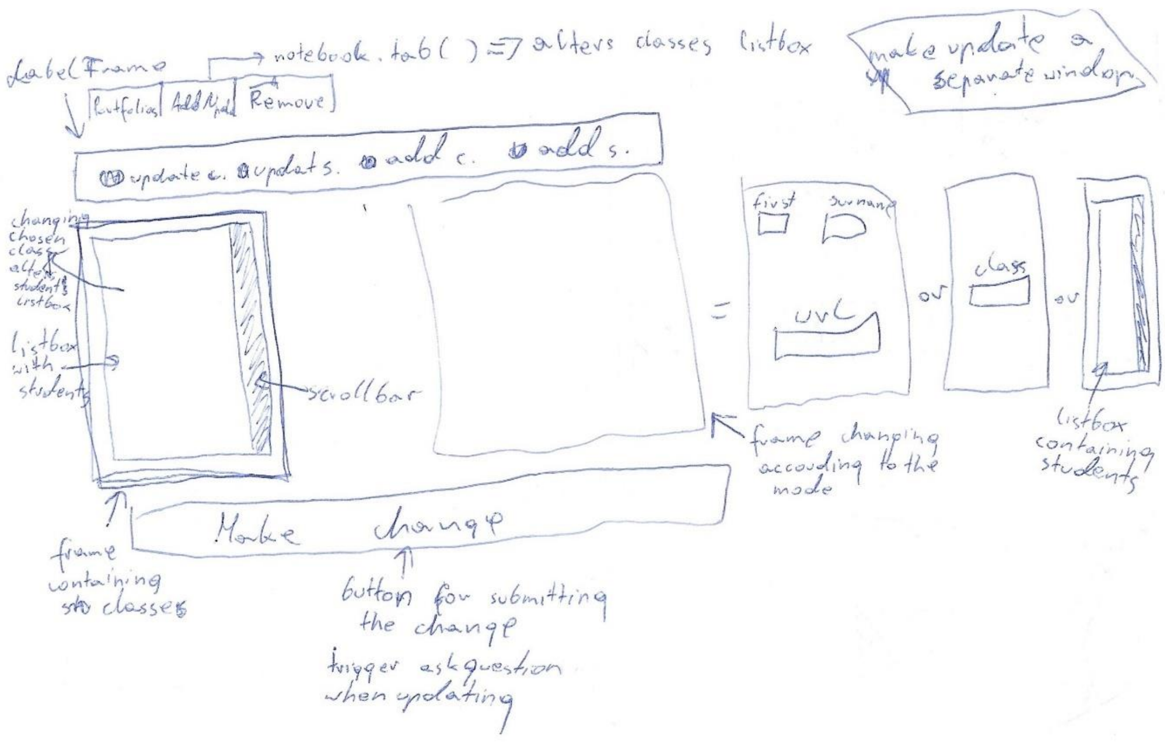
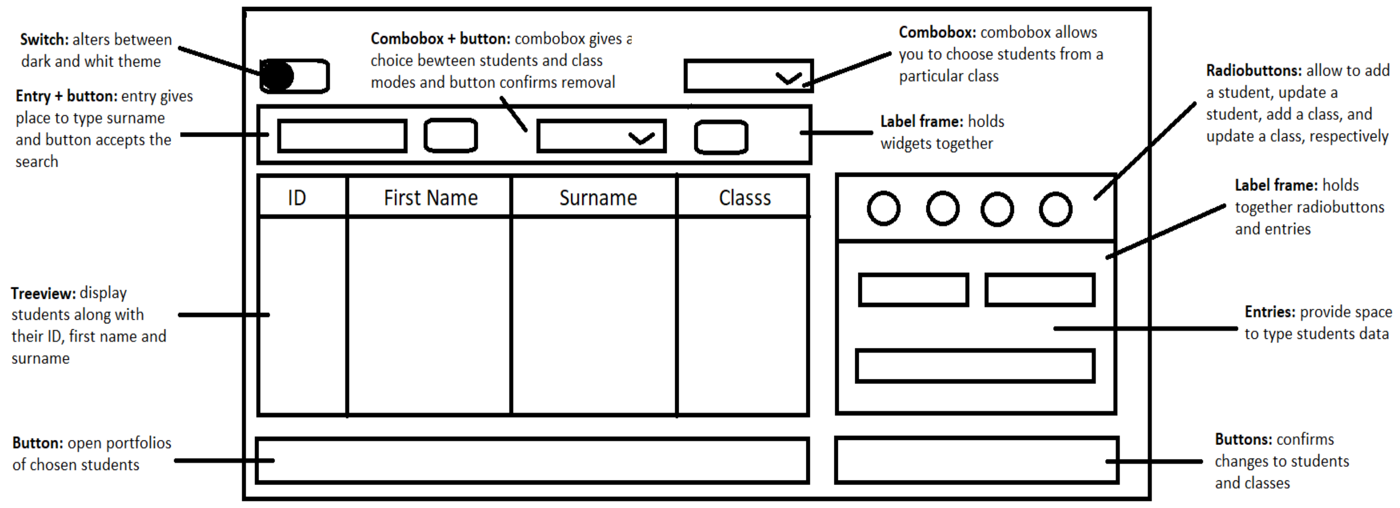
Criterion B: Design Overview

1. GUI development

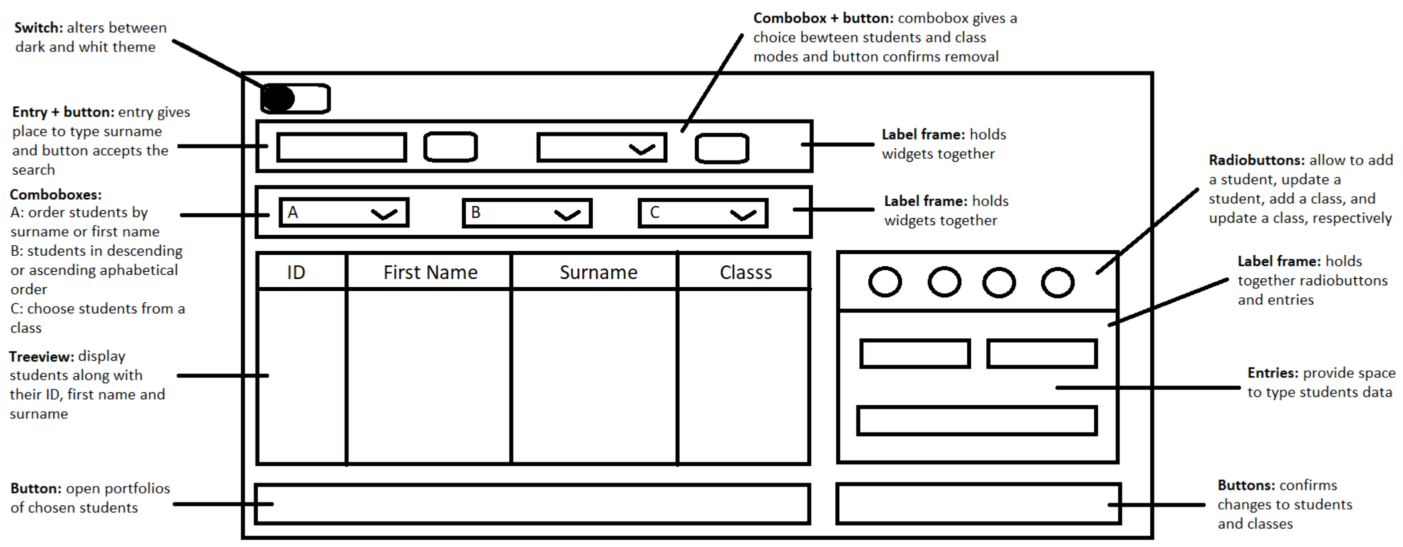
a) GUI deprecated version: not shown to the client


LabelFrame
notebook.tab( ) ⇒ alters classes listbox
make update a separate window
Portfolios
Add/Mod
Remove
update c.
updat s.
add c.
add s.
changing chosen class alters student's listbox
listbox with students
scrollbar
=
first
surname
URL
or
class
or
frame changing according to the mode
listbox containing students
Make change
frame containing classes
button for submitting the change
trigger askquestion when updating

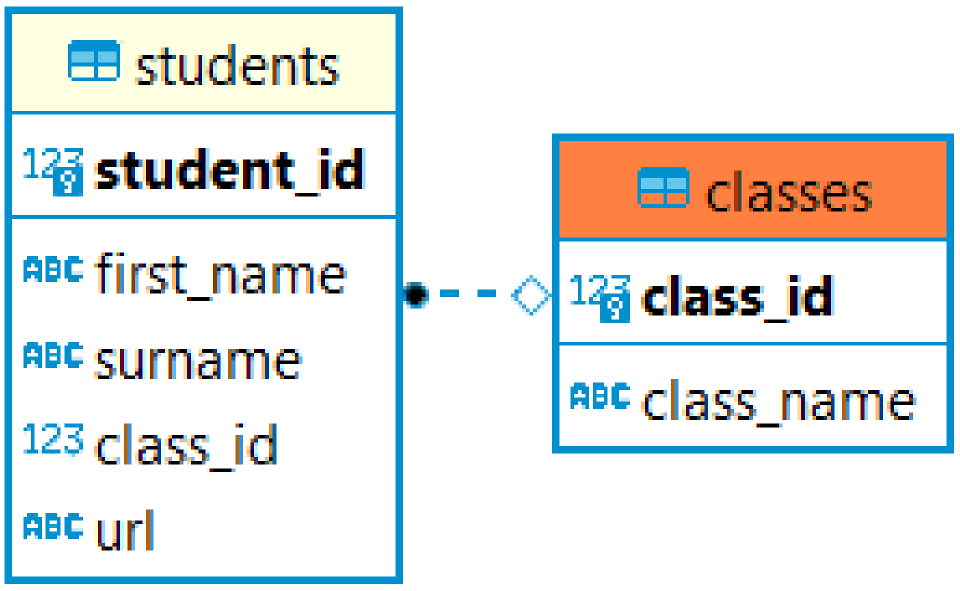
b) GUI version 1: Before client feedback (see Appendix A.2)


Switch: alters between dark and whit theme
Combobox + button: combobox gives a choice bewteen students and class modes and button confirms removal
Combobox: combobox allows you to choose students from a particular class
Entry + button: entry gives place to type surname and button accepts the search
Label frame: holds widgets together
Radiobuttons: allow to add a student, update a student, add a class, and update a class, respectively
ID
First Name
Surname
Classs
Label frame: holds together radiobuttons and entries
Treeview: display students along with their ID, first name and surname
Entries: provide space to type students data
Button: open portfolios of chosen students
Buttons: confirms changes to students and classes

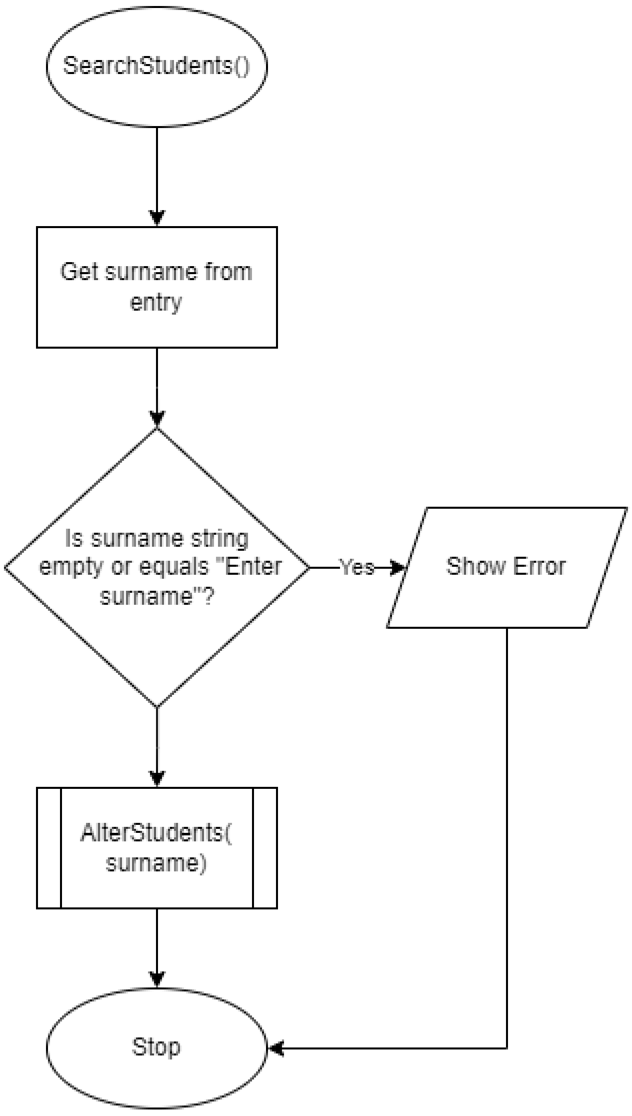
c) GUI version 2: After client feedback (see Appendix A.2)


Switch: alters between dark and whit theme
Combobox + button: combobox gives a choice bewteen students and class modes and button confirms removal
Entry + button: entry gives place to type surname and button accepts the search
Label frame: holds widgets together
Radiobuttons: allow to add a student, update a student, add a class, and update a class, respectively
Comboboxes: A: order students by surname or first name B: students in descending or ascending aphabetical order C: choose students from a class
A
B
C
Label frame: holds widgets together
ID
First Name
Surname
Classs
Label frame: holds together radiobuttons and entries
Entries: provide space to type students data
Treeview: display students along with their ID, first name and surname
Button: open portfolios of chosen students
Buttons: confirms changes to students and classes

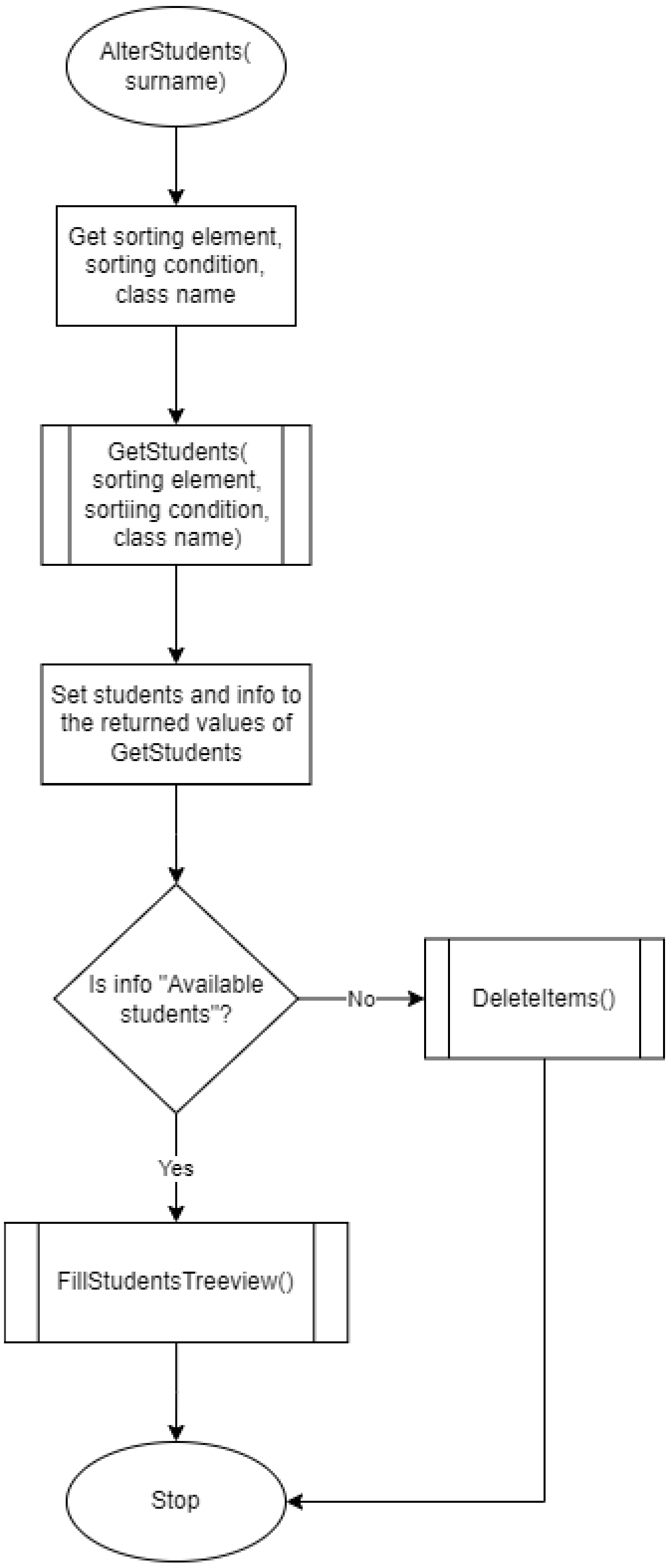
2. Database schema



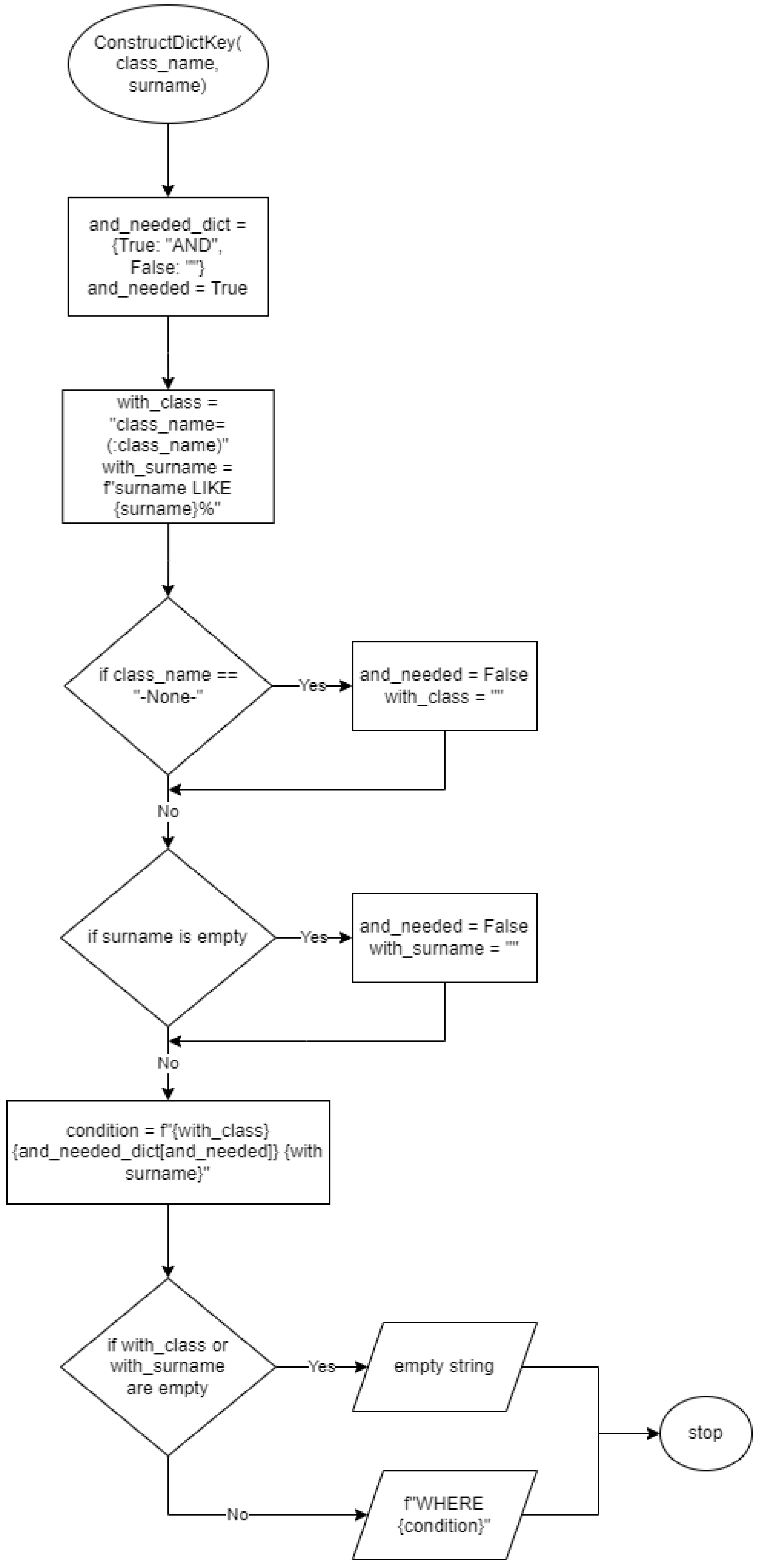
3. **Flowcharts outlining the retrieval of data from the database using functions**
   a) SearchStudent – initialiaze the search for a student



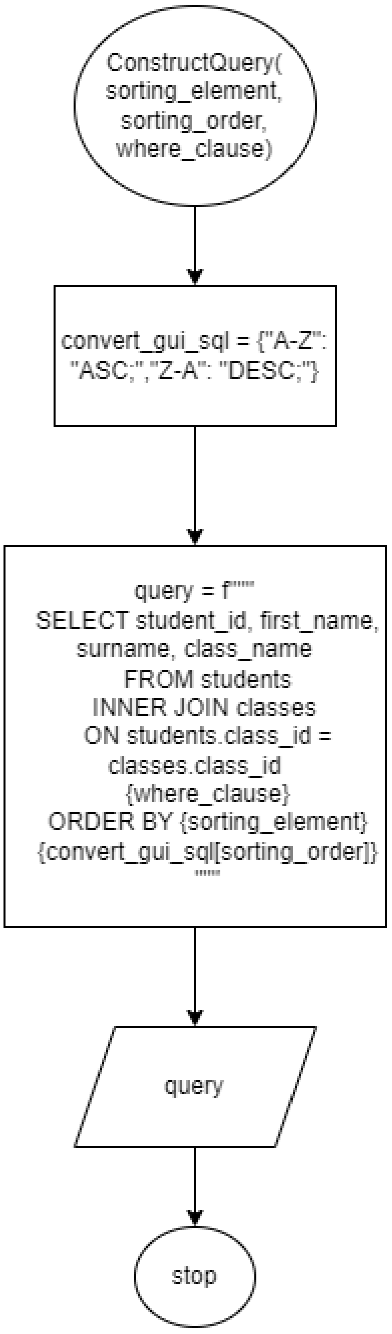
b) AlterStudents – manages what is shown on the students display


AlterStudents(surname)
Get sorting element, sorting condition, class name
GetStudents(sorting element, sortiing condition, class name)
Set students and info to the returned values of GetStudents
Is info "Available students"?
No
DeleteItems()
Yes
FillStudentsTreeview()
Stop

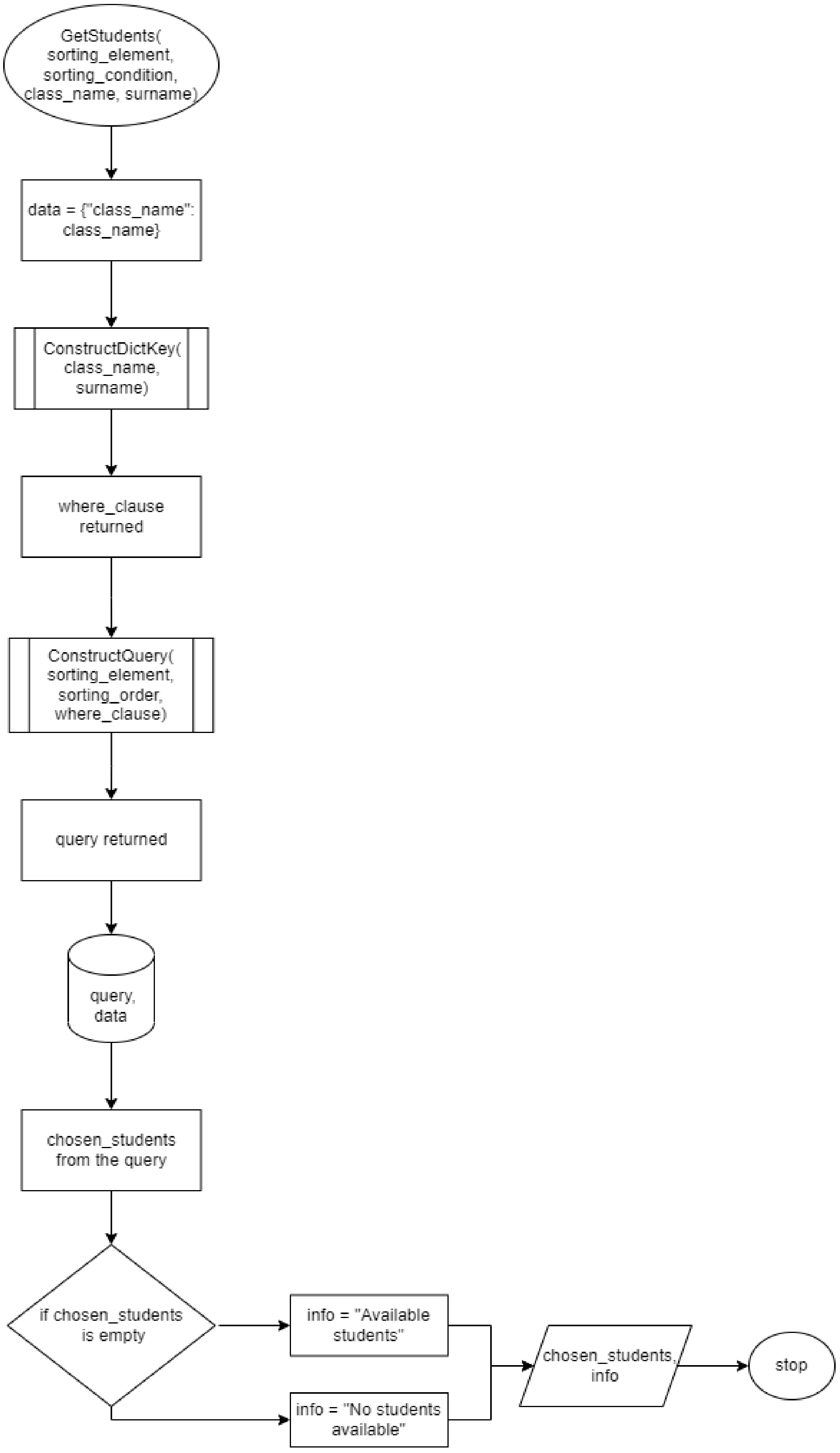
c) ConstructDictKey – adds necessary search conditions for the students look-up in the database


ConstructDictKey(class_name, surname)
and_needed_dict = {True: "AND", False: ""}
and_needed = True
with_class = "class_name= (:class_name)"
with_surname = f"surname LIKE {surname}%"
if class_name == "-None-"
Yes
and_needed = False
with_class = ""
No
if surname is empty
Yes
and_needed = False
with_surname = ""
No
condition = f"{with_class} {and_needed_dict[and_needed]} {with surname}"
if with_class or with_surname are empty
Yes
empty string
No
f"WHERE {condition}"
stop

d) ConstructQuery – creates a dynamic prompt for an SQL query



e) GetStudents – manages transaction against the database


GetStudents(sorting_element, sorting_condition, class_name, surname)
data = {"class_name": class_name}
ConstructDictKey(class_name, surname)
where_clause returned
ConstructQuery(sorting_element, sorting_order, where_clause)
query returned
query, data
chosen_students from the query
if chosen_students is empty
info = "Available students"
info = "No students available"
chosen_students, info
stop

4. Testing plan:

All tests will reside within the /tests folder, categorized into two main sections: "students alter" and "students display". The former focuses on testing changes to existing and new student records, while the latter handles testing for deletion and data retrieval. Within each of these folders, there will be separate sections for testing GUI elements and database elements.

Database element tests will be conducted using the 'pytest' package, which allows running specific sets of tests using "marks". I will use four marks:

- dbalteration: for testing code related to altering records such as adding classes
- validation: for testing data validation code
- constructing: for testing the code responsible for constructing dynamic SQL queries
- fetching: for testing code that retrieves data

The pytest configuration, including the marks, will be saved in "pytest.ini" for easy management of tests.

Test plan is summarized in the following table:

Action to be tested	**Test Method**
Modes for adding and updating classes and students display correctly	Run the frame with the four options as a separate GUI sub-app
Entries are correctly added to the database	Create multiple arguments for adding options and check database
Validation prevents adding incomplete entries	Use pytest to create incorrect entries and check for correct errors
Display settings, student view, and tools display correctly	Run frames for each part as separate GUI sub-apps
Entries fetched correctly in different variations (e.g., ordered by name)	Use pytest to create different data requests and check output
Database queries constructed correctly for required data retrieval	Use pytest to create data requests and check for expected queries

Word count: 247