Vanier College
Faculty of Science & Technology
Computer Science Technology

420-203-RE Program Development in a Graphical Environment

Teacher: Sleiman Rabah

Assignment I — Fall 2024

Due date: scheduled on LÉA

Revision History

Revision	Date	Author(s)	Description
1.0	Sept. 10, 2024	S.R.	Initial handout.

Contents

1 Learning Objectives

Getting started with implementing GUI applications with *JavaFX*, event-driven programming model, using existing Java library, working with comma-separated values (CSV), creating custom JavaFX stage, alert dialog, data logging, computing distance between two geographical locations.

2 Instructions and Constraints

- This assignment must be done **individually**.
- You must write your own code. You must submit a reference list that includes all sources (e.g., online code, documentation, etc.) used in your assignment.
- Using ChatGPT or similar tools is not permitted.
- Sharing code with, writing code for, and looking at code written by another student is not permitted. If you do so, this will be considered plagiarism.
- If you have any questions or need further clarification, feel free to reach out.

Note the following:
Using libraries or an existing implementation in this assignment is **not permitted**. You are required to implement your own Java classes as instructed in this document.

3 Required Software and Tools

You must use Apache NetBeans or IntelliJ IDE, Open JavaFX 22, and OpenCSV 5.9.

4 Problem Statement

In this assignment, you will develop a *JavaFX* application that calculates the distance between two geographical locations using the Haversine formula. The Haversine formula is a well-known method for computing the great-circle distance between two points on the Earth's surface, which is essential in applications like mapping, navigation, and geographic data analysis.

The application must offer the following major features: 1) computing the distance between a pair of Canadian postal codes, and 2) finding nearby locations within a radius of 100KM.

The Haversine formula [1] must be used to determine the great-circle distance between two points on a sphere given their longitudes and latitudes.

A *CSV* file (called *postalcodes.csv*) containing information about all the Canadian postal codes is provided. The content of this file must be parsed by your application at runtime to retrieve the information about the postal codes including the pairs of longitude and latitude (see Section 7 for more details).

5 Part I: Computing the Distance Between Two Canadian Postal Codes

In this part, you must design and implement a user interface that prompts the user to enter information about a pair of locations: the source (point A) and the destination (point B). Specifically, the user should enter two sets of three characters, each representing the first three characters of a **valid** Canadian postal code.

Scenario: Upon clicking the *compute distance* button, your application should open a *dialog box* that prompts the user to enter two sets of three characters representing Canadian area code identifiers. If the entered sets are ***valid***, the distance should be computed and displayed in the main window of the application. The user must be notified if any of the entered sets are ***invalid***.

5.1 Part I: Requirements

As a first step, you must implement the classes that are depicted in (see Figure 1).

When the user submits the form in the dialog box, two instances of the *PostalCode* class, which represent information about a Canadian postal code, must be passed from the dialog box back to the *main window*.

Upon receiving the PostalCode instances that contain information about the pair of locations, your application must use the **PostalCodeController class** to compute the distance and display the result as instructed

6 Part II: Computing Nearby Locations Within a Given Radius

In this part of the assignment, you must implement a feature that allows the user to find nearby locations within a given radius (e.g., 100 km) based on an initial Canadian postal code that must be entered in advance.

Scenario: Using a dialog box, your application should prompt the user to enter: 1) the first **three characters** of a **valid** Canadian postal code, and 2) the desired radius. Upon clicking the *find locations* button, the entered postal code and radius must be used to find nearby locations within the specified radius. If any matched locations are found, your application should display them to the user.
Since there could be zero or more matched locations, your app must display information about all the matched locations using a TableView and log messages (i.e., using System.out statements).

7 Implementation Details and Requirements

As a first step, you need to use the provided JavaFX template that was posted on LÉA. Once the template is properly configured, you need to implement and test the required Java classes as detailed below.

7.1 Classes to Be Designed and Implemented

Note: Before you implement a GUI for your application, the following part of the assignment must be implemented first.

You are required to design and implement the classes that are depicted in Figure 1.

You must use the Opencsv [2] library to parse the *postalcodes.csv* file. The lines in this file can be read into Java Beans. Therefore, the PostalCode class can be implemented as a Java Bean (refer to the Opencsv documentation, Section "Reading into Beans," for more details). However, the limitation of this approach is that the lines will be loaded into a list of PostalCode objects rather than into a *HashMap* as instructed. Loading the information about the postal codes into a *HashMap* provides better runtime performance for lookup operations. You should consider using the latter approach.

7.1.1 Steps to Follow

1. Create a new Java package called *controllers*.
   In this new package, create the *PostalCodeController* class.
2. Create a new Java package called *models*.
   In this new package, create the *PostalCode* class.
3. Create a new Java package called *tests*.
   In this new package, create the *Driver* class. The Driver class must contain a main method and must be used for testing the implementation of the PostalCodeController class.
4. Create a new Java package called *ui*.
   The *ui* package must be used to organize all the UI-related classes such as the *MainApp*, dialog box classes, etc.
5. Add OpenCSV as dependency to the *build.gradle* file. This step will be demonstrated in class.
6. Once the above-mentioned packages and classes created, implement the *PostalCode* and *PostalCodeController* classes that are depicted in the UML Class diagram below (see Figure 1).
7. Implement the test methods in the *Driver* class to test the following: **1)** parsing the .csv file (that is, reading its content and loading it into a *HashMap*); **2)** finding the distance between two postal codes; and **3)** finding all nearby locations. Include a description of the testing process and results.
8. Once implemented and tested, now you will be able to use the PostalCode and PostalCodeController classes in the UIs of your *JavaFX* app.
9. Design and implement the required user interfaces and dialog boxes as detailed in Section 5 and Section 6. At this stage, you are also required to implement the logic for rendering the results to the user.
10. Your implementation should include data cleansing and validation when appropriate.
11. You are allowed to implement additional Java classes if you judge necessary.
12. Handle any potential runtime exceptions. Check how your application handles edge cases such as invalid inputs.
13. Document your code using JavaDoc [3] . Provide comments and explanations for your code, especially the Haversine function and the user interface.

7.1.2 Additional Resources

1. For more details about the **Haversine formula:**, refer to:
   https://en.wikipedia.org/wiki/Haversine_formula
2. Another complete reference on the Haversine formula:
   https://www.movable-type.co.uk/scripts/latlong.html
3. Finding distance between two points based on Latitude and Longitude using the Haversine formula: https://andrew.hedges.name/experiments/haversine/
4. Online service: finding the distance between tow Canadian postal codes:
   https://www.freemaptools.com/distance-between-canada-postcodes.htm
5. A store locator service that locates nearby stores based on a Canadian postal code:
   https://stores.bestbuy.ca/en-ca/search
6. Java packages

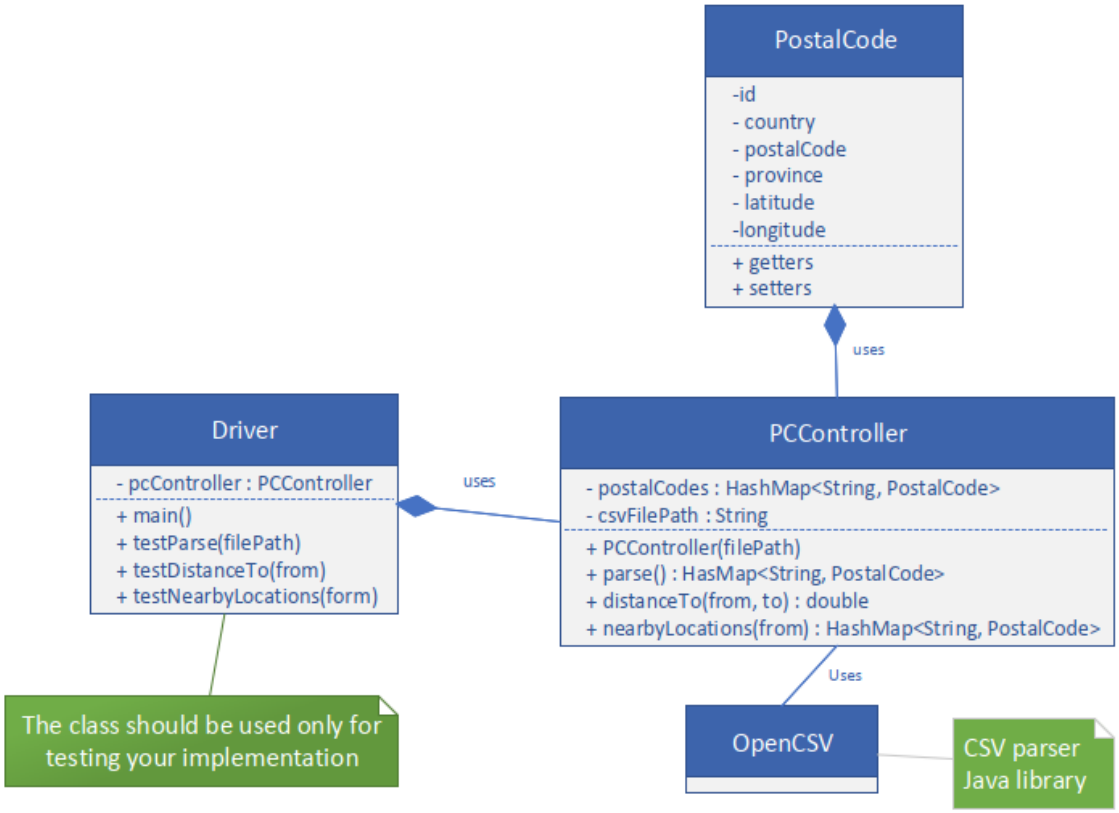

Figure 1: The classes that need to be implemented and used by your GUI

8 Evaluation Criteria

Your work will be evaluated based on the following criteria:

Criteria	Points
Correctness of the implementation: no errors, the program works well and meets all the specifications. Ability to perform as required and producing correct output.	5%
Relevance and accuracy of the source code documentation as instructed.	2%
Code readability, naming convention, and clarity. The use of meaningful self-explanatory names for identifiers (classes, variables, constants, methods, etc.).	3%
Compliance of the implementation with the stated requirements as well as simplicity and appropriateness of your implementation.	70%
The user-friendliness of the user interface(s) and presentation of the results	10%
Proper data validation and error handling	5%
Overall comprehension of the submitted source code.	5%
Total	**100%**

9 What to Submit

- ❑ Compress (zip) your NetBeans project and upload the compressed file to LÉA **before the due date**.

References

[1] Wikipedia, "Haversine formula." https://en.wikipedia.org/wiki/Haversine_formula. [Online; accessed 27-Aug-2024]. 2

[2] Opencsv, "Opencsv Users Guide." http://opencsv.sourceforge.net/. [Online; accessed 27-Aug-2024]. 3

[3] Oracle, "How to Write Doc Comments for the Javadoc Tool." https://www.oracle.com/ca-en/technical-resources/articles/java/javadoc-tool.html. [Online; accessed 27-Aug-2024]. 4