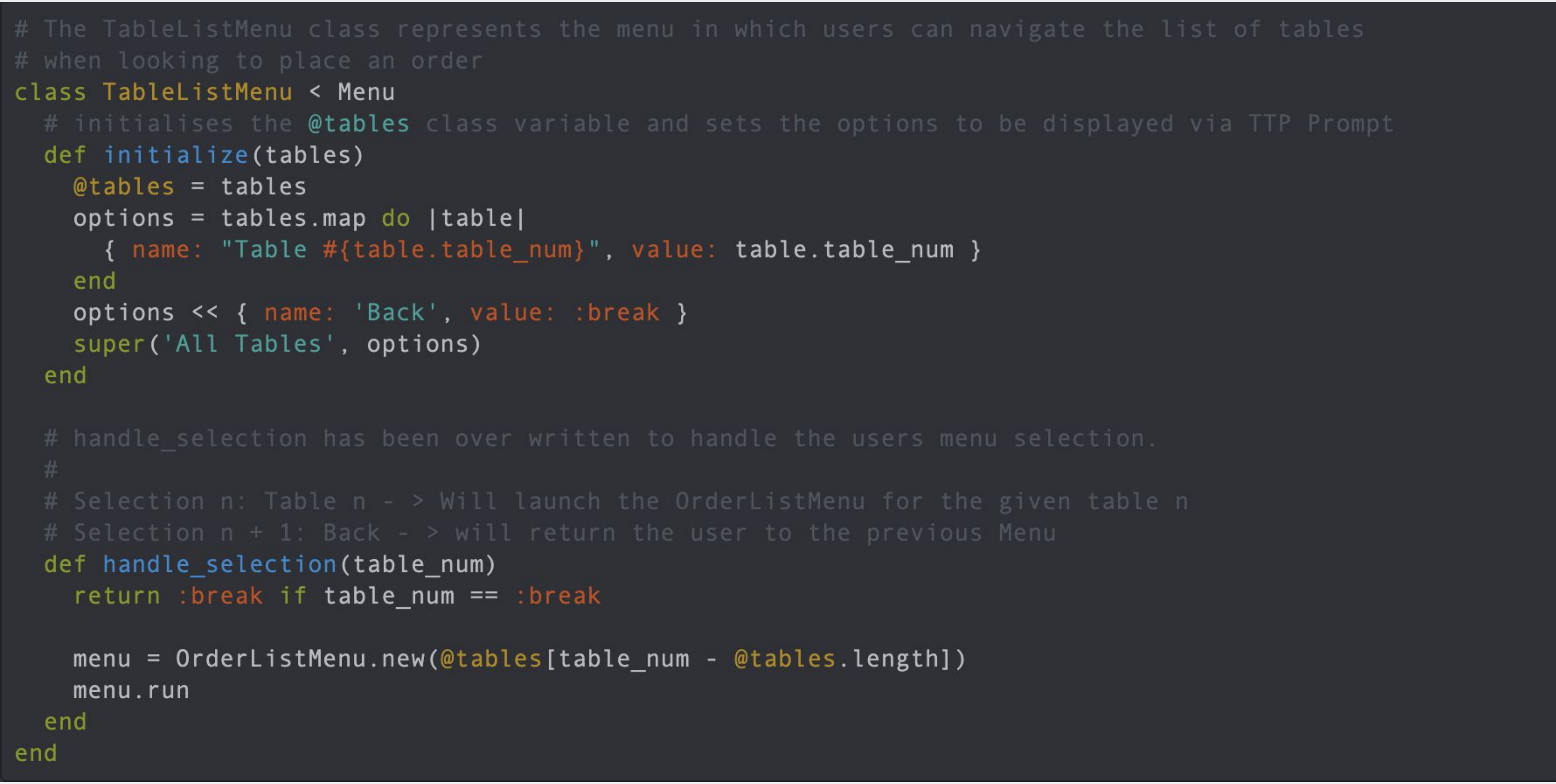

```
menus
  business_settings_menu.rb
  edit_ingredient_menu.rb
  edit_ingredients_list_menu.rb
  edit_menu_item_menu.rb
  list_menuitems_menu.rb
  main_menu.rb
  menu_item_settings_menu.rb
  menu.rb
  order_list_menu.rb
  place_order_menu.rb
  settings_menu.rb
  staff_member_menu.rb
  staff_settings_menu.rb
  table_list_menu.rb
  view_ingredients_menu.rb
  view_menu_item_menu.rb
  view_staff_menu.rb
```

Parent class



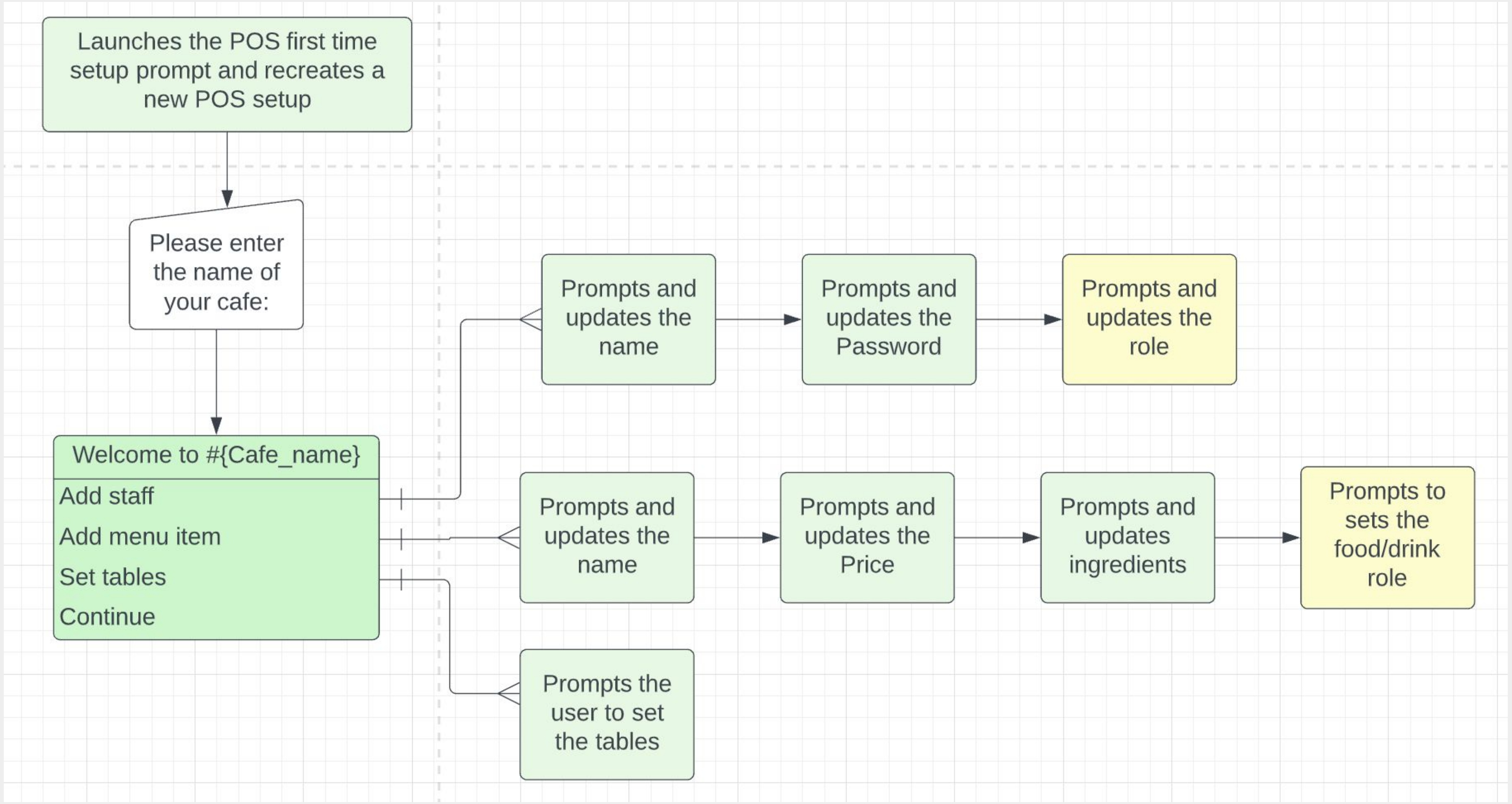


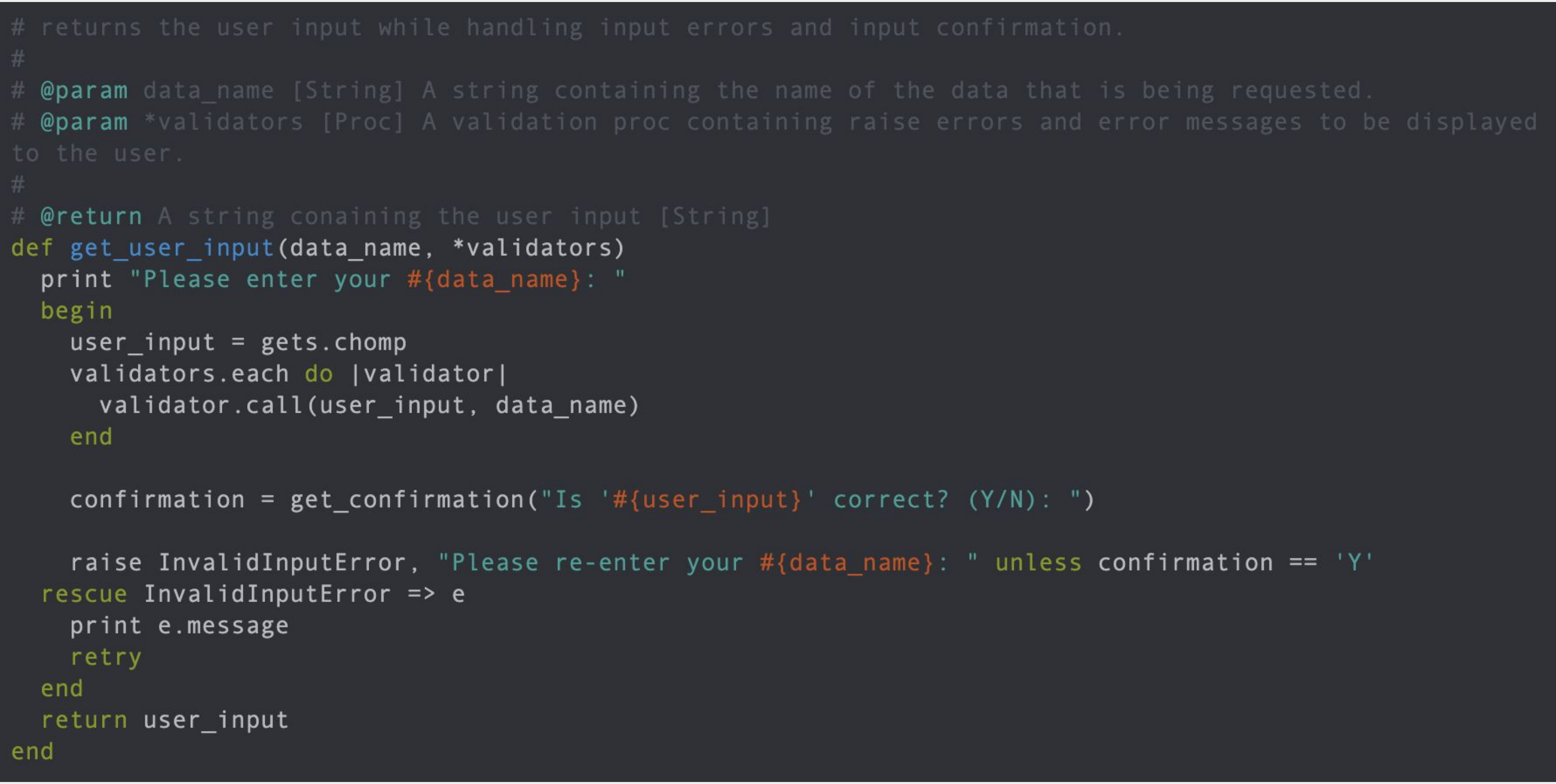
```ruby
# The Menu class represents the generic menu prompt that is inherited by all menu subclasses
class Menu
  # Sets the Menu class variable that is used to access the business object by all Menu subclasses
  @@business = nil
  # Initialises the tty prompt
  #
  # @param menu_name [String] A string containing the name of the menu to be displayed
  # @param options [Array] An array of hashes containing all option keys and their selection values to
be displayed by ttp prompt
  def initialize(menu_name, options)
    @menu_name = menu_name
    @options = options
    @prompt = TTY::Prompt.new
  end

  # the run method is used to begin the menu loop
  def run
    loop do
      selection = @prompt.select(@menu_name, @options, cycle: true, filter: true)
      break if handle_selection(selection) == :break
    end
  end

  # handle_selection is used to determine what to do based on the users menu selection.
  # the handle_selection raises a NotImplementedError if the handle_selection has not been overwritten
by inheriting classes.
  def handle_selection(_selection)
    raise NotImplementedError, 'handle_selection must be implmenented'
  end

  # self.business= is used to set the class attribute @@business outside of the class.
  def self.business=(business)
    @@business = business
  end
end
```



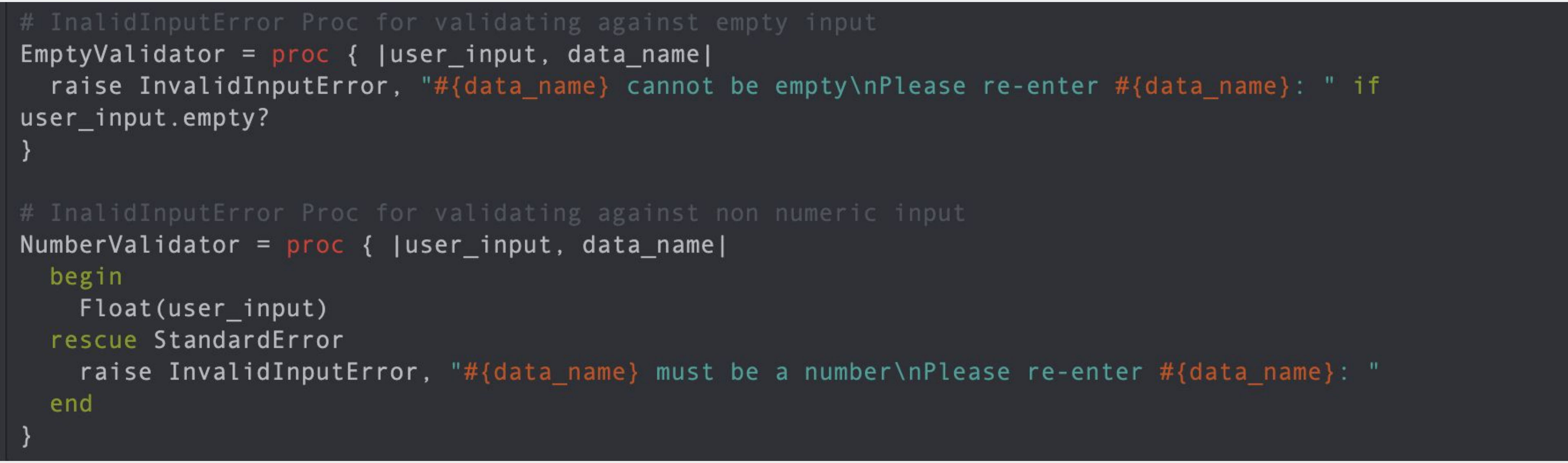
```ruby
# The TableListMenu class represents the menu in which users can navigate the list of tables
# when looking to place an order
class TableListMenu < Menu
  # initialises the @tables class variable and sets the options to be displayed via TTP Prompt
  def initialize(tables)
    @tables = tables
    options = tables.map do |table|
      { name: "Table #{table.table_num}", value: table.table_num }
    end
    options << { name: 'Back', value: :break }
    super('All Tables', options)
  end

  # handle_selection has been over written to handle the users menu selection.
  #
  # Selection n: Table n - > Will launch the OrderListMenu for the given table n
  # Selection n + 1: Back - > will return the user to the previous Menu
  def handle_selection(table_num)
    return :break if table_num == :break

    menu = OrderListMenu.new(@tables[table_num - @tables.length])
    menu.run
  end
end
```

Feature 2: Business Setup

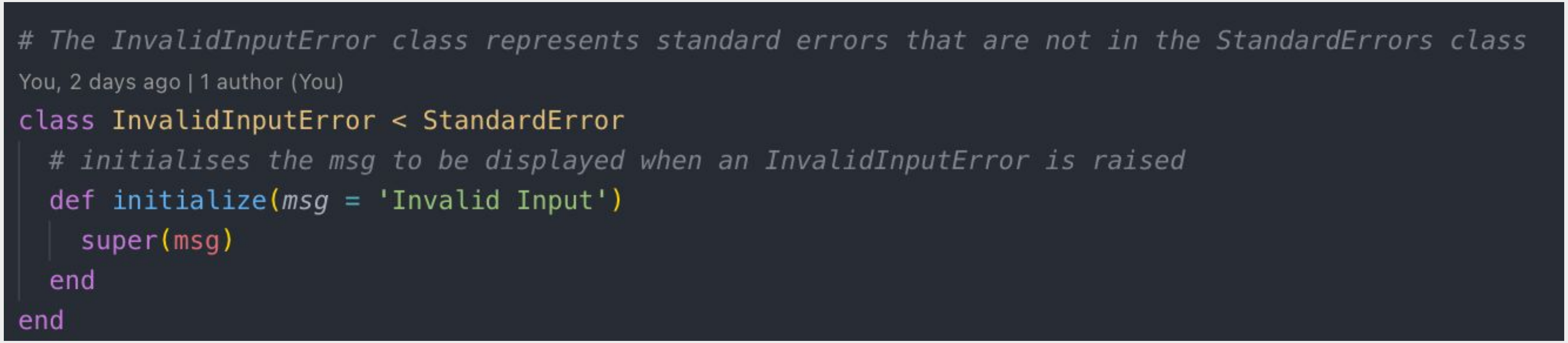

Launches the POS first time setup prompt and recreates a new POS setup
Please enter the name of your cafe:
Welcome to #{Cafe_name}
Add staff
Add menu item
Set tables
Continue
Prompts and updates the name
Prompts and updates the Password
Prompts and updates the role
Prompts and updates the name
Prompts and updates the Price
Prompts and updates ingredients
Prompts to sets the food/drink role
Prompts the user to set the tables



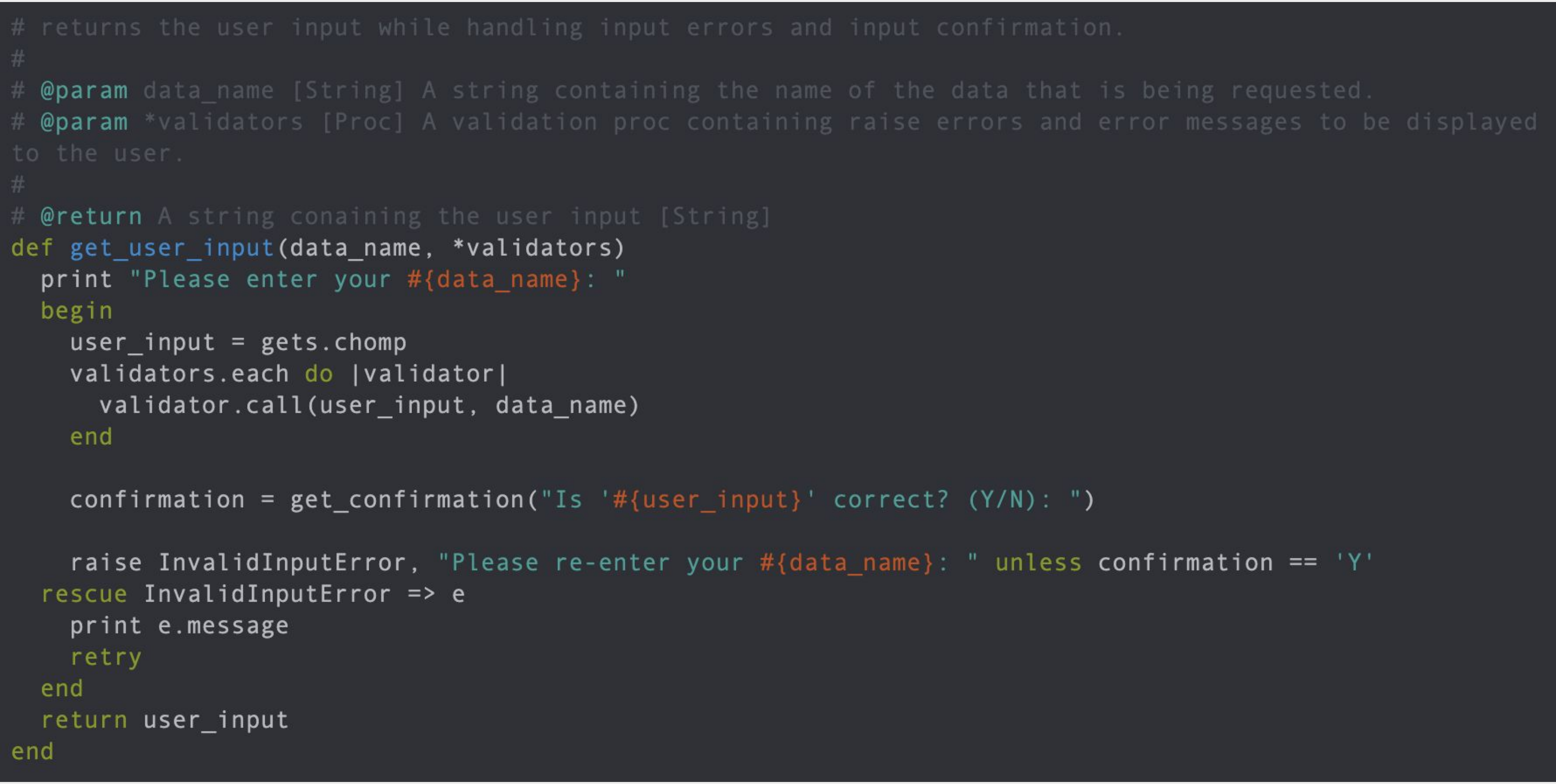
```ruby
# returns the user input while handling input errors and input confirmation.
#
# @param data_name [String] A string containing the name of the data that is being requested.
# @param *validators [Proc] A validation proc containing raise errors and error messages to be displayed
to the user.
#
# @return A string conaining the user input [String]
def get_user_input(data_name, *validators)
  print "Please enter your #{data_name}: "
  begin
    user_input = gets.chomp
    validators.each do |validator|
      validator.call(user_input, data_name)
    end

    confirmation = get_confirmation("Is '#{user_input}' correct? (Y/N): ")

    raise InvalidInputError, "Please re-enter your #{data_name}: " unless confirmation == 'Y'
  rescue InvalidInputError => e
    print e.message
    retry
  end
  return user_input
end
```



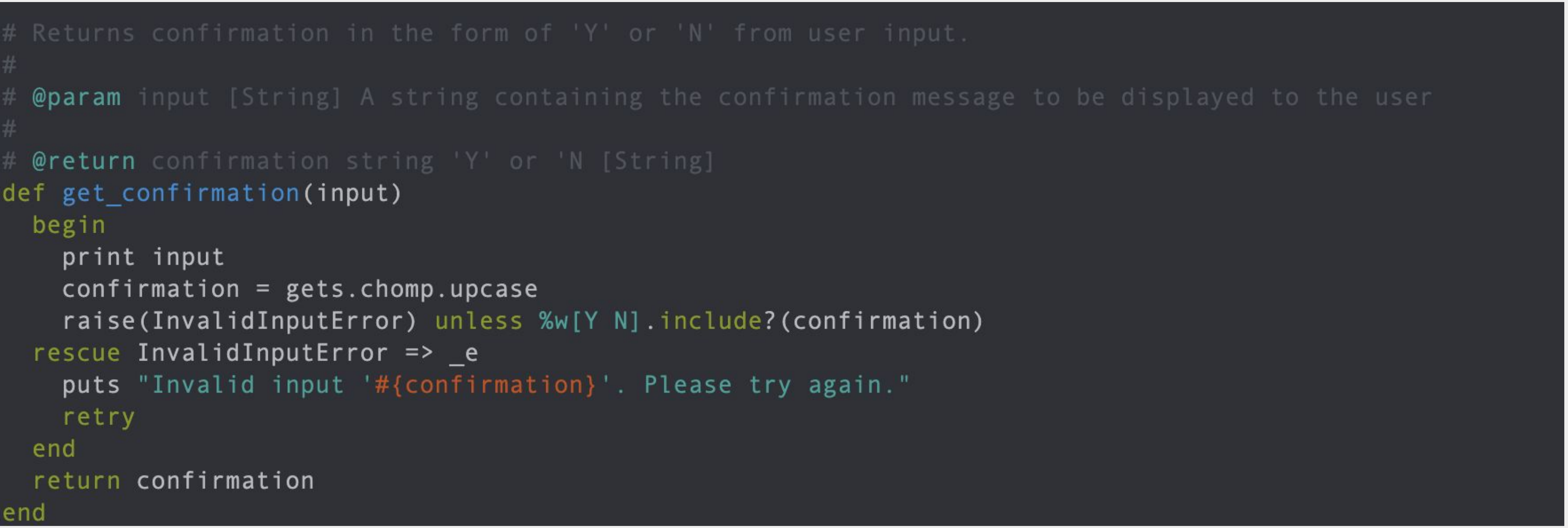
```
# InalidInputError Proc for validating against empty input
EmptyValidator = proc { |user_input, data_name|
  raise InvalidInputError, "#{data_name} cannot be empty\nPlease re-enter #{data_name}: " if
user_input.empty?
}

# InalidInputError Proc for validating against non numeric input
NumberValidator = proc { |user_input, data_name|
  begin
    Float(user_input)
  rescue StandardError
    raise InvalidInputError, "#{data_name} must be a number\nPlease re-enter #{data_name}: "
  end
}
```



```
# The InvalidInputError class represents standard errors that are not in the StandardErrors class
You, 2 days ago | 1 author (You)
class InvalidInputError < StandardError
  # initialises the msg to be displayed when an InvalidInputError is raised
  def initialize(msg = 'Invalid Input')
    super(msg)
  end
end
```

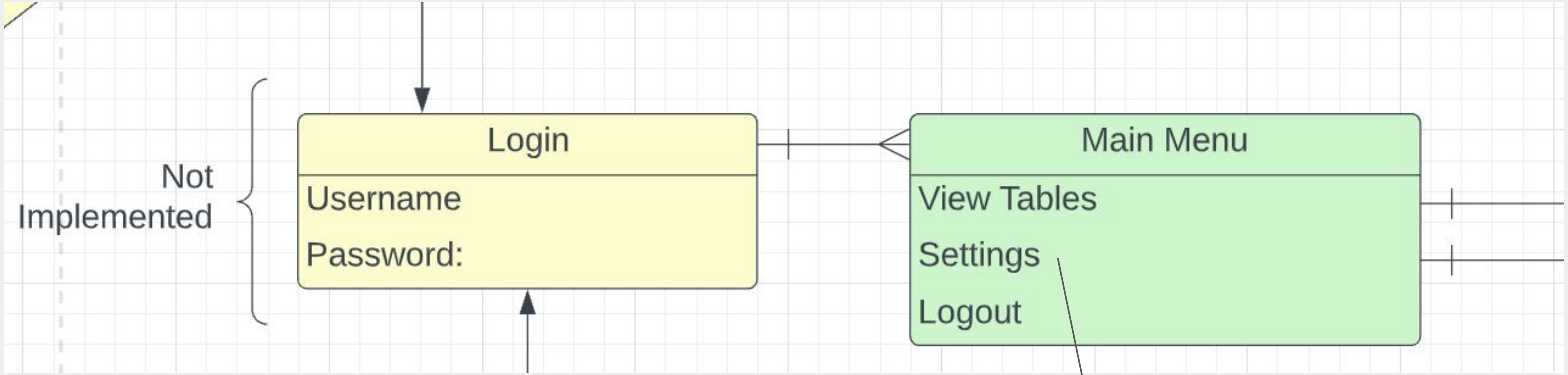


```ruby
# returns the user input while handling input errors and input confirmation.
#
# @param data_name [String] A string containing the name of the data that is being requested.
# @param *validators [Proc] A validation proc containing raise errors and error messages to be displayed
to the user.
#
# @return A string conaining the user input [String]
def get_user_input(data_name, *validators)
  print "Please enter your #{data_name}: "
  begin
    user_input = gets.chomp
    validators.each do |validator|
      validator.call(user_input, data_name)
    end

    confirmation = get_confirmation("Is '#{user_input}' correct? (Y/N): ")

    raise InvalidInputError, "Please re-enter your #{data_name}: " unless confirmation == 'Y'
  rescue InvalidInputError => e
    print e.message
    retry
  end
  return user_input
end
```



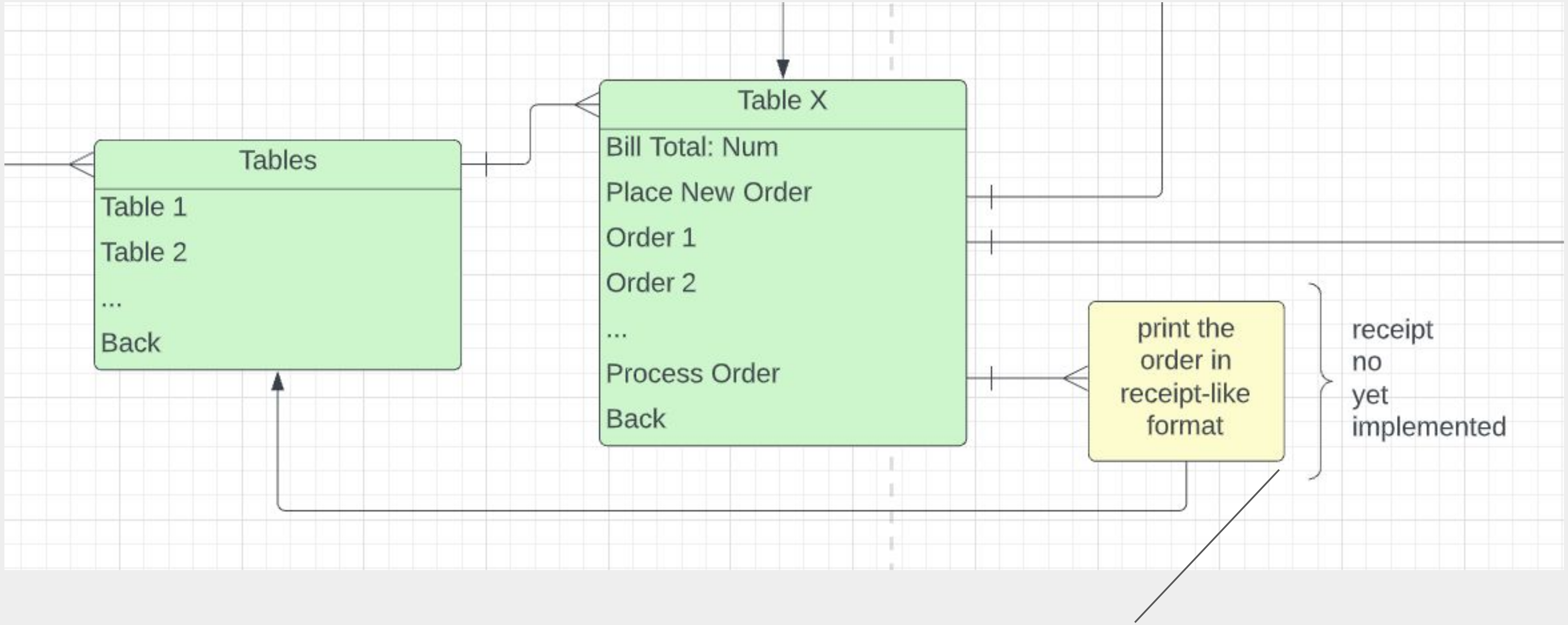
```ruby
# Returns confirmation in the form of 'Y' or 'N' from user input.
#
# @param input [String] A string containing the confirmation message to be displayed to the user
#
# @return confirmation string 'Y' or 'N [String]
def get_confirmation(input)
  begin
    print input
    confirmation = gets.chomp.upcase
    raise(InvalidInputError) unless %w[Y N].include?(confirmation)
  rescue InvalidInputError => _e
    puts "Invalid input '#{confirmation}'. Please try again."
    retry
  end
  return confirmation
end
```

Feature 3: Login Menu



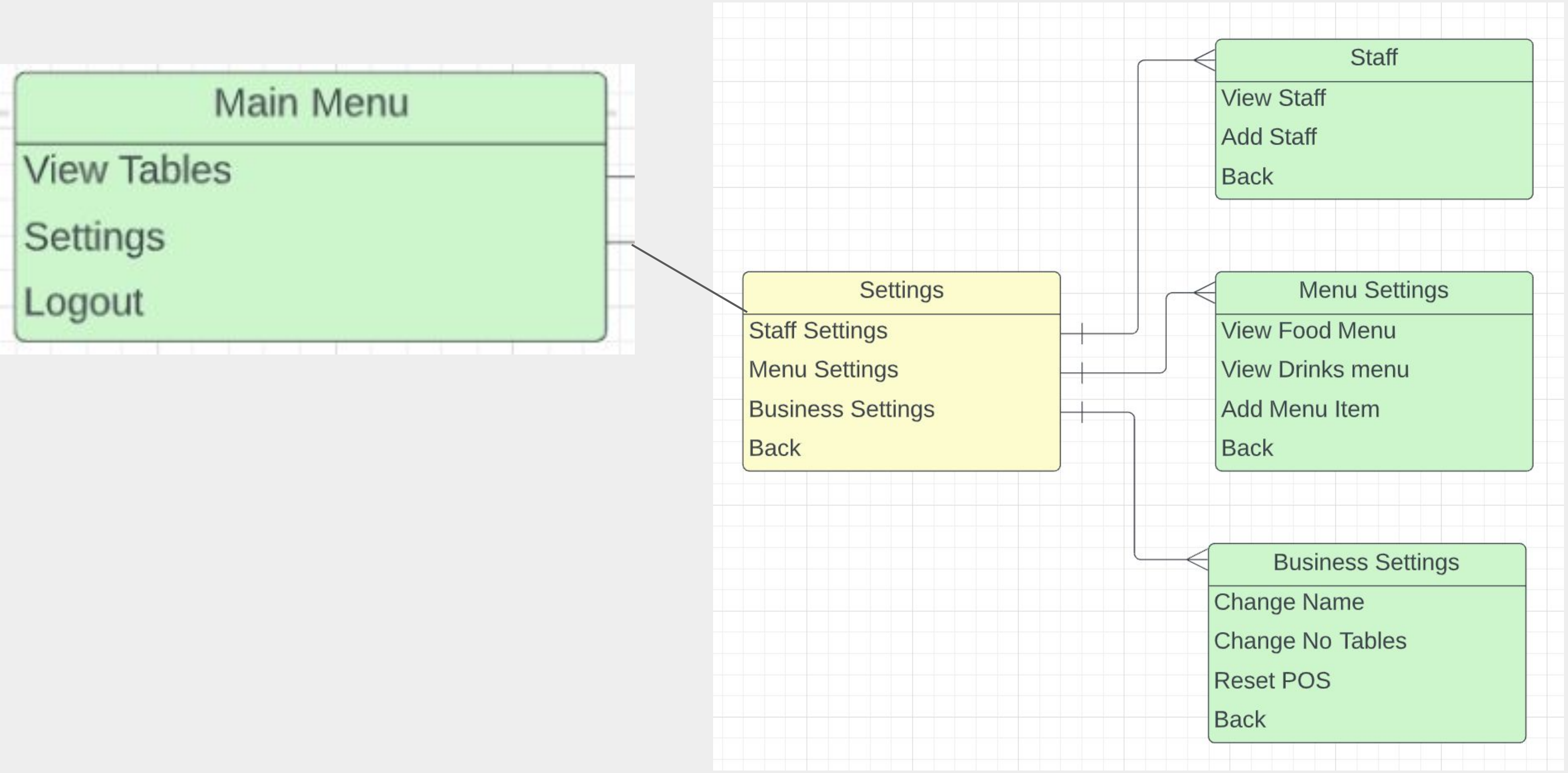
Managers Only: The whole settings branch will be locked if staff member object does not have manager role

Feature 4: Users can track tables



Will generate the receipt using TTY Table gems

Feature 5: Manager Permissions



Feature 6: Save/Load Functionality

1. On startup my application will check if the save filepath has a json save file.

2. If the Savefile is there it will load the savefile into memory to run the program.

3. If the safefile has not been made it will launch the first time setup of the application.

4. After changes are made in the pos (such as orders added to a table, staff are added to the pos etc) all objects will be converted into hashes and saved in the json file.

Review:

Challenges/issues:

1. Trying to develop the menu classes without a flowchart.
2. Finding Gems.
3. Writing reusable code.
4. Will be saving over everything every single time.

Favourite Parts:

1. Flowchart - it made writing my menu classes and methods more efficient.
2. Using procs to dry up the code - really cool feature of programming languages that opens a whole new world of possibilities
3. Using the NotImplementedError - A cool feature that made it so my program did not break as I was coding the menus.
4. Seeing it all come together