H. Einsendeaufgaben

Objektorientierte Programmierung mit C++ und Qt

Code:
CPBS11E-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:

Qt verfügt über eine Klasse `QBitmap` zur Verarbeitung von Bildern. Welche Bilder können Sie mit dieser Klasse verarbeiten? Benutzen Sie die Qt-Referenzdokumentation, um die Frage zu beantworten.

10 Pkt.

Lösung:

Die Reference Documantation gibt keine eindeutigen Angeben über von QBitmap unterstützte Bildformate, außer dass es ein monochromes Zeichnungsvorrichtung ist, das hauptsächlich zum Erstellen von benutzerdefinierten QCursor- und QBrush-Objekten, zum Konstruieren von QRegion-Objekten und zum Festlegen von Masken für Pixmaps und Widgets verwendet wird. Allerdings ist QBitmap eine Unterklasse von QPixmap, die wiederum folgende Bildformate akzeptiert (siehe Tabelle 1.1):

allgemeine Qt Bildformate		
Format	Description	Qt's support
BMP	Windows Bitmap	Read/write
GIF	Graphic Interchange Format (optional)	Read
JPG	Joint Photographic Experts Group	Read/write
JPEG	Joint Photographic Experts Group	Read/write
PNG	Portable Network Graphics	Read/write
PBM	Portable Bitmap	Read
PGM	Portable Graymap	Read
PPM	Portable Pixmap	Read/write
XBM	X11 Bitmap	Read/write
XPM	X11 Pixmap	Read/write

Tabelle 1.1: allgemeine Qt Bildformate

2 Aufgabe:

Aufgabe:

Erweitern Sie das Programm mit der Digitaluhr so, dass zusätzlich die Sekunden angezeigt werden und bei der Anzeige des Datums das Jahr erscheint. Dazu müssen Sie auch die Eigenschaften der LCD-Anzeige verändern.

20 Pkt.

Lösung:

- Schritt 1. Die Header-Datei bleibt für diese Aufgabe unverändert.
- Schritt 2. Die Änderungen in Quelldatei und der Methode zeigeUhrzeit() vornehmen.

  Die Methode zeigeUhrzeit sollte in der Lage sein auch die Sekunden anzuzeigen. Dafür wird bei der Methode currentTime() 8 statt 5 Zeichen aus dem String entnommen (siehe Listing 2.1, Zeile 10). Außerdem wird in derselben Methode zusätzlich ein Doppelpunkt im String an der Stelle [5] (2.1, Zeile 20) ausgeblendet und die Eigenschaft von der Anzeige auf 8 Stellen (2.1, Zeile 22, setDigitCount) erweitert.

```
1   /*##########################################
2   Einsendeaufgabe 11.2
3   ##########################################*/
4   ...
5   //die Methode zeigeUhrzeit()
6   void Digitaluhr::zeigeUhrzeit()
7   {
8     //die Zeit abfragen und aufbereiten
9     QString zeitAnzeige;
10    zeitAnzeige = QTime::currentTime().toString().left(8);
11    //den Status des Doppelpunkts aendern
12    if (doppelpunkt == false)
13    doppelpunkt = true;
14    else
15    doppelpunkt = false;
16    //den Doppelpunkt abschalten
17    if (doppelpunkt == false)
18    {
19      zeitAnzeige[2] = ' ';
20      zeitAnzeige[5] = ' ';
21    }
22    setDigitCount(8);
23    display(zeitAnzeige);
24  }
25  ...
26
```

Listing 2.1: Listing von Quelldatei digitaluhr.cpp und der Methode zeigeUhrzeit()

- Schritt 3. Die Änderungen in Quelldatei und der Methode zeigeDatum() vornehmen.

  Als erstes wird ein String aus der Methode currentDate() um den Jahr ("yyyy") ergänzt (siehe Listing 2.2, Zeile 14) und anschließend die Eigenschaft von der Anzeige auf 10 Stellen (2.2, Zeile 15) erweitert.

```
1   /*##########################################
2   Einsendeaufgabe 11.2
3   ##########################################*/
4   ...
5   //die Methode zeigeDatum()
6   void Digitaluhr::zeigeDatum()
7   {
8     //lauft der Timer fuer das Datum noch?
9     //dann verlassen wir die Methode direkt wieder
10    QString datumAnzeige;
11    if (timerDatum->isActive() == true)
12    return;
13    //das Datum aufbereiten und anzeigen
14    datumAnzeige = QDate::currentDate().toString("dd.MM.yyyy");
15    setDigitCount(10);
16    display(datumAnzeige);
17    //den Timer fuer die Zeit anhalten
18    //sonst verschwindet das Datum nach 1 Sekunde wieder
19    timerZeit->stop();
20    //den Timer fuer das Datum starten
21    timerDatum->start(2000);
22  }
23  ...
24
```

Listing 2.2: Listing von Quelldatei digitaluhr.cpp und der Methode zeigeDatum()

- Schritt 4. Die Änderungen in Quelldatei und der Methode stopDatum() vornehmen.

  Zum Schluss sollte noch die Methode stopDatum() soweit angepasst werden, damit die Anzeige der Uhrzeit wieder die richtige Anzahl an der Zeichen darstellt. Das wir durch Ergänzung bereits bekannter Eigenschaft anhand der Methode setDigitCount() auf 8 Stellen erreicht (siehe Listing 2.3, Zeile 12).

```
1   /*##########################################
2   Einsendeaufgabe 11.2
3   ##########################################*/
4   ...
5   void Digitaluhr::stopDatum()
6   {
7     //den Timer fuer die Uhr anhalten
8     timerDatum->stop();
9     //den Timer fuer die Uhrzeit wieder starten
10    timerZeit->start(1000);
11    //die Uhrzeit wieder anzeigen
12    setDigitCount(8);
13    zeigeUhrzeit();
14  }
15  ...
16
```

Listing 2.3: Listing von Quelldatei digitaluhr.cpp und der Methode stopDatum()

3 Aufgabe:

Aufgabe:

Erstellen Sie ein Programm für eine Bilderschau. Es sollen zwei Bilder geladen werden, die abwechselnd jeweils für fünf Sekunden angezeigt werden.

Die Bilderschau soll automatisch nach dem Start des Programms beginnen. Eine Auswahl der Bilder durch den Anwender müssen Sie nicht implementieren. Sie können also zwei fest vorgegebene Bilder benutzen. Das Programm soll außerdem über eine Schaltfläche zum Beenden verfügen.

Zwei Tipps zur Lösung:

Für das Öffnen und Anzeigen der Bilder können Sie eine Methode erstellen, die so ähnlich arbeitet wie die Methode `oeffneBilddatei()` aus dem Bildbetrachter. Ändern Sie die Methode aber so, dass der Name der Datei als Argument vom Typ `QString` übergeben wird.

Um die Bilder zu wechseln, überprüfen Sie, welche Datei gerade angezeigt wird, und benutzen dann den jeweils anderen Dateinamen.

Die Aufgabe lässt sich sehr viel einfacher lösen, als es vielleicht beim ersten Durchlesen scheint.

70 Pkt.

Lösung:

- Schritt 1. Die Änderungen an der Header-Datei durchführen.

  Die Klassen für Widgets, die nicht gebraucht werden (QLineEdit), sind gelöscht und neue (QTimer), die eine Verwendung im Quellcode finden, werden eingebunden (include). Es werden drei, obwohl nur zwei davon zu bewerten sind, Slots deklariert und entsprechende Attributen und Variablen vorgesehen. Relevanten Änderungen in der Header-Datei (siehe Listing 3.1) sind:

  - void toggelnBilddatei(): Slot zum Bildwechsel;
  - int oeffneBilddatei(QString): Slot zum Bildöffnen. Übernimmt die Variable vom Typ QString (Dateiname) und gibt eine Rückmeldung von Typ int für möglichen Errorhandler zurück;
  - bool triggerBild: Die Variable um einen aktuellen Zustand zu speichern und somit einen Bildwechsel zu ermöglichen;
  - QTimer *timerZeit: Eine Zeigervariable vom Typ QTimer zur Implementierung des Bildaustauschs unter Verwendung eines Timers;
  - QString qPath: Zusätzliche Variable um aktuellen Verzeichnis und den Weg zu den Bildern zu speichern;

```
/*##########################################
Einsendeaufgabe 11.3
##########################################*/

#ifndef BILDBETRACHTER_H
#define BILDBETRACHTER_H
//die Header-Dateien einbinden
#include <QWidget>
#include <QLabel>
#include <QPushButton>
#include <QTimer>

//unsere eigene Klasse erbt von QWidget
class Bildbetrachter : public QWidget
{
  //das Makro Q_OBJECT
  Q_OBJECT
  public:
    //der Konstruktor
    Bildbetrachter();

  //die Slots
  private slots:
    void toggelnBilddatei();
    void ifClose();
    int oeffneBilddatei(QString);

  private:
    //die Attribute fuer die Widgets
    bool triggerBild;
    QLabel* einLabel, *bildLabel;
    QPushButton *beendenButton;
    QTimer *timerZeit;
    QString qPath;
};
#endif
```

Listing 3.1: Listing von Headerdatei bildbetrachter.h

- Schritt 2. Die Änderungen in Quelldatei und dem Konstruktor vom Bildbetrachter.

  Da es in der Aufgabenstellung um die Beschreibung von den grundsätzlichen Schritte die Rede ist, werden im Text nur solche erläutert, die alle andere nebensächliche Änderungen sind ausführlich im Quellcode kommentiert (siehe Listing 3.2).

  - triggerBild = false; //Anfangswert für die Variable zur Bildwechselerkennung;
  - timerZeit = new QTimer(this); //Erstellung von dem Timer;
  - timerZeit->start(1000); //Timer wird mit dem Wert von 1[s] geladen;
  - QObject::connect(timerZeit,SIGNAL(timeout()),this,SLOT(toggelnBilddatei())); //Eine Signal-Slot Verbindung wird hergestellt. Dabei ist ein zeitlich gesteuerte Bildwechsel gewährleistet;

```
1     ...
2     Bildbetrachter::Bildbetrachter()
3     {
4       triggerBild = false;
5       timerZeit = new QTimer(this);
6       //die Groesse und den Titel setzen
7       resize(250,250);
8       setWindowTitle("Smiley");
9
10      //die Widgets erzeugen fuer interne Debugging
11      einLabel = new QLabel(this);
12      einLabel->setGeometry(90, 250, 500, 10);
13
14      //"Platzhalter" bzw. Anzeigecontainer fuer die Bilder
15      bildLabel = new QLabel(this);
16      bildLabel->setGeometry(50, 30, 10, 10);
17
18      //der Beenden-Button
19      beendenButton = new QPushButton(this);
20      beendenButton->setGeometry(90, 200, 70, 30);
21      beendenButton->setText("Beenden");
22      //Signal mit dem Slot verbinden
23      QObject::connect(beendenButton,SIGNAL(clicked()),this,SLOT(ifClose()));
24
25      //Aktuelles Arbeitsverzeichnis, wo die Bilder abgespeichert sind, setzen
26      if (QDir::setCurrent("../bildbetrachter"))
27      {
28        einLabel->setText(QDir::currentPath());
29        qPath = QDir::currentPath();
30      }
31      else
32      {
33        einLabel->setText("ERR");
34      }
35
36      //Timer fuer die Bilderaustausch definieren
37      timerZeit->start(1000);
38      //Signal mit dem Slot verbinden
39      QObject::connect(timerZeit,SIGNAL(timeout()),this,SLOT(toggelnBilddatei()));
40    }
41    ...
42
```

Listing 3.2: Listing von Quelldatei bildbetrachter.cpp und dem Konstruktor

- Schritt 3. Die neue Methode toggelnBilddatei().

  Nachdem die kompletten Pfade (Listing 3.3, Zeilen 9..10) zu den einzelnen Bildern aufgebaut sind und der Bildwechsel durch Trigger-Variable (3.3, Zeile 13) gesichert ist, folgen die Methoden-Aufrufe von oeffneBilddatei() mit der Übergabe von entsprechenden vollständigen Bildnamen. Eine zusätzlich realisierte Fehlerbehandlung, die in einer verschachtelten IF-Abfrage implementiert ist, dient zur besseren Handhabung und sollte nicht bewertet werden. In dem Fall wenn ein Bild aus irgendeinem Grund nicht geöffnet werden könnte, steht dem Benutzer ein Dialogfenster zur Verfügung in dem die Bilder direkt ausgewählt werden können.

```
1     ...
2     //Slot zum Bildwechsel
3     void Bildbetrachter::toggelnBilddatei()
4     {
5       QString DateiName1, DateiName2;
6       static int ret;
7
8       //Komplette Pfad zu den Bildern
9       DateiName1 = qPath + "/Bild1";
10      DateiName2 = qPath + "/Bild2";
11
12      //Zustandswechsel
13      triggerBild = !triggerBild;
14
15      //Routine zum Bildwechsel und moeglichen Fehlerbehandlung
16      if (ret == 0)
17      {
18        if (triggerBild == false)
19        {
20          ret = oeffneBilddatei(DateiName1);
21          //Info zum Debugging
22          einLabel->setText(DateiName1);
23        }
24        else
25        {
26          ret = oeffneBilddatei(DateiName2);
27          //Info zum Debugging
28          einLabel->setText(DateiName2);
29        }
30      }
31      else
32      {
33        //Abfrage des Speicherortes im Fehlerfall
34        qPath = QFileDialog::getExistingDirectory();
35        ret = 0;
36      }
37    }
38
```

Listing 3.3: Listing von Quelldatei bildbetrachter.cpp und der Methode toggelnBilddatei()

- Schritt 4. Die geänderte Methode oeffneBilddatei(QString).

  Die Methode zum Öffnen von Bildern (siehe Listing 3.4) ist durch Übernahme von einem QString Parameter und einer Return-Variable zum Fehlerbehandlung nur wenig geändert. Bei dem Aufruf wird der komplette Name des Bildes übergeben. Mit einer, bereits bekannter, Methode .load() der Klasse QPixmap wird entsprechende Bild geladen (3.4, Zeile 8). Die Zusätzlich realisierte Fehlermeldung und somit auch die Rückgabe an die aufrufende, in unserem Fall die Methode toggelnBilddatei(), Funktion ist nicht ein Teil der Aufgabe und sollte nicht bewertet werden.

```
1     ...
2     //Slot zum Oeffnen den Bildern
3     int Bildbetrachter::oeffneBilddatei(QString FileName)
4     {
5       QPixmap bild;
6       //konnte das Bild geladen werden?
7       //dann anzeigen
8       if (bild.load(FileName))
9       {
10        //die Groesse wird dabei an die Breite 150 angepasst
11        bild = bild.scaledToWidth(150);
12        //im Label anzeigen
13        bildLabel->setPixmap(bild);
14        //die Groesse des Labels an die Groesse des Bildes anpassen
15        bildLabel->resize(bild.size());
16      }
17      else
18      {
19        //Bitte in einer Zeile eingeben
20        QMessageBox::warning(this, "Fehler beim Oeffnen der Datei",
21        "Die Grafikdatei konnte nicht geoeffnet werden.\n");
22
23        einLabel->setText(FileName);
24        return -1;
25      }
26      return 0;
27    }
28
```

Listing 3.4: Listing von Quelldatei bildbetrachter.cpp und der Methode oeffneBilddatei(QString)

Listings

Tabellenverzeichnis