Prof. Dr. Thomas Schultz
Olivier Morelle (morelle@uni-bonn.de)

B-IT
Bonn-Aachen International
Center for Information Technology
November 19, 2024

Winter term 2024/25

Computer Science for Life Scientists

Assignment Sheet 7

Solution has to be uploaded by November 27, 2024, 10:00 a.m., via eCampus

- This exercise can be submitted in **small groups** of 2-3 students. Submit each solution only once, but clearly indicate who contributed to it by forming a team in eCampus. Remember that all team members have to be able to explain all answers.
- Remember to include proper **documentation** in all your code, in particular, docstrings for functions.

If you have any questions concerning this exercise, please ask them on Monday or Wednesday, or use the forum on eCampus.

Exercise 1 (GUI Programming, *19 Points*)

In this exercise, you will learn how to develop GUI applications using Qt and its Python language bindings. To get you started, we provide a scaffold `list.py` for a small GUI application that shows a list of subjects with their personal data, and allows the user to interactively edit it. Our code already defines `ListWindow` as a class for the main window, containing a member `list` of type `QListWidget` to show the list of subjects. Buttons underneath should allow editing. The personal data (name, year of birth, gender and symptoms) is stored in the member variable `ListWindow.data`.

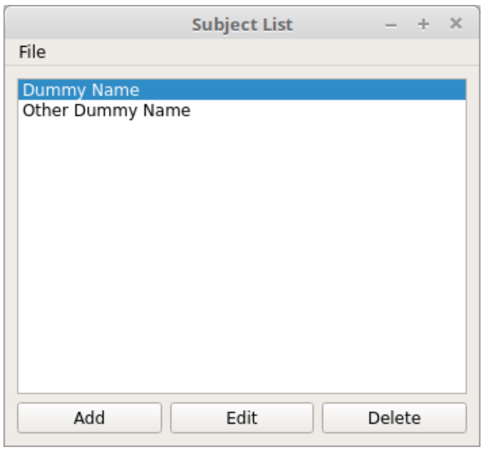

Figure 1: The main window with the list widget.

- Implement the function `updateList()` in the class `ListWindow`. It should populate `list` with the subject names currently stored in `data`. This requires the use of `QListWidget.addItem()` for each subject. Since `updateList()` might get called repeatedly, you have to `QListWidget.clear()` all previous entries from `list` first. (1P)
- When the user clicks on the button *Edit*, we want the function `onEditClicked()` to be called. In the Qt library you can set a function handling the `clicked` signal via `some_button.clicked.connect(some_function)`. (1P)

- Add a new button `Delete` and a new function `onDeleteClicked()` to `ListWindow`, that deletes the subject that is currently selected by the user in `list`. Don't forget that this change should be reflected both in `data` and in `list`. (2P)
- Now it is time to implement the `SubjectDialog`. It should present the detailed information of a single subject. For this we need to add GUI elements in the constructor `SubjectDialog.__init__()`. We want to be able to edit the subject's name (text), the year of birth (number), gender (multiple choice) and symptoms (again, text).

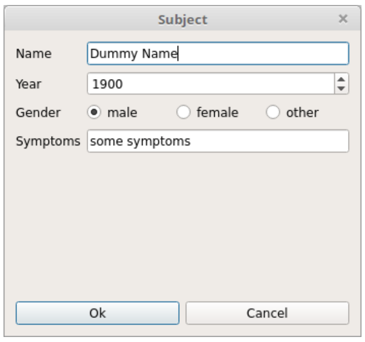

Figure 2: The dialog box for the subject data.

  - Create two new `QLineEdits` for the name and symptoms, respectively. Note that only creating them won't make them visible on the dialog box, we will take care of that later with the help of layouts. (1P)
  - Create a new `QSpinBox` for the year of birth. Set an appropriate range of allowed values with `setRange()`. (1P)
  - For the gender, create three new `QRadioButtons` (*'m', 'f'* and *'?'*). Since only one choice should be possible simultaneously, we also need to create a `QButtonGroup` and add the 3 radio buttons to it. (1P)
  - Create two buttons for *Ok* and *Cancel*. Qt already provides default handlers for both (`self.accept` and `self.reject`), but you still need to connect them to the buttons. (1P)
  - Create a suitable layout. Use a `QHBoxLayout` to arrange the two buttons in a horizontal row. `QFormLayout` allows to stack the other widgets vertically and automatically add labels next to them (`'Name'` etc.). Stack these two layouts vertically using a `QVBoxLayout` and apply it to the dialog via `self.setLayout()`. (2P)
    Hint: layouts in Qt have separate functions `addWidget()` and `addLayout()` to add child GUI elements directly or another child layout.
  - To get our subject's data into the dialog, pass it as a parameter to the initializer, and set the values in the corresponding widgets (for example, using `QLineEdit.setText()` and `QRadioButton.setChecked()`). (1P)
  - To read out the modified data, define a function `getData()` in `SubjectDialog`. It should retrieve values from the GUI elements (for example `QLineEdit.text()`), collect the data in a list, and return it. (1P)
- We can now also add a function in `ListWindow` to handle clicking the *Add* button. Similarly to `onEditClicked()` this new function should also show the `SubjectDialog` but add the result to `data` instead of replacing an existing element. (1P)
- Add two new actions *Load* and *Save* to the File menu that let the user choose a file (via `QFileDialog.getOpenFileName()` and `QFileDialog.getSaveFileName()`) and load/store `data` to the chosen file. (2P)
  Hint: to store the subject list into a file, you can use any method you like, including the pandas library or Python's pickle mechanism.

- Add a drag and drop functionality for re-ordering the list.
  - To enable drag and drop, use the `setDragDropMode` method from `QAbstractItemView`, and set it to `InternalMove` behavior. (1P)
  - Now, make sure that changes in the ordering are correctly reflected in `data`. For this, implement a new method `onRowsMoved()` that gets connected to the signal `rowsMoved` that is emitted by `list.model()` and re-orders `data` accordingly (e.g., via list slicing operations). (3P)

    Hint: The `rowsMoved` signal provides the parameters `(parent, start, end, destination, row)`, which contain the relevant information as follows: The (zero-based) row indices of the items that have been moved range from `start` to `end` (inclusive). The index of the row in front of which they have been inserted is given as `row`.

Exercise 2 (Function Classes, *6 Points*)

Fill out the following table. Use one of the symbols from $\{O, o, \Omega, \omega, \Theta, -\}$ to express the relationship between a function f from a row with a function g from a column. Always make the strongest possible statement: for $f(n) = n^3$ and $g(n) = n^4$, we have $f = o(g)$. The resulting entry should be "o" and not just "O" (make sure we can unambiguously see the difference, by spelling out "little-o" vs. "big-O" if needed). Use "$-$" if there is no relationship between functions.

Here $s(n)$ is defined as the function $s(n) = \begin{cases} 1 & \text{if } n \text{ is odd} \\ n & \text{if } n \text{ is even} \end{cases}$

	$\log_2(n)$	$s(n)$	5	2^n	$1/n$	n	e^n
$\log_2(n)$							
$s(n)$							
5							
2^n							
$1/n$							
n							
e^n							