FASTA-OMSSA Protein Peptide Visualiser - Technical Document

Introduction

This is a program for reading in a FASTA file containing a single protein sequence and a OMSSA CSV file containing peptides that belong to the protein. The program displays: a visual representation of the protein with its associated OMSSA peptides, the protein sequence with peptides highlighted, and a table of the peptides.

Program structure and design

The program consists of three public classes: `MyFrame`, `FASTAMethods`, and `OMSSAMethods`. It executes from the main method within the `MyFrame` class.

`MyFrame`

`MyFrame` is the main class of the program. It constructs the GUI `JFrame`, and all of the components within. It contains Action Events for the menu items and button in the GUI. The first Action Event is carried out when the user clicks the 'Open FASTA File' menu item. It responds by creating an instance of the `FASTAMethods` class, which calls upon the `getFile` method, using `jTextArea1` as the argument, to parse the FASTA file and display the protein sequence in the `jTextArea1` (see `FASTAMethods` section below). Also, the Protein Viewer's `jLayeredPane1` is set up for protein visualisation (background set, visibility set to true, any components previously added removed), any previous table displayed in the Peptide Viewer is removed, and the 'Hint' section is updated.
The second Action Event, when the user clicks the 'Open OMSSA File' menu item, responds by creating an instance of the `OMSSAMethods` class (see `OMSSAMethods` section below). If the program has already read in a FASTA file, it proceeds to carry out the three `OMMSAMethods` methods: `getCSV`, `highlightPeptides`, and `viewPeptides`. `getCSV` uses `jTable1` as an argument to parse the CSV file and display the filtered table in the Peptide Viewer. `highlightPeptides` uses `jTextArea1` as an argument to highlight the peptides matching the protein sequence in the Sequence Viewer. `viewPeptides` uses `jLayeredPane1` and `jTextArea1` as arguments to add the peptides to the Protein Viewer visualisation of the protein. If there is either no FASTA file or one that does not contain a UniProt ID from the CSV file a dialogue box appears notifying the user that they must first input the correct FASTA file. If the table has been produced successfully (i.e. if the CSV file has been read in successfully) the message in the Hint section is updated to describe what is being shown in the window.
The last Action Event, when the user clicks 'Quit', closes the program.

`FASTAMethods`

`FASTAMethods` is a class that extends the `MyFrame` class. It has one method, `getFile`, which uses the `jTextArea1` as an argument and opens a `FileDialog` box in front of the frame, allowing the

user to navigate to and open the FASTA file. In order to minimise the chances of a user opening the wrong file type, the method only allows opening of files with file extensions containing ".fa". If a file with an extension containing this is selected, a `BufferedReader` instance is created to parse the text. The first line, containing the unique description, is parsed for the UniProt ID, which is stored in a global variable accessible by any of the classes. Then, a document is created in the `jTextArea1`, into which the remaining lines of the FASTA file are inserted. If a file that does not contain ".fa" in its extension is chosen, a message dialogue appears advising the user that it is not a FASTA file.

`OMSSAMethods`

`OMSSAMethods` is a class that extends the `MyFrame` class. It has three methods (`getCSV`, `highlightPeptides`, `viewPeptides`) and a subclass (`MyHighlightPainter`). The class has three global String variables that are used in more than one method: `peptide` (for storing the OMSSA peptides), `pepStart` (for storing peptide start positions), and `pepStop` (for storing peptide stop positions). The first method, `getCSV`, uses `JTable` as a parameter to which the parsed CSV file is displayed. It is called upon in `MyFrame` only if the `uniProtID` variable is initialised, otherwise a message dialogue appears advising that first a FASTA file must be uploaded. Once called upon succesfully, a `FileDialog` box is displayed from which the user chooses the OMSSA CSV file. The program tries to parse 5 columns from rows in which the Defline column contains the UniProt ID, using the openCSV package's CSVReader class (http://opencsv.sourceforge.net/). If the table contains 0 rows (i.e. a CSV file without the UniProt ID was chosen) a message dialogue appears advising the user that the CSV file did not contain the UniProt ID and to choose one that does. If the CSV file does contain corresponding peptides, new rows with 5 columns are added to `jTable1`.
The subclass, `MyHighlightPainter`, extends the `DefaultHighlighterPainter` and is used in the `highlightPeptides` method. This method uses the peptide sequences saved to the global `peptide` String variable to highlight the sequences in the Sequence Viewer. First, `jTextArea1`'s `Highlighter` is fetched and stored in a new interface. Three new `HighlightPainter` renderers are created for the start, stop and rest of the peptide sequence. Then, one for loop is used to highlight the start and stop positions of the peptides, before another for loop highlights the remaining amino acids of the peptide sequences. It is structured in this way so that the start and stop positions of peptide sequences that overlap one another can still be viewed in the Sequence Viewer.
The final method called upon from `MyFrame` is `viewPeptides`. It creates panes within `jLayeredPane1` to produce a visualisation of the OMSSA peptides along the protein sequence. It requires two arguments, `jLayeredPane1` and the `jTextArea1`. A for loop is used to cycle through the start and stop positions of the peptides and adjust them to be relative to the width of the `jLayeredPane1` and length of the protein sequence. Another for loop is used to create and add a `JPanel` for each peptide onto `jLayeredPane1`, producing the Protein Viewer.

Word count: 962.

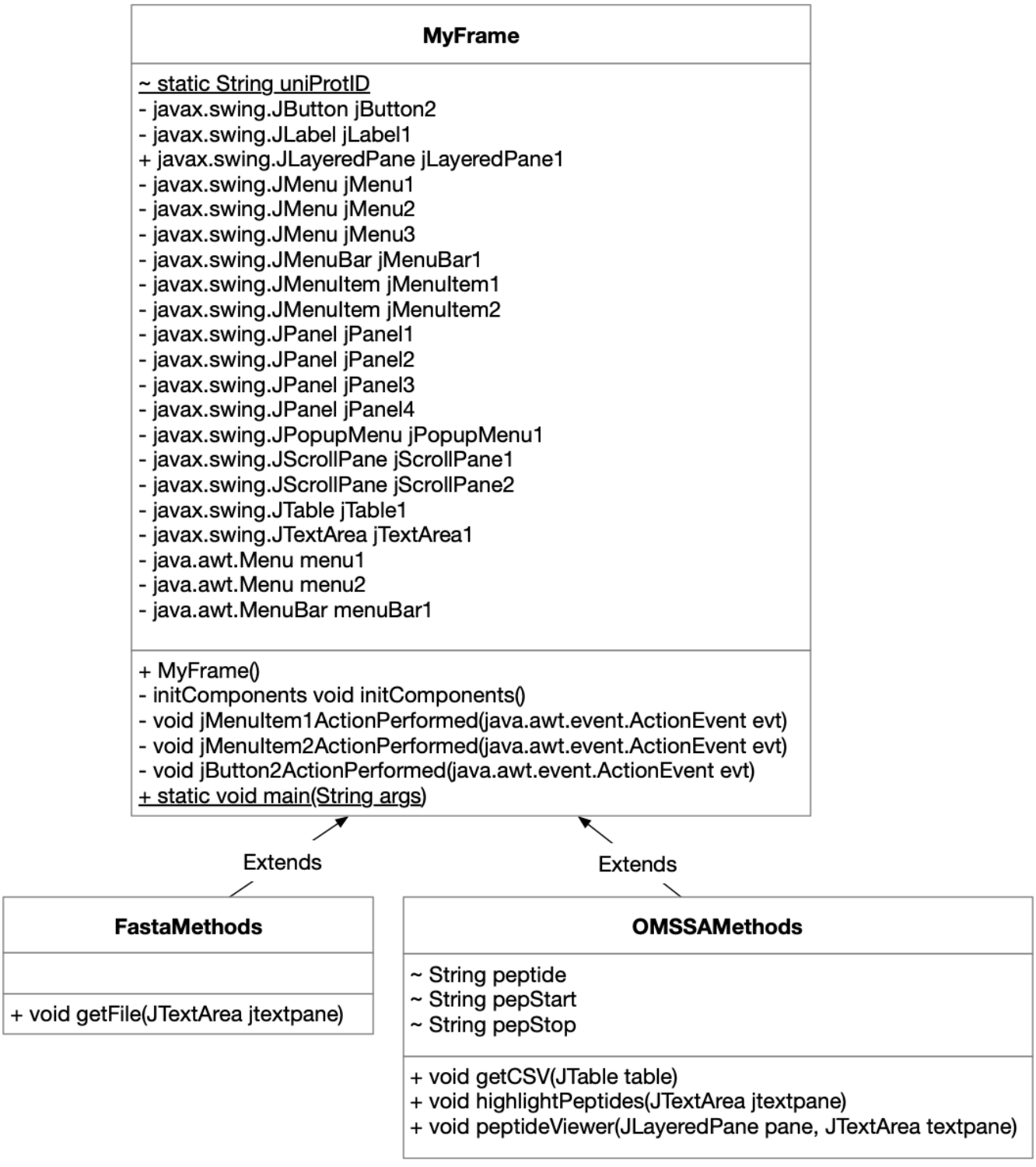

Figure 1: **UML class diagram**

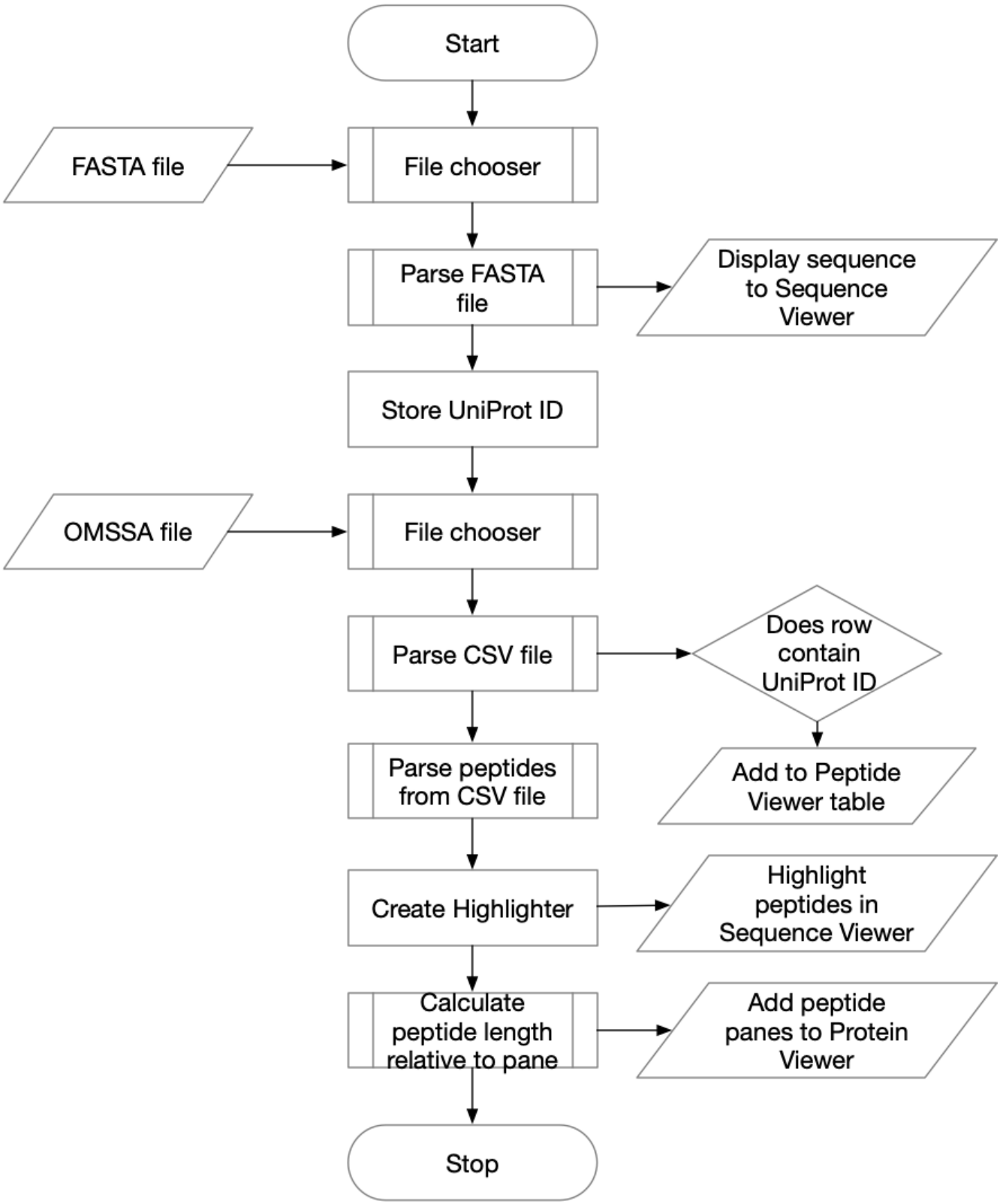

Figure 2: **Program flowchart**