G. Einsendeaufgabe

Objektorientierte Programmierung mit C++ und Qt

Code:
CPBS 16E-XX1-N01

Name:	Vorname:
Rein	Wjatscheslaw
Postleitzahl und Ort:	**Straße:**
14193 Berlin	Cunostr. 54A
Studien- bzw. Vertrags-Nr.:	**Lehrgangs-Nr.:**
800463563	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 Texteditor aus diesem Studienheft eine Funktion Info, die in einem Dialog Informationen zum Programm anzeigt.

Der Aufruf der Funktion Info soll sowohl über ein Symbol als auch über einen Eintrag Info in einem Menü Hilfe möglich sein. *20 Pkt.*

Lösung:

Es wird eine neue Aktion namens „Info“ im Aktionseditor erstellt mit gewünschtem Icon versehen und in der Menüleiste sowie im Menü „Hilfe“ platziert (Abbildung 1.1). Automatisch wird ein Slot für diese Aktion angelegt. Angenommen und vorausgesetzt (auch für die Aufgabe 2), dass alle notwendige Headerdateien eingebunden sind, sowie benötigte Deklarationen vorgenommen wurden, lauten die Anweisungen im Slot wie folgt (Listing 1.1):

```
1   /*****************************************************************
2   * void MainWindow::on_actionInfo_triggered()
3   * Aufgabe 1
4   * Funktion: Dialog Informationen zum Programm wird anzeigt.
5   *****************************************************************/
6   void MainWindow::on_actionInfo_triggered()
7   {
8     //Bitte in einer Zeile eingeben
9     QMessageBox::information(this, "Information",
10    "Programmiert von Wjatscheslaw Rein");
11  }
12
```

Listing 1.1: Anweisungen für die Aufgabe 1

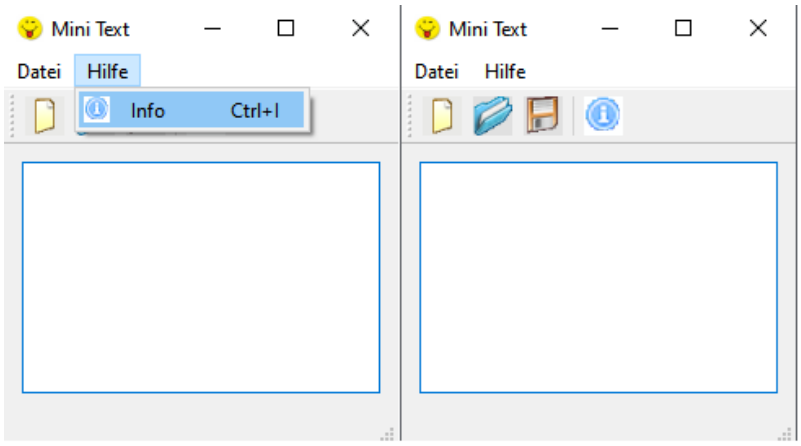

Abbildung 1.1: Ergebnis der Aufgabe 1

2 Aufgabe:

Aufgabe:

Beim Texteditor müssen Sie beim Speichern immer wieder den Namen angeben. Ändern Sie die Funktion zum Speichern so, dass nur dann nach einem Namen für die Datei gefragt wird, wenn die Datei nicht schon einmal gespeichert wurde.

Sie müssen also zwischen dem ersten und allen nachfolgenden Speichervorgängen unterscheiden. Beim ersten Speichern muss der Dialog zur Eingabe des Namens erscheinen, bei allen weiteren Speichervorgängen soll direkt gespeichert werden.

Bei einer Datei, die Sie geladen haben, soll auch beim ersten Speichern nicht der Dialog erscheinen, sondern die Datei soll direkt unter dem Namen gespeichert werden, unter dem Sie sie geladen haben.

Erweitern Sie den Editor auch um eine Funktion **Speichern unter...**. Diese Funktion soll nur im Menü **Datei** angeboten werden und in jedem Fall den Dialog zur Eingabe des Dateinamens anzeigen.

30 Pkt.

Lösung:

Damit der Name beim Speichern nur einmal abgefragt wird, soll dieses bei erster Eingabe in einem Attribut zwischengespeichert werden. Danach reicht es nur die Datei auf Existenz zu prüfen um unnötige Abfrage zu vermeiden (Listing 2.1).

```
1   /*****************************************************************************
2   * void MainWindow::dateiSpeichern()
3   * Aufgabe 2
4   * Funktion: Speichern einer Datei. Dabei wird der Name nur einmal abgefragt.
5   *****************************************************************************/
6   void MainWindow::dateiSpeichern()
7   {
8     if (QFile::exists(dateiName) == false)
9     {
10      //Bitte in einer Zeile eingeben. Den Speichern-Dialog erzeugen
11      dateiName = QFileDialog::getSaveFileName(this, "Datei speichern",
12      QDir::currentPath(), "Textdateien (*.txt)");
13    }
14    //wenn eine Datei ausgewaehlt wurde
15    if (dateiName.isEmpty() == false)
16    {
17      //Der Name an Hilfsfunktion zum Speichern uebergeben.
18      dateiSpeichernUnter(dateiName);
19    }
20  }
21
```

Listing 2.1: Änderung in der Methode Datei Speichern

Attribut der Klasse QString für den File-Name wird als „public“ in der Headerdatei deklariert und steht danach an jeder Stelle im Quellcode zur Verfügung. Außerdem wird selbst die Routine zum Speichern in eine eigene Hilfsfunktion bzw. Methode verlagert um Mehrfachverwendbarkeit zu erlauben (siehe Listing 2.2, Zeile 23). Um die Funktion Speichern Unter zu realisieren und über Menü Datei diese Aktion zur Verfügung zu stellen, geht man die gewöhnliche Schritte durch wie diese in der Aufgabe 1 beschrieben sind. Ein automatisches Erzeugen von dem Slot wird bevorzugt, da in der Aufgebe nichts Konkretes diesbezüglich vorgeschrieben wurde. In dem Slot wird ein Dialog zu der Name-Eingabe gestartet und untergeordnete Hilfsfunktion zum Speichern aufgerufen (Listing 2.2, Zeile 7).

```
1   /*****************************************************************
2   * void MainWindow::on_actionSpeichern_unter_triggered()
3   *
4   * Aufgabe 2
5   * Funktion: Der Slot fuer die Aktion Datei Speichern Unter...
6   *****************************************************************/
7   void MainWindow::on_actionSpeichern_unter_triggered()
8   {
9     //Bitte in einer Zeile eingeben. Den Speichern-Dialog erzeugen
10    dateiName = QFileDialog::getSaveFileName(this, "Datei speichern",
11    QDir::currentPath(), "Textdateien (*.txt)");
12    //Der Name an Hilfsfunktion zum Speichern uebergeben.
13    dateiSpeichernUnter(dateiName);
14  }
15
16  /*****************************************************************
17  * void MainWindow::dateiSpeichernUnter()
18  *
19  * Aufgabe 2
20  * Funktion: Eine verlagerte Hilfsfunktion zum Speichern.
21  * Die Datei wird unter vorgegebenem Namen gespeichert.
22  *****************************************************************/
23  void MainWindow::dateiSpeichernUnter(QString dateiNameToSave)
24  {
25    //eine Instanz von QFile fuer die Datei erzeugen
26    QFile *datei = new QFile(dateiNameToSave);
27    //die Datei im Modus Nur-Schreiben oeffnen
28    if (datei->open(QIODevice::WriteOnly))
29    {
30      //den kompletten Inhalt aus dem Widget in die Datei //schreiben
31      datei->write(ui->textEdit->toPlainText().toUtf8());
32      //eine Meldung in der Statusleiste anzeigen
33      ui->statusBar->showMessage("Die Datei wurde gespeichert.", 5000);
34      //die Datei wieder schliesssen
35      datei->close();
36    }
37  }
38
```

Listing 2.2: Methoden bzw. Slots für die Aufgabe 2

3 Aufgabe:

Aufgabe:

Erstellen Sie eine Anwendung, die Einträge in einem Listenfeld in einer XML-Datei speichert. Die Datei soll im aktuellen Pfad der Anwendung gespeichert werden.

Die Einträge in der Liste sollen vom Anwender selbst über ein Eingabefeld erfasst werden können.

Wie Sie die Knoten und Einträge benennen, ist Ihnen selbst überlassen. Achten Sie aber bitte darauf, dass Sie die Vorgaben für die Struktur einer XML-Datei einhalten.

Ob die Datei korrekt angelegt wurde, können Sie sehr einfach überprüfen, indem Sie sie mit einem Browser anzeigen lassen.

50 Pkt.

Lösung:

Das Ergebnis der Aufgabe könnte wie folgt aussehen (Abbildung 3.1):

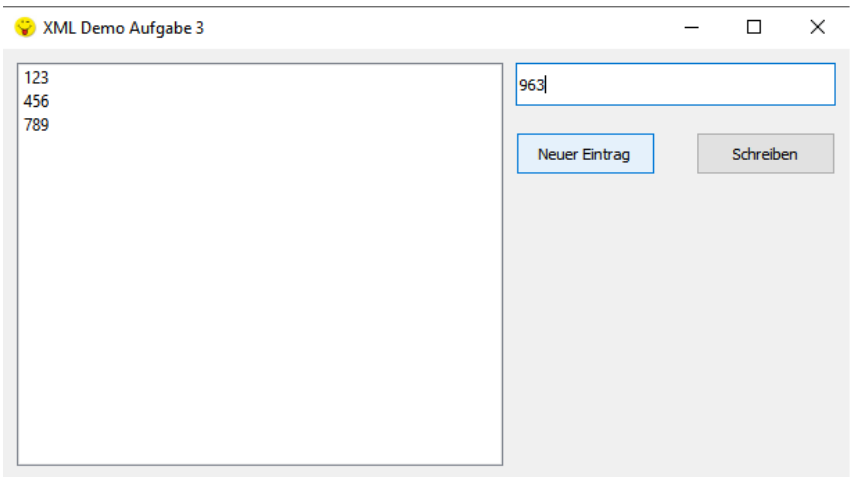

Abbildung 3.1: Ergebnis der Aufgabe 3

Basierend auf der Beispielaufgebe „XMLDemo3“ aus dem Studienheft wird die geforderte Lösung implementiert. Die benötigten Elemente wie ein Listenfeld, zwei Schaltflächen und ein Eingabefeld werden im Formular an den ausgewählten Stellen platziert. Es gibt keine Besonderheiten bei der Konfiguration mit einer Ausnahme, dass die Slots für die Schaltflächen automatisch generiert werden (Listings 3.2 und 3.3). Es wird auch gewährleistet, dass die Datei im aktuellen Pfad der Anwendung gespeichert wird (Listing 3.1).

```
MainWindow::MainWindow(QWidget *parent):QMainWindow(parent),ui(new Ui::MainWindow)
{
  ui->setupUi(this);
  this->setWindowIcon(QIcon(iconNamen));
  //Aufgabe 3
  //Funktion: Datei wird im aktuellen Pfad der Anwendung gespeichert
  xmlName = QCoreApplication::applicationFilePath() + ".xml";

  datei = new QFile(xmlName);
}

```

Listing 3.1: Anpassung im Konstruktor

```
/*****************************************************************
* void MainWindow::on_buttonSchreiben_clicked()
*
* Aufgabe 3
* Funktion: Beim Anklicken der Taste werden die Eintraege von einem
* Listenfeld in einer XML-Datei gespeichert.
*****************************************************************/
void MainWindow::on_buttonSchreiben_clicked()
{
  ...
  //die Eintraege schreiben

  //Aufgabe 3
  //Funktion: Die Eintraege von einem Listenfeld werden in einer XML-Datei
  //gespeichert. Zusaetzlich werden die Knoten Namen mit einer laufenden der
  //Zeile entsprechenden Nummer versehen.
  for (int index = 0; index < ui->listWidget->count(); index++)
  {
    ui->listWidget->setCurrentRow(index);
    sIndex = "Eintrag_"+(QString::number(index));
    xmlWriter->writeTextElement(sIndex, ui->listWidget->currentItem()->text());
  }

  xmlWriter->writeEndElement();
  ...
}

```

Listing 3.2: Ein Slot zum Schreiben in eine XML Datei

```
/*****************************************************************
* void MainWindow::on_pushButtonList_clicked()
*
* Aufgabe 3
* Funktion: Die Liste wird vom Anwender mit den Daten befuellt.
*****************************************************************/
void MainWindow::on_pushButtonList_clicked()
{
  ui->listWidget->addItem(ui->lineEdit->text());
  ui->lineEdit->clear();
  ui->lineEdit->setFocus();
}

```

Listing 3.3: Ein Slot für Befühlen einer Liste

Listings