Learning Outcomes

Upon completing this assignment, students should be able to

- Write a *reusable* class that can easily be put into future projects.
- Implement an object, `Menu`, that maintains a collection (list) of a *hidden* **inner** object, `MenuItem`. **(This is review.)**
- Read a *file* with an arbitrary number of text descriptions of objects of a single type and insert them into a `List`. **(This is review.)**
- Document a thorough test plan for a reusable object.

Introduction

It would be nice to have a standard user interface (UI) for interacting with our programs this semester. To that end, this first assignment is to implement a `Menu` class. The items in the menu will be loaded at runtime rather than part of the program.

If this class is going to be used, over and over again, you need to have confidence that it is not adding bugs to your later programs. This requires testing that it works, which means understanding what it *should* do. And testing that it does just that.

Method

Getting Started

There will be three files in this program: `Menu.java`, `RunMenu.java`, and `TestMenu.java`. Both `RunMenu` and `TestMenu` will have `main` methods, one to run the actual program (which, in this case, is not much more than doing really simple arithmetic) and the other to do its best to automate some testing of `Menu`.

`Menu`

The `Menu` class is a wrapper around a `List` (you will use the `ArrayList` implementation in the Java standard library) of `MenuItem` objects. The `MenuItem` class is an inner class of `Menu` and should not leak to other classes.

The primary constructor for `Menu` should take a `Scanner` and read lines from it, using groups of lines to create `MenuEntry` objects. There should be a secondary constructor that takes an `InputStream` and wraps it in a `Scanner` before calling the primary constructor, something like this:

```
public Menu(InputStream in) {
  this(new Scanner(in));
}
```

(Remember that **`this`**, treated as a function, as the *first* line of a constructor, calls another constructor. So, the one taking an `InputStream` does no work except make a `Scanner` for the primary constructor.)

The primary constructor will read triples of lines from the `Scanner`: the `name` of the item, its `displayText` and its `helpText`. When it gets a pair, it should construct a new `MenuItem` with those two values and add it to its collection of `items`.

`Menu` has two public methods: `match` and `help`.

`String match(String prompt)` does the following:

```
display all items
prompt user for an item ->user_name
while (user_name is not the name of any item in the menu)
  print an error message
  prompt user for an item -> user_name

return user_name
```

Displaying an item is to print the name, a dash, and the display text. Notice that `match` will not return until the user enters a valid name of one of the items. If there is more than one item with a given name, that is not your problem.

`void help()` will print each item (that is name, dash, display text) and then, on the next line, indent the corresponding help text four spaces.

Input/Output

File Formats

`RunMenu` should initialize the menu with a text file that looks like the following:

```
+
add
prompts for a number and adds it to the total so far
-
minus
prompts for a number and subtracts it from the total so far
0
zero
resets the total so far to zero
h
help
display this help screen
q
quit
quit the program
```

User Interface

If `match("Action? ")` is called on this menu, then the following could be the user's interaction with it:

```
+ - add
- - minus
```

```
0 - zero
h - help
q - quit
Action? wombat
"wombat" fails to match.
Action? 2
"2" fails to match.
Action? 0
```

And the call would return the string `"0"`.

A call to `help()` would produce something like

```
+ - add
    prompts for a number and adds it to the total so far
- - minus
    prompts for a number and subtracts it from the total so far
0 - zero
    resets the total so far to zero
h - help
    display this help screen
q - quit
    quit the program
```

Design Considerations

Obviously `RunMenu` should perform the action described in the menu. So it should have an **`int`** total value (initialize it to zero) and update it as mentioned. For two of the choices you will prompt the user for an **`int`** and use that to change the total.

Testing

For the moment, you will supply some menu files for `TestMenu` to load and then prompt the actual user, making notes (in `README`) as to what to answer and what response to expect.

Make sure the right thing happens when the input file is empty, has one item, and has multiple items (these are constructor tests). Should you ever call `match` on an empty menu?

You should type out what should be seen for the calls to `match` and `help` for your test menus.

We will work on *automating* these tests soon.

Documentation

`README`

Must document how you tested. How do you know that it is right?

Must include instructions on how to **compile** and how to **run** the program as submitted.

Deliverables

Submission medium: `git` to Gitea at `cs-devel.potsdam.edu`.