DOKUMENTATION TEAMPROJEKT 2018

NANTIA LEONIDOU

LINA GÖNNHEIMER

LISA ADAMS

INHALTSVERZEICHNIS

Main-Klasse

1. Definition von Model, View, ...
2. mzTab file parser
3. Filter
4. Checkboxen
5. Überschriften
6. Zeige nur gewünschte Spalten an
7. Maximumsbestimmung

BarDelegate-Klasse

1. paint

bardelegatepep-Klasse

1. paint

Peptidtabelle-Klasse

1. handleButton
2. slotSelectionChanged

Proteintabelle-Klasse

Unser vorliegendes Programm dient zur Darstellung von bei der Massenspektrometrie gemessenen Proteinen. Dabei werden zwei Tabellen angezeigt, wobei die obere alle gefunden Proteine anzeigt, und die untere die zugehörigen Peptide. Dabei werden Zahlenwerte über Balken visualisiert. Zudem können durch Auswahl einer Proteinzeile alle zugehörigen Peptide angezeigt werden. Des Weiteren besitzen beide Tabellen auch eine Suchfunktion.

Main-Klasse

1. Definition von Model, View, ...

In der Mainklasse wird zuerst ein QFileDialog implementiert, um ein Fenster zu öffnen, in dem der Benutzer die gewünschte anzuzeigende Datei auswählen kann.

QFile file(QFileDialog::getOpenFileName());

Anschließend werden QSplitter zur Darstellung der Widgets initialisiert: ein Gesamtsplitter splitter, der die beiden Tabellen, die zwei Filterbereiche und die Buttons zum Anzeigen aller Proteine und zum Resetten aller Filter enthält.
Der Splitter filterareaprotein bekommt später zwei QLineEdits zum Filtern der Accession- und Descriptionspalte der Proteintabelle und dazugehörige Reset Buttons; der Splitter filterareapeptides wird die QLineEdits zum Filtern der Accession- und Sequencespalte der Peptidtabelle bekommen.
Dann erhalten die Tabellen jeweils noch Delegates zum Zeichnen der Balken (vgl. BarDelegate-Klasse bzw. bardelegatepep-Klasse).

2. mzTab file parser

Im zweiten Teil wird nun der Parser für mzTab-Dateien implementiert. Dabei definieren wir uns zuerst Indexvariablen für die Spalten der Tabelle, die wir anzeigen möchten. Das sind hier für die Proteintabelle:

Spaltenname	**Name in mzTab Datei**
Protein Coverage	protein_coverage
# Peptides	num_peptides_distinct_ms_run[1]
# Spectra	num_psms_ms_run[1]
MS2Quant	protein_abundance_assay[1]
Confidence	best_search_engine_score[1]
Accession	accession
Description	description

Tabelle

Für die Peptidtabelle sind das:

Spaltenname	**Name in mzTab Datei**
Sequence	sequence
Confidence	search_engine_score[1]
Start	start
Accession	accession

Tabelle

Wir lesen die im QFileDialog ausgewählte Datei Zeile für Zeile ein, und suchen jeweils nach den Schlüsselwörtern für die jeweiligen Zeilen. Dabei suchen wir in den Protein- bzw. Peptidheaderzeilen nach den richtigen Indizes für die Spalten, die wir anzeigen wollen, und lesen in den Zeilen, in denen die Protein- und Peptiddaten stehen die Dateien ein und übergeben sie dem zur Tabelle gehörigen Model.

3. Filter

Zuerst wird sortingenabled für die Tabelle auf true gesetzt, damit das auf- und absteigende Sortieren durch Klick auf den Header möglich ist.

Dann wird der Filter implementiert. Dies geschieht mithilfe von QSortFilterProxyModels. Dabei initialisieren wir je ein ProxyModel und ein QLineEdit pro Spalte, die wir filtern wollen. Jedes QLineEdit wird dann mittels einer Signal Slot Connection mit einem ProxyModel verbunden. Dabei verschachteln wir die ProxyModels für jede Tabelle, indem wir sie der Reihe nach als SourceModel der anderen ProxyModels setzen, während das erste ProxyModel unser zu Beginn erstelltes QStandardItemModel als SourceModel bekommt.

Bsp.: proxyModelSequencePep->setSourceModel(proxyModelAccessionPep);

Das letzte ProxyModel wird dann als Model der Tabelle gesetzt.
Über setFilterColumn wird für jedes ProxyModel außerdem definiert, welche Spalte es filtern soll.
Dann wird jedes ProxyModel und QLineEdit noch über eine Signal Slot Connection mit dem resetfilter-Button verbunden, der alle Filter resettet. Zusätzlich erstellen wir für jeden Filter nun noch einen eigenen Reset-Button:

Bsp.:

```
QPushButton *resetFilterSequencePep = new QPushButton(filterareapeptide);
resetFilterSequencePep->setText("Reset Sequence Filter");
QObject::connect(resetFilterSequencePep, SIGNAL (clicked()),
lineEditSequencePep, SLOT(clear()));
QObject::connect(lineEditSequencePep, SIGNAL(textChanged(QString)),
proxyModelSequencePep, SLOT(setFilterFixedString(QString)));
```

Bei allen Filtern der Peptidtabelle wird zusätzlich noch die Selection, die durch die Auswahl eines Proteins zustande kam, aufgehoben. Auch dies geschieht wieder durch eine Signal Slot Connection.

4. Checkboxen

Anschließend werden die Checkboxen hinzugefügt, indem wir ein QStandardItem erstellen, das wir auf checkable setzen. Diese wird dann sowohl in der Protein- als auch der Peptidtabelle in jeder Reihe in die „Checkboxcolumn“ hinzugefügt, die hier als letzte Spalte definiert wurde.

5. Überschriften

Im nächsten Teil benennen wir die Spalten anhand unserer Spaltenindexvariablen (vgl. Tabelle 1 und Tabelle 2). Dies tun wir über setHorizontalHeaderItem.

6. Zeige nur gewünschte Spalten an

Im Anschluss daran gehen wir nun alle Spalten durch, und zeigen im View nur die an, die wir auch darstellen wollten, indem wir die Indizes unserer Spaltenindexvariablen durchgehen (vgl. Tabelle 1 und Tabelle 2).

7. Maximumsbestimmung

Abschließend bestimmen wir noch für jede Spalte das Maximum, indem wir alle Reihen durchgehen. Das Maximum jeder Spalte merken wir uns im zugehörigen Header durch die Qt::AccessibleDescriptionRole; so wird die Zahl nicht dargestellt, aber trotzdem im Header gespeichert. Das Maximum brauchen wir zur Darstellung der Balken (vgl. Klasse BarDelegate und bardelegatepep).

8. Verbinden der Tabellen

Zum Schluss verbinden wir noch die beiden Tabellen, und den Button zur Anzeige aller Peptide mit der Peptidtabelle, jeweils über eine Signal Slot Connection. Bei den Tabellen wird dabei das SelectionModel der Proteintabelle mit der Peptidtabelle verbunden (vgl. Klasse Peptidtabelle).

Dann wird der QSplitter angezeigt.

BarDelegate-Klasse

1. paint

Hier werden die Balken für die Proteintabelle gezeichnet. Dazu wird erst der Painter initialisiert. Dann werden die Daten des aktuellen Indexes eingelesen. Nun werden die Balken gezeichnet, wenn der aktuelle Index aus einer gewünschten Spalte stammt. Aktuell soll der Balken für folgende Spalten gezeichnet werden: PI, Confidence, #Peptides, MS2Quant, # Spectra, Protein Coverage.
Das Maximum der aktuellen Spalte wird aus dem Header geholt, der aktuelle Wert wird dann durch das Maximums der Spalte geteilt, um einen Wert kleiner gleich 1 zu erhalten, womit dann der Balken skaliert wird. In den Spalten #Peptides und #Spectra werden zusätzlich Teile der Balken in Gelb und Rot angezeigt. Dies wird dadurch erreicht, dass wir erst einen roten Balken zeichnen, darüber dann einen gelben und einen grünen Balken legen.
Die PI Spalte wird zudem von einem komplett grünen Balken ausgefüllt.

bardelegatepep-Klasse

1. paint

Die Funktion funktioniert analog zur Paint Funktion der Bardelegate-Klasse. Diese Klasse zeichnet zusätzlich für die Start-Column keinen kompletten Balken, sonder platziert ein Rechteck relativ zum Tabellenwert. Die Größe des Rechtecks wurde mit 10 festgelegt.

Peptidtabelle – Klasse

Eine Instanz dieser Klasse ist quasi ein QTableView mit zwei zusätzlichen Funktionen, die im Folgenden beschrieben werden.

1. handleButton

Dieser Slot zeigt alle Reihen der Peptidtabelle an.

2. slotSelectionChanged

Dieser Slot bekommt in der main-Klasse ausgewählte Indizes der Proteintabelle überreicht. Zuerst werden dann alle Reihen der Peptidtabelle nicht mehr angezeigt. Dann wird über den Header der Index der Accessionspalte der Peptidtabelle ermittelt. Anschließend werden nacheinander die Daten aus den ausgewählten Indizes der Proteintabelle mit den Daten in der Accessionspalte der Peptidtabelle verglichen. Finden wir dieselbe Accessionnummer, wird das Peptid angezeigt.

Proteintabelle – Klasse

Diese Klasse ist von der QTableViewKlasse abgeleitet, und hat noch keine weiteren Funktionen. Sie wurde erstellt, falls in Zukunft die Funktionen der Proteintabelle erweitert werden sollen.