Program Usage:

In order to run the application on Windows or MacOS/Linux, execute the following command in the terminal:

> java -cp ".;target/classes;json-simple-1.1.1.jar" default_proj.Main <file extension> <parking violations data filename> <properties data filename.csv> <population data filename.txt>

where the main class for the program entry point is "target/class/default_proj/Main" and the required library for parsing JSON files is "json-simple-1.1.1.jar" (located in the root directory of the project folder.) As indicated above, there are four mandatory arguments to be supplied by the user. The first argument must either specify "json" or "csv", which helps the program to identify the expected file extension in the file name provided as the second argument (either ".json" or ".csv".) The third argument must be the file name of a ".csv" file and the fourth argument must reference a ".txt" file. *Note* The user's current working directory needs to be the root directory of the project folder when executing the application, since the application's graphical user interface (GUI) loads images from the "data" folder, which is also located in the root directory.

Upon reading in valid arguments, the program reads in and parses the appropriate data regarding parking violations, properties, and population in Philadelphia. Then the GUI (the "window") is created and presented to the user, featuring hard-coded options ("buttons") related to a query that the user can perform. Most of the queries require the user to input a valid zip code. In this case, a separate small window will pop-up when the user left-clicks the associated button, which provides text space for the user to input a zip code. This pop-up window will also have buttons, labeled with "Ok" and "Cancel", for the user to confirm or cancel input, respectively. Confirming no input or invalid input for a zip code will essentially close and bring up the same pop-up window, except with an additional message indicating that invalid input was supplied.

When the user selects an input-less query or confirms valid input when prompted for a zip code, the application processes the relevant data and returns its result in the middle portion (the "paper") of the original window. For most queries, this will be a singular number. Only the "Parking Fines per Capita" button, when pressed, returns a two column list of zip codes accompanied by their respective parking fines per capita. ***It is recommended to select this query first, as the results show all the possible zip codes that will most likely return a non-zero value when provided as input for other queries.*** Lastly, for convenience, the application displays up to six "sticky notes" with names of previous searches, where the search names are

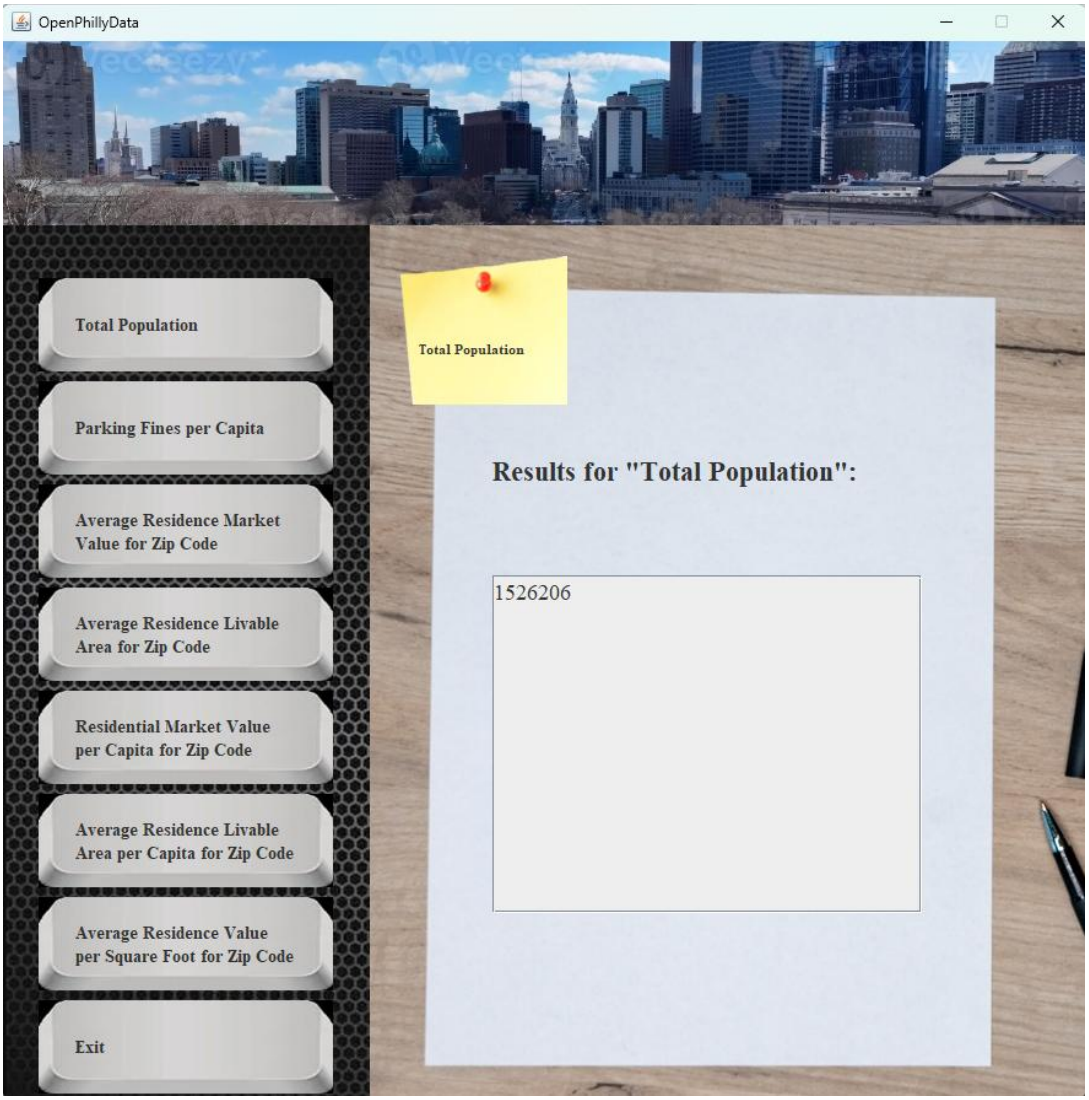
cycled through the sticky notes upon a successful query not currently reflected on the sticky notes.

Lastly, to cleanly close the program, either left-click on the “X” at the top-right of the original window or press on the “Exit” button. This immediately closes the application and restores the user’s access to the terminal.

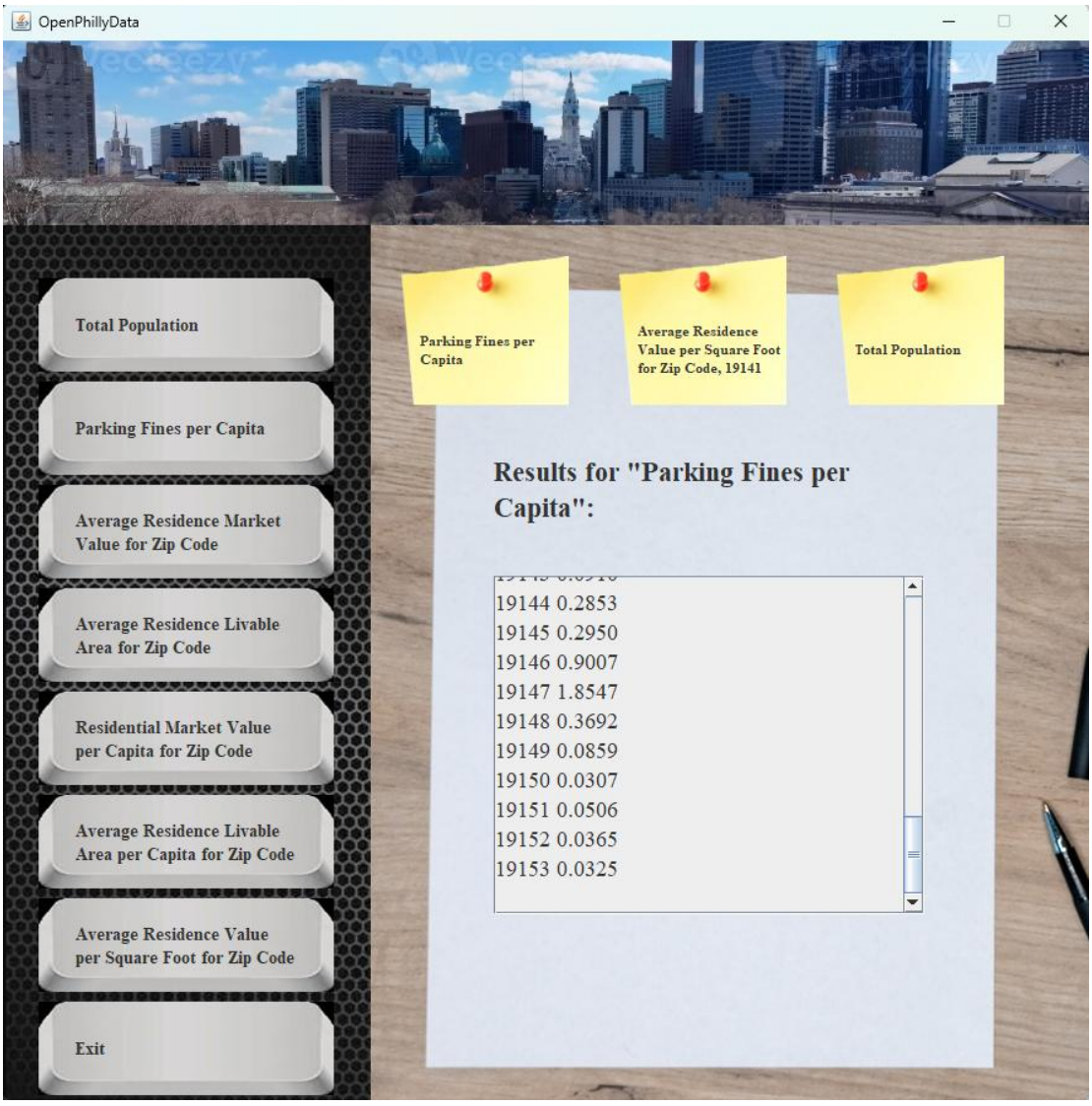
Usage Screenshots:

The initial application GUI/window that appears:

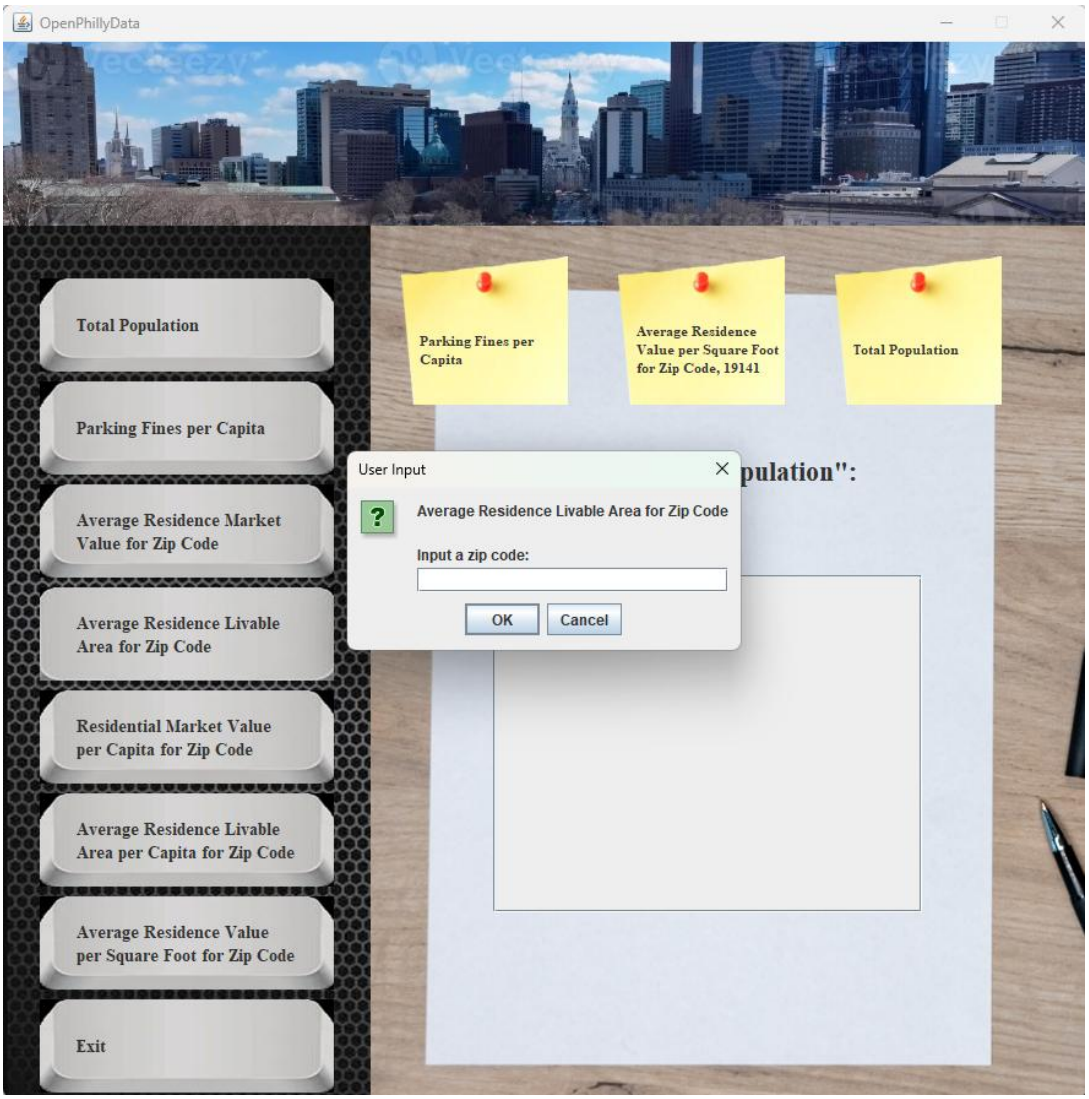
Performing a non-input search (pressing the “Total Population” button, for example):



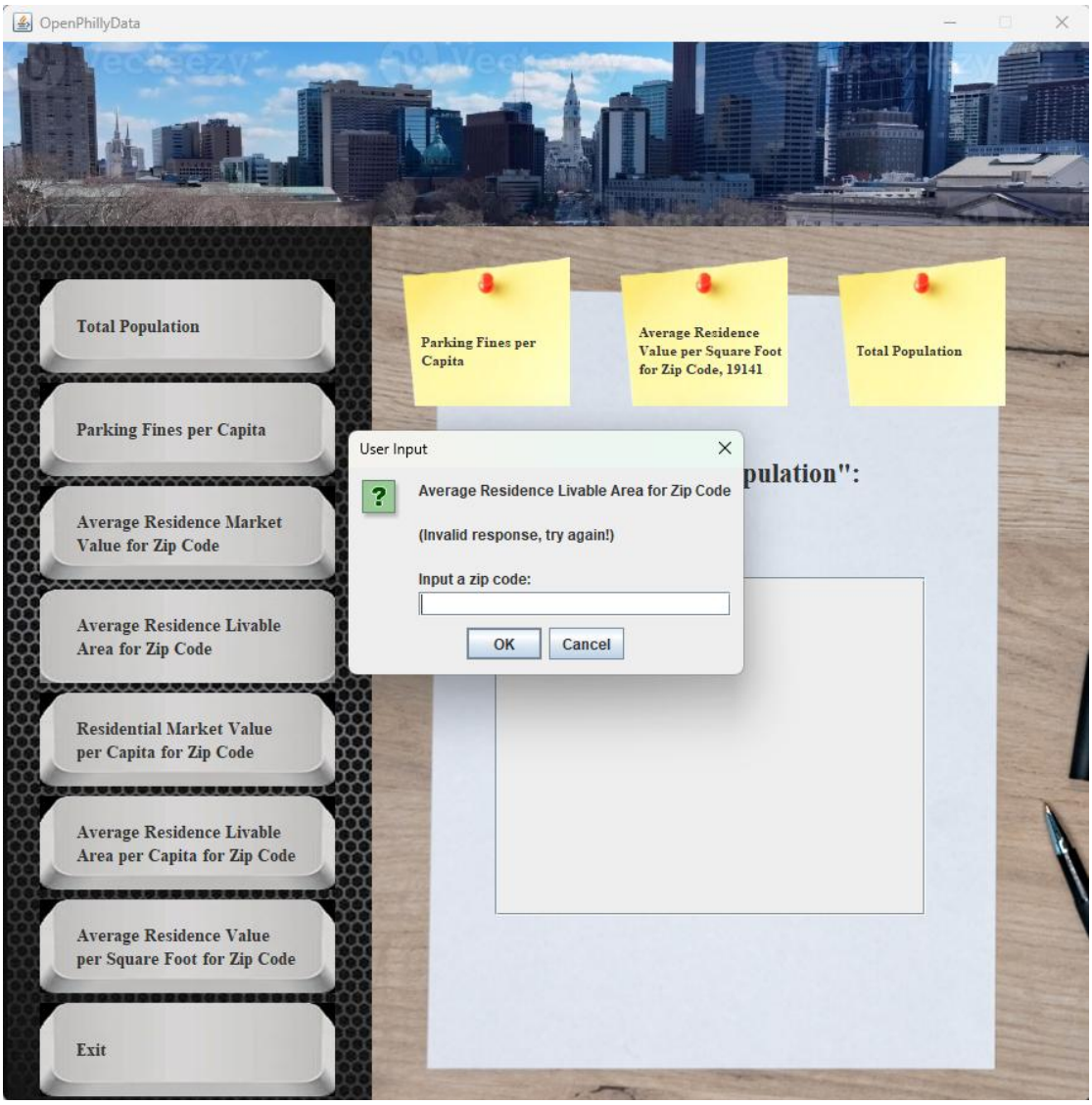
Returning a previous search result via left-clicking a sticky note (such as for referencing “Parking Fines per Capita” for future searches requiring zip code input):



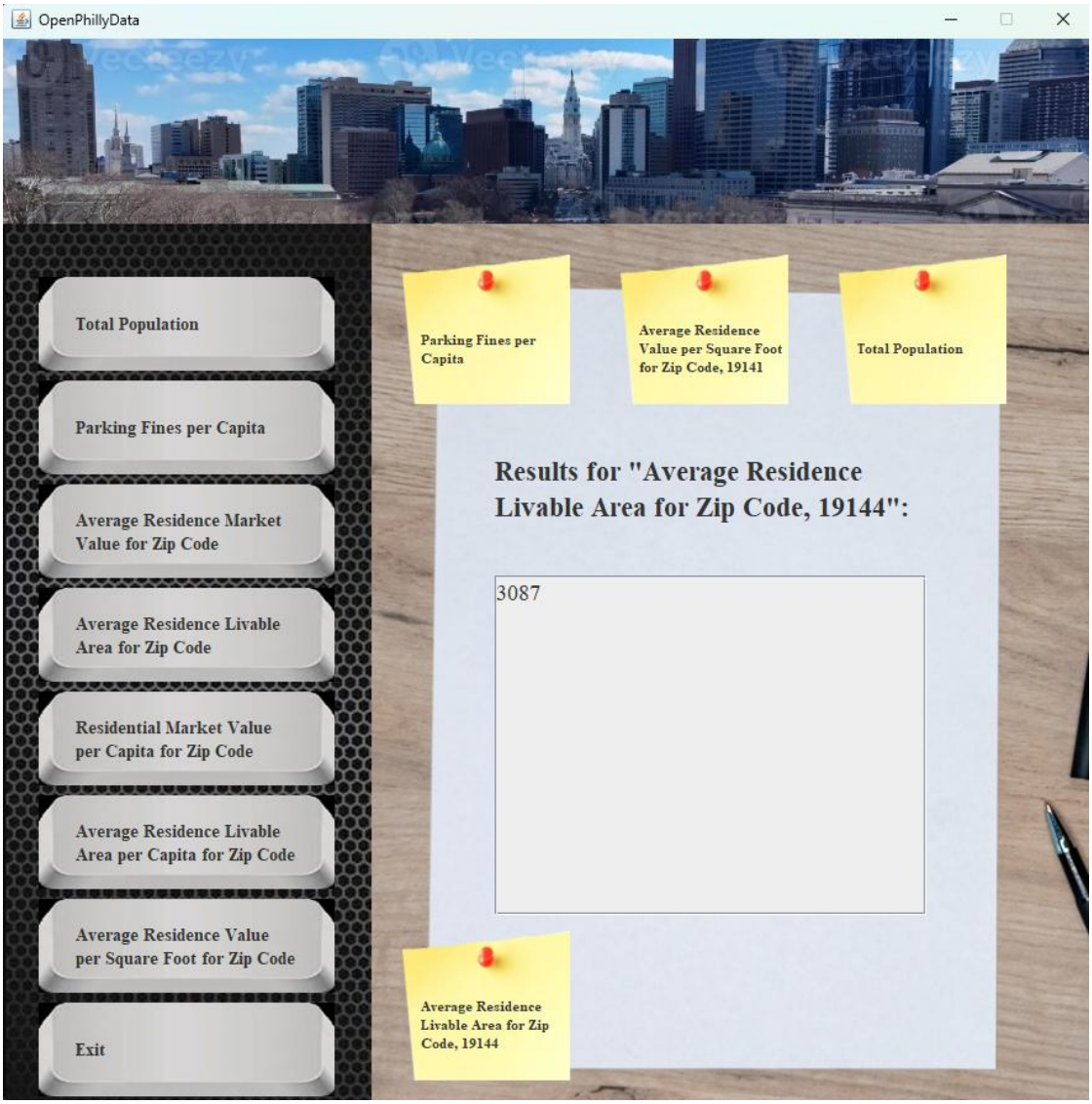
Getting user input for a search requiring input (pressing the “Average Residence Livable Area for Zip Code” button, for example):



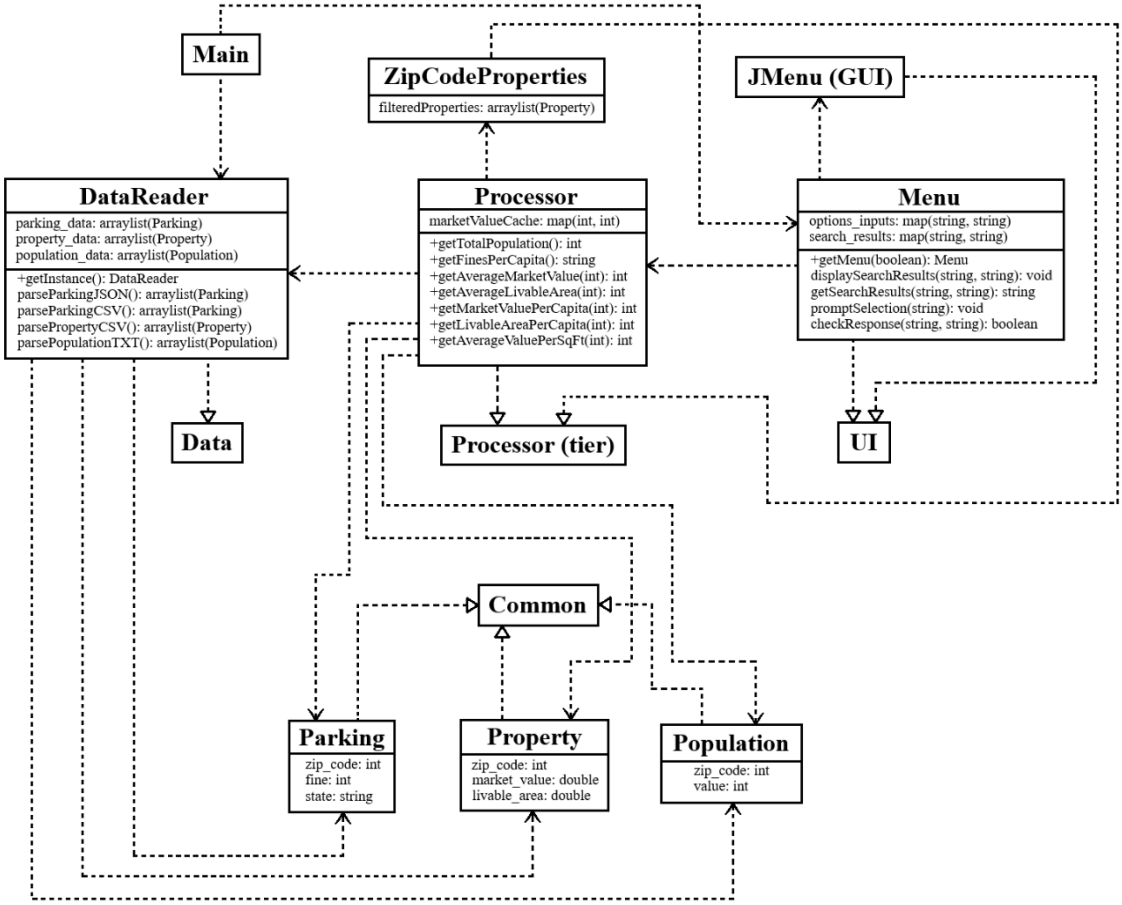
Returning the pop-up window after confirmed invalid user input (for provided input such as “hi” or “3005”, for example):



Successfully performing a search requiring user input (see the previous two examples):



UML Diagram:


Main
ZipCodeProperties
filteredProperties: arraylist(Property)
JMenu (GUI)
DataReader
parking_data: arraylist(Parking)
property_data: arraylist(Property)
population_data: arraylist(Population)
+getInstance(): DataReader
parseParkingJSON(): arraylist(Parking)
parseParkingCSV(): arraylist(Parking)
parsePropertyCSV(): arraylist(Property)
parsePopulationTXT(): arraylist(Population)
Processor
marketValueCache: map(int, int)
+getTotalPopulation(): int
+getFinesPerCapita(): string
+getAverageMarketValue(int): int
+getAverageLivableArea(int): int
+getMarketValuePerCapita(int): int
+getLivableAreaPerCapita(int): int
+getAveragePerSqFt(int): int
Menu
options_inputs: map(string, string)
search_results: map(string, string)
+getMenu(boolean): Menu
displaySearchResults(string, string): void
getSearchResults(string, string): string
promptSelection(string): void
checkResponse(string, string): boolean
Data
Processor (tier)
UI
Common
Parking
zip_code: int
fine: int
state: string
Property
zip_code: int
market_value: double
livable_area: double
Population
zip_code: int
value: int