G. Einsendeaufgabe

Objektorientierte Programmierung mit C++ und Qt

Code:
CPBS17E-XX1-N01

Name: Rein	Vorname: Wjatscheslaw
Postleitzahl und Ort: 14193 Berlin	Straße: Cunostr. 54A
Studien- bzw. Vertrags-Nr.: 800463563	Lehrgangs-Nr.: 246

Fernlehrer/in:
Datum:
Note:
Unterschrift Fernlehrer/in:

Bitte reichen Sie Ihre Lösungen über die Online-Lernplattform ein oder schicken Sie uns diese per Post. Geben Sie bitte immer den Code zum Studienheft an (siehe oben rechts).

Inhaltsverzeichnis

1 Aufgabe:

Aufgabe:

Erstellen Sie für den Dialog zum Bearbeiten der Datensätze aus diesem Studienheft eine Leiste, in der unten im Dialog die Nummer des aktuellen Datensatzes und die Anzahl aller Datensätze in der Datenbanktabelle angezeigt werden. Der Dialog mit der Leiste könnte ungefähr so aussehen:

Lassen Sie die Anzeige in der Leiste beim Navigieren und beim Löschen aktualisieren. Denken Sie beim Löschen daran, dass sich auch die Gesamtzahl verändert.

40 Pkt.

Lösung:

Als erstes wird eine Signal-Slot Verbindung hergestellt. Dieses Signal wird ausgegeben, nachdem sich der aktuelle Index geändert hat und alle Widgets mit neuen Daten belegt wurden. Empfänger ist ein Slot, der die Liste aktualisiert (Listing 1.1, Zeile 5).

Die Nummer des aktuellen Datensatzes und die Anzahl aller Datensätze in der Datenbanktabelle werden über Signal-Slot Verbindung aktualisiert und in einem Label angezeigt. Zusätzlich werden Zähler mit einem Korrekturwert (+1) versehen um eine „menschenfreundliche“ Anzeige zu realisieren (Listing 1.1, Zeile 10).

```
1         void einzelAnzeige::datenMappen()
2         {
3           ...
4           //Bitte in einer Zeile eingeben.
5           connect(mapper ,&QDataWidgetMapper::currentIndexChanged ,
6                   this,&einzelAnzeige::indexRefresh);
7           indexRefresh();
8         }
9         //
10        void einzelAnzeige::indexRefresh()
11        {
12          QString count, cntMax;
13          count = QString::number(mapper->currentIndex()+1);
14          cntMax = QString::number(modell->rowCount());
15          datenSatzLbl->setText("Datensatz "+count+" von "+cntMax);
16        }
17
```

Listing 1.1: Anweisungen bzw. Methoden für die Aufgabe 1

Die Abbildung (1.1) zeigt die Ergebnisse der ersten Aufgabe.

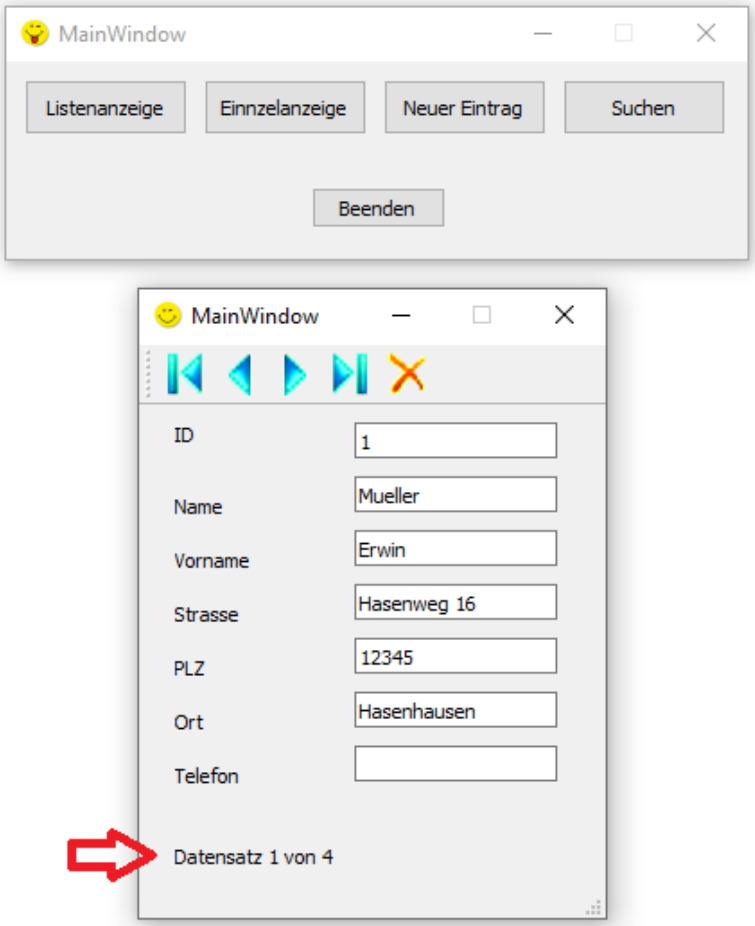

Abbildung 1.1: Ergebnis der Aufgabe 1

2 Aufgabe:

Aufgabe:

Erstellen Sie für die Datenbankanwendung eine Suchfunktion, über die gezielt nach Einträgen mit einem bestimmten Nachnamen gesucht werden kann. Die Suchfunktion soll über eine Schaltfläche aus dem Hauptfenster gestartet werden können. Danach soll zuerst ein Formular zur Eingabe des Suchkriteriums erscheinen. Die Ergebnisse der Suche sollen anschließend in einer Tabelle in einem weiteren Formular erscheinen. Wenn die Suche keine Treffer liefert, kann diese Tabelle auch leer bleiben.

Zwei Hinweise zur Lösung:

Einen Eingabedialog können Sie sehr einfach über die Klasse `QInputDialog` erstellen. Mehr zu dieser Klasse finden Sie in der Dokumentation. Wenn Sie die Klasse `QInputDialog` nicht einsetzen wollen, können Sie aber auch einen eigenen Dialog erstellen.

Für die Anzeige der Ergebnisse können Sie die Klasse `QTableView` benutzen.

60 Pkt.

Lösung:

Im Konstruktor wird eine Methode namens „suchFunktion“ deklariert, die ein Parameter vom Typ QString übernimmt (Listing 2.1).

```
1    class suchFunktion : public QDialog
2    {
3      public:
4      //der Konstruktor
5      //Aufgabe 2
6      //Deklaration von der Methode mit einem Parameter vom Typ QString.
7      suchFunktion(QString setFilter);
8    };
9
```

Listing 2.1: Konstruktor für die Aufgabe 2

Bei der Betätigung der Schaltfläche im Hauptfenster wird zuerst ein Formular zur Eingabe des Suchkriteriums erscheinen. Der Eingabedialog ist vereinfacht mit der Klasse QInputDialog realisiert. Der Rückgabewert vom getText() wird als QString Parameter an die Suchfunktion übergeben und stellt somit ein Suchkriterium dar. Zusätzlich wird eine leere Eingabe und „Abrechen“-Wunsch vom Benutzer detektiert um die unnötige Anzeige von „leeren“ Daten zu vermeiden (Listing 2.2, Vgl. Qt Dokumentation).

```
1   void MainWindow::on_buttonSuchen_clicked()
2   {
3     bool ok;
4     QString suchFilter;
5     suchFilter = QInputDialog::getText(this, tr("Suchfunktion"),
6             tr("Eingabe des Suchkriteriums:"), QLineEdit::Normal,
7             "Nachname", &ok,
8           windowFlags().setFlag(Qt::WindowContextHelpButtonHint, false));
9
10    if (ok && !suchFilter.isEmpty())
11    {
12      //wenn eine Verbindung zu Datenbank besteht
13      if (dbVerbunden == true)
14      {
15        //das Formular anzeigen
16        suchFunktion *formNeu = new suchFunktion(suchFilter);
17        formNeu->show();
18      }
19    }
20  }
21
```

Listing 2.2: Ein Slot für die Aufgabe 2

Die Ergebnisse der Suche werden in einer Tabelle in einem weiteren Formular gefiltert angezeigt (siehe Listing 2.3). Die tabellarische Aufbau wird von der listen-Anzeige Funktion übernommen um die einheitliche Darstellung zu erreichen. Die Abbildung (2.1) zeigt die Ergebnisse der zweiten Aufgabe.

```
1   suchFunktion::suchFunktion(QString setFilter)
2   {
3     //den Fenstertitel setzen
4     setWindowTitle("Listenanzeige nach Suchkriterium");
5     ...
6
7     //Aufgabe 2
8     //Funktion: Aufbau vom Filter (Suchparameter).
9     modell->setFilter("anachname = '" + setFilter + "'");
10
11    ...
12  }
13
```

Listing 2.3: Ergebnisse einer Suchfunktion in der Tabelle dargestellt

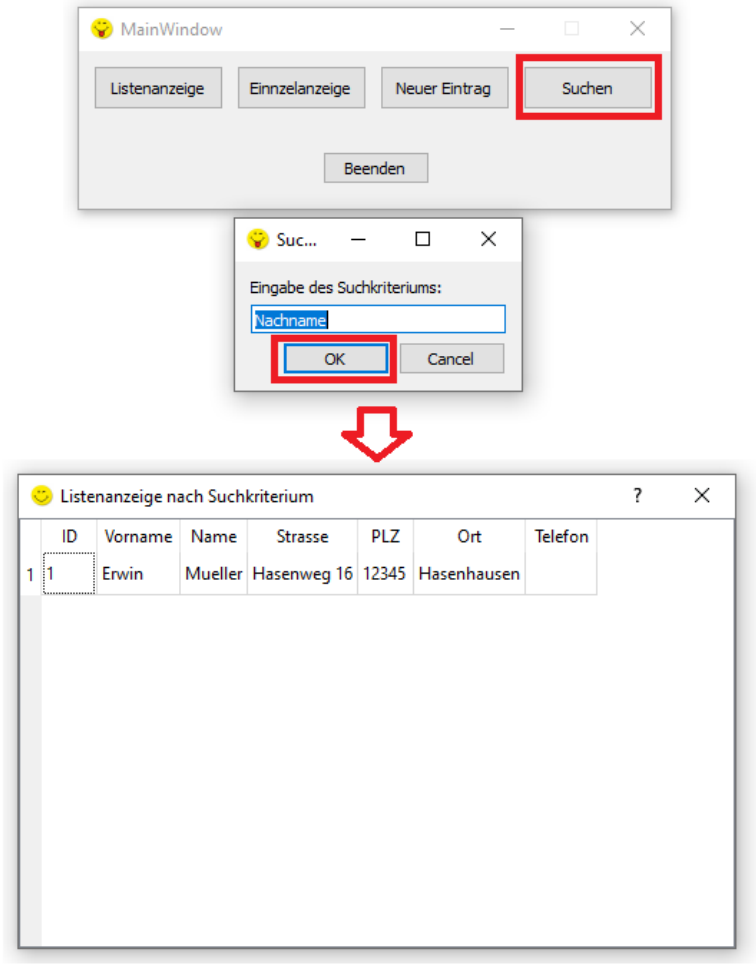

Abbildung 2.1: Ergebnis der Aufgabe 2

Abbildungsverzeichnis

Listings