HOCHSCHULE RUHR WEST
UNIVERSITY OF APPLIED SCIENCES
INSTITUT INFORMATIK

25.09.2020

Klausur

MMI und GUI-Programmierung

Sommersemester 2020

Studiengänge: AI, MTI, WI

Prüfer: Prof. Dr. Malte Weiß

Persönliche Angaben

Nachname	Vorname	Matrikelnummer

Bewertung

Frage	Punkte	Erreicht
1	4	
2	9	
3	8	
4	12	
5	12	

Frage	Punkte	Erreicht
6	12	
7	30	
8	5	
9	8	
Gesamt:	100	

Ablauf der Prüfung

- **Zeit für die Lösung dieses Aufgabenblatts**: 90 Minuten.
- **Erlaubte Hilfsmittel**: Cheat Sheet am Ende der Klausur. Handgeschriebener, ein- oder beidseitig beschriebener Notizzettel im DIN-A4-Format.
- **Nicht erlaubte Hilfsmittel**: Mobiltelefon, Kommunikation (E-Mail, Web, SMS, etc.). Die Verwendung eines nicht erlaubten Hilfsmittels wird als Täuschungsversuch gewertet.
- **Stift und Papier**: Schreiben Sie ausschließlich auf dem Papier der Aufgabenblätter. Sollten die Zwischenräume nicht ausreichen, können Sie zusätzliches Papier von der Aufsicht erhalten. Beschriften Sie dieses sofort mit ihrem Namen und Ihrer Matrikelnummer. Eigenes Papier ist nicht zulässig. Schreiben Sie ausschließlich mit dokumentenechten schwarzen oder blauen Stiften.
- **Vollständigkeit des Aufgabenblatts**: Die Klausur besteht aus den Aufgabenseiten 5 – 21 und einer Funktionsreferenz auf Seite R1. Die Aufgabenblätter sind doppelseitig bedruckt. Prüfen Sie, ob alle Blätter vorhanden sind.
- **Fragen**: Melden Sie sich, wenn Sie eine Frage haben. Die Aufsicht kommt zu Ihnen, verlassen Sie nicht unaufgefordert Ihren Platz.
- **Zwischenschritte**: Geben Sie Zwischenschritte an. So können Sie selbst bei falschem Endergebnis einen Teil der Punkte erreichen.

MMI

1. Definieren Sie den Begriff “Usability” gemäß der in der Vorlesung vorgestellten DIN-Norm (DIN EN ISO 9241-11). (4)

2. Erläutern Sie die drei menschlichen Deadlines und nennen Sie jeweils ein konkretes Beispiel, das **nicht** aus der Vorlesung stammt. (9)

Deadline 1:

Beispiel zu Deadline 1:

Deadline 2:

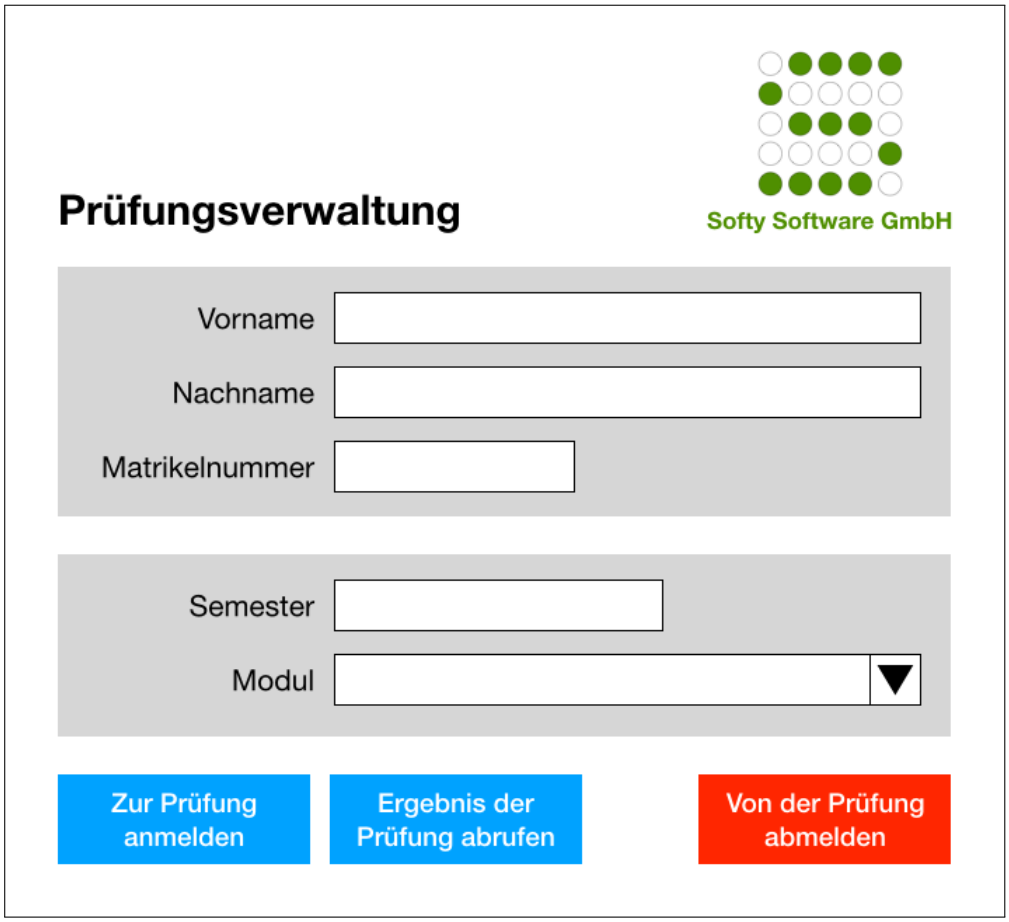
Beispiel zu Deadline 2:

Matrikelnummer:

Deadline 3:

Beispiel zu Deadline 3:

3. Die *Softy Software GmbH* möchte der Hochschule eine Anwendung zur Prüfungsverwaltung verkaufen. Die Firma hat diese GUI entworfen: (8)



Prof. Walte Meiß, der das Angebot der Firma begutachten soll, kommt zu folgendem Urteil: "Bei einem bemerkenswerten Mangel an Ästhetik setzt die Firma zumindest Gestaltgesetze wirkungsvoll ein."

Identifizieren Sie **vier** Gestaltgesetze, die im Wireframe eingesetzt wurden. Nennen Sie für jedes Gestaltgesetz den Namen und wie es in der GUI eingesetzt wurde.

Matrikelnummer:

Gesetz 1:

Gesetz 2:

Matrikelnummer:

Gesetz 3:

Gesetz 4:

C++

4. Beantworten Sie die folgenden Fragen:

   (a) Über welches Schlüsselwort wird eine Methode als polymorph markiert. (2)

   (b) Über welches Sprachkonstrukt verhindert man die Instanziierung einer vollständig ausimplementierten Klasse (abstrakte Klasse). (2)

   (c) Welches Schlüsselwort verwendet man, um einer globalen Funktion Zugriff auf die privaten Attribute und Methoden einer Klasse zu gewähren. (2)

   (d) Über welches Sprachkonstrukt stellt man beim Einsatz von Mehrfachvererbung sicher, dass eine gemeinsame Oberklasse nicht mehrfach geerbt wird? (2)

   (e) Nennen Sie eine Art von Attribut, die über eine Initialisierungsliste initialisiert werden **muss**. (2)

(f) Nehmen wir an, es gibt eine Template-Funktion, die das Maximum von zwei Werten berechnet, wobei der Typ der Werte über einen Template-Platzhalter definiert wird. (2)

Sie rufen die Funktionen dreimal im Code auf, zweimal mit `int`-Werten, einmal mit `double`-Werten.

Wie oft befindet sich die Funktion nach dem Kompilieren im Maschinencode?

5. Gegeben sei eine Klasse, die eine quadratische 2D-Matrix der Form $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ darstellt. (12)

   Die **Determinante** einer 2D-Matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ ist definiert als: $a \cdot d - b \cdot c$

   Überladen Sie die folgenden Operatoren:

   1. Den einstelligen Operator `!` (Ausrufezeichen) zur Berechnung der Determinante in der Klasse `Matrix2D`.
   2. Den zweistelligen Operator `<<`, um eine 2D-Matrix in einem Stream auszugeben.

   Somit soll folgender Code möglich sein:

```
Matrix2D mat(1, 2, 3, 4);
std::cout << mat << std::endl                    // (1, 2, 3, 4)
          << "det(mat): " << !mat << std::endl; // det: -2
```

   Ergänzen Sie den folgenden Code auf der folgenden Seite.

Matrikelnummer:

Headerdatei matrix2d.h

```
class Matrix2D
{
public:
    Matrix2D(double a, double b, double c, double d)
        : m_a(a), m_b(b), m_c(c), m_d(d) {}

    inline double a() { return m_a; }
    inline double b() { return m_b; }
    inline double c() { return m_c; }
    inline double d() { return m_d; }

private:
    double m_a, m_b, m_c, m_d;
```

```

};
```

Sourcedatei matrix2d.cpp

```
#include "matrix2d.h"
```

```

```

6. (a) Betrachten Sie das folgende Programm: (6)

```
#include <iostream>

class Value {
public:
    Value(int x) : m_x(x) {}

    void setValue(int x) { m_x = x; }
    int getValue()       { return m_x; }

private:
    int m_x;
};

int main()
{
    const Value v(1);
    std::cout << "value = " << v.getValue() << std::endl;
    return 0;
}
```

Das Programmierung ist syntaktisch korrekt, kompiliert aber trotzdem nicht. Was ist der Fehler?

Wie wird der Fehler korrigiert?

(b) Nehmen wir an, es gibt eine Klasse für Hamster (`Hamster`), die von einer polymorphen Klasse für Lebewesen erbt (`Lebewesen`). Ein Hamster besitzt eine eigene Methode, um seine Flauschigkeit als `double`-Wert zurückzugeben (`getFluffyness`). (6)

Bob hat folgende Methode geschrieben, um die Flauschigkeit eines Hamsters auszugeben:

```
void printHamsterFluffyness(Lebewesen* lebewesen) {
    Hamster* hamster = (Hamster*) lebewesen;

    std::cout << "Fluffyness of hamster: "
              << hamster->getFluffyness() << std::endl;
}
```

Alice behauptet, die Funktion sei unsicher und könnte zum Absturz führen.

Schreiben Sie eine neue Variante der Funktion, die **sicherer** ist und nicht abstürzen kann:

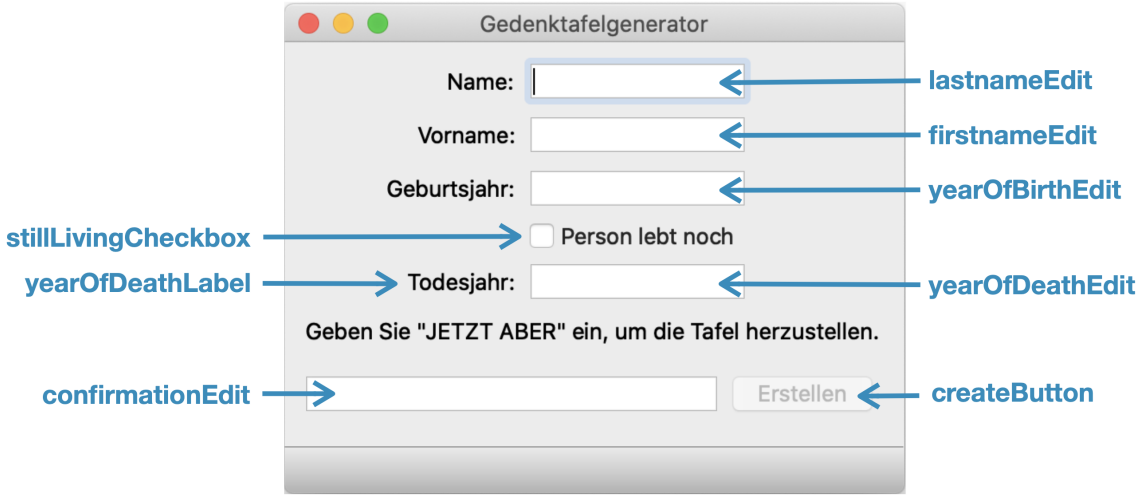
```
void printHamsterFluffyness(Lebewesen* lebewesen) {
```

```
}
```

Qt

7. Die Firma *Bankrupt Software AG* hatte mit ihren bisherigen Produkten leider keinen großen Erfolg. Das neue Produkt Gedenktafelgenerator™ soll das Unternehmen jetzt aus den roten Zahlen bringen.

   Gedenktafelgenerator™ erlaubt es, den Schriftzug für eine Gedenktafel zu definieren, der dann im Anschluss von einer CNC-Fräse in Stein gefräst wird. Die GUI sieht wie folgt aus. Objektnamen sind blau markiert.



   Das Hauptfenster (Klasse `MainWindow`) soll wie folgt funktionen:

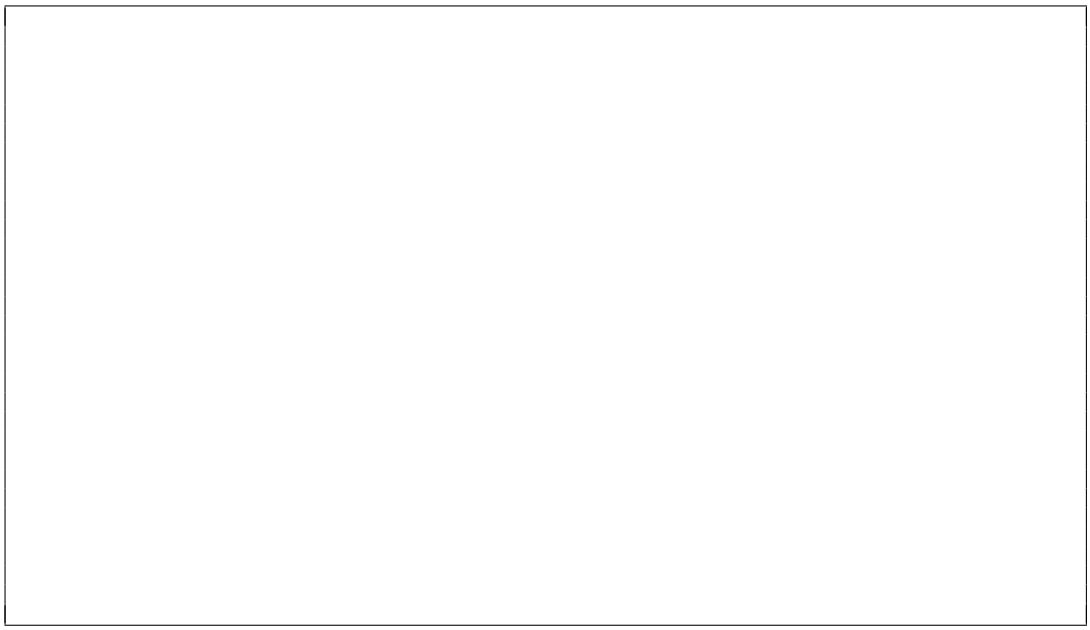
   - Die Felder für Name, Vorname, Geburtsjahr und Todesjahr sind normale Textfelder.
   - Das Label und das Feld für Todesjahr sind nur **sichtbar**, wenn die Checkbox "Person lebt noch" **nicht angehakt** ist.
   - Die Schaltfläche Erstellen ist genau dann **aktiviert**, wenn im Feld confirmationEdit der Text "JETZT ABER" eingetragen ist. (Hintergrund: Das Fräsen der Tafel ist teuer, daher sollte es nicht versehentlich passieren).
   - Klickt ein Nutzer auf "Erstellen", passiert folgendes:
     1. Ist eines der **sichtbaren** Felder leer, wird eine Fehlermeldung ausgegeben.
     2. Ansonsten wird die Meldung "Tafel wird hergestellt" ausgegeben und ein Signal createTablet ausgelöst.
   - Beim Start des Programms sollen alle Felder leer und "Person lebt noch" nicht angehakt sein.

   Hinweise: Sie können davon ausgehen, dass ein Nutzer immer ordentliche Jahreszahlen eingibt. Das **Qt Cheat Sheet** auf der letzten Seite enthält hilfreiche Informationen.

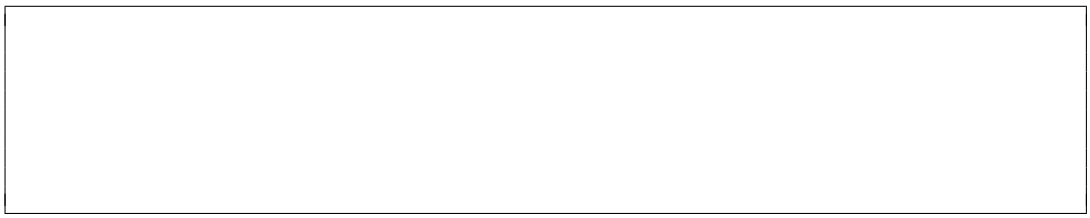
(a) Überlegen Sie, welche Signals und Slots Sie benötigen und ergänzen Sie die folgende Header-Datei von `MainWindow` entsprechend. (6)

Der Slot `onCreateButtonPushed` für das Erstellen der Tafel ist schon definiert.

```
#include <QMainWindow>

namespace Ui { class MainWindow; }

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:

    void onCreateButtonPushed();
```



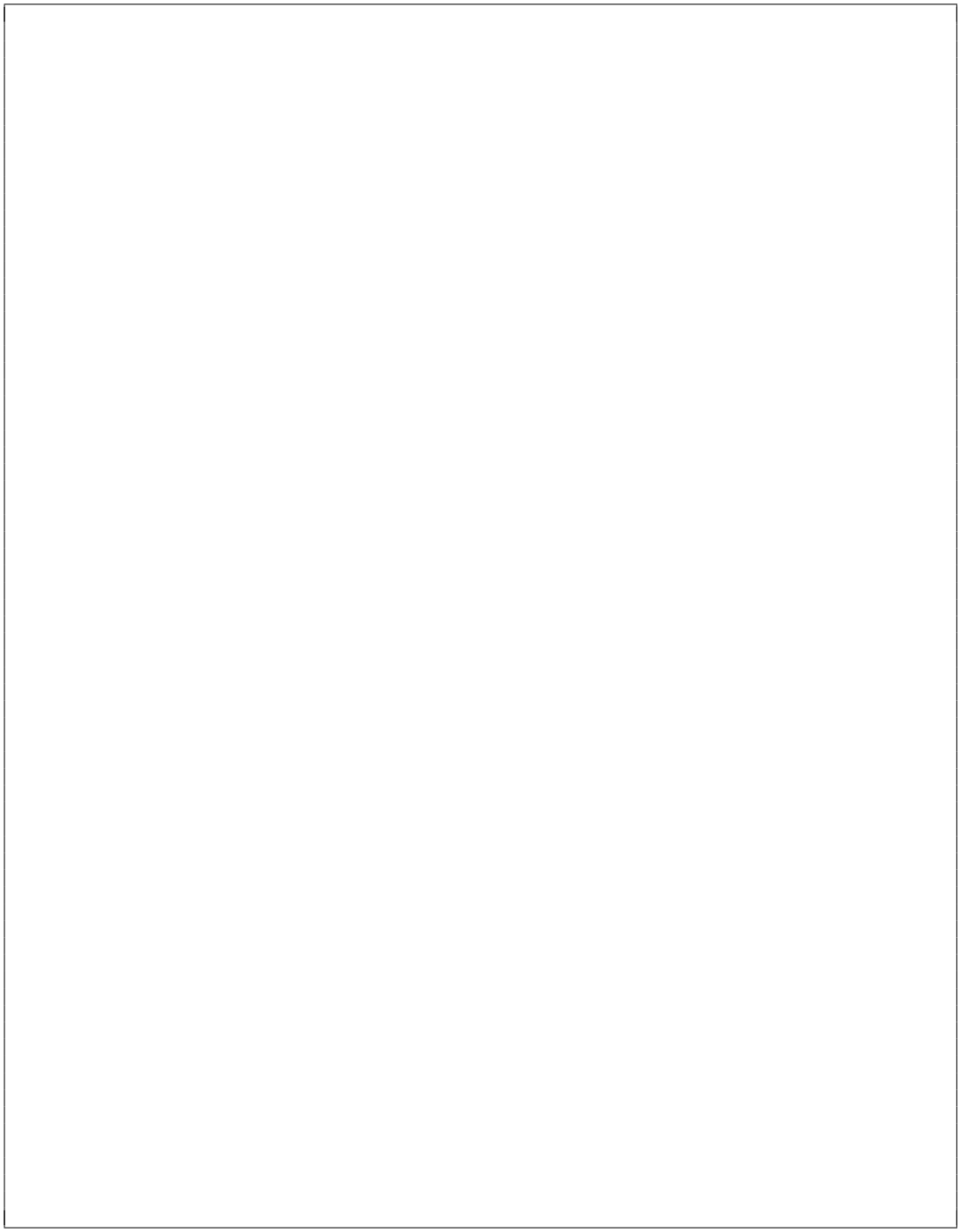
```
signals:
```



```
};
```

(b) Ergänzen Sie nun die Implementierung des Slots `onCreateButtonPushed()`, gemäß des oben beschriebenen Verhaltens. (8)

```
void MainWindow::onCreateButtonPushed()
{
```



```
    QMessageBox::information(this, "Fertig",
        "Tafel wird hergestellt.", "Ok");
}
```

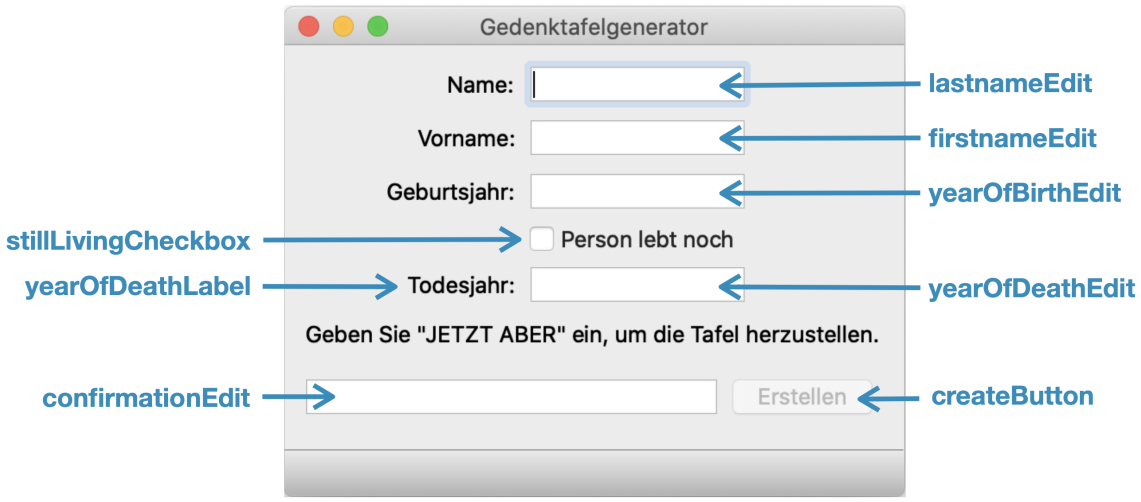
(c) Implementieren Sie nun den Konstruktor. Hier sollen die Signale und Slots korrekt verbunden werden. Außerdem sollte die GUI sinnvoll initialisiert werden. (8)

```
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
```

```
}
```

(d) Implementieren Sie nun die verbleibende GUI-Logik Ihrer Applikation in der Source-Datei. (8)

8. Betrachten wir nochmal die GUI aus Aufgabe 7. (5)



Erklären Sie an diesem Beispiel, wie das Layout einer solchen GUI in Qt erstellt wird.

Tipp: In der Referenz finden Sie die relevante Qt-Klassen.

Nebenläufigkeit

9. Nehmen wir an, es gibt ein Programm, das über einen Sensor Daten aufzeichnet. Zwei Funktionen stehen zur Verfügung, um mit diesen Sensordaten zu arbeiten: (8)

   - `logSensorValue(value)` schreibt einen Sensorwert ans Ende einer Logdatei.
   - `getAverageSensorData()` liest die aktuelle Logdatei komplett ein und gibt die Summe aller Sensorwerte zurück.

   Wie Sie wissen, ist gleichzeitiger Schreib- und Lesezugrifff auf eine Datei nicht zulässig, während gemeinsames Lesen in Ordnung ist. Schreiben Sie den folgenden Code so um, dass **keine Konflikte beim nebenläufigen Aufruf** der Funktionen gibt. Wählen Sie eine möglichst **effiziente** Lösung.

```
void logSensorValue(int value) {



    FILE* file = fopen("sensor.txt", "at");  // Add log entry to file
    fprintf(file, "%d\n", value);
    fclose(file);



}

int getSensorValueSum() {



    FILE* file = fopen("sensor.txt", "rt");  // Open log file and sum all values
    double sum = 0, value;
    while(!feof(file)) {
        fscanf(file, "%lf\n", &value);
        sum += value;
    }
    fclose(file);



    return sum;
}
```

Qt Cheat Sheet

Signale

QPushButton / QAbstractButton

```
void    clicked(bool checked = false)
void    toggled(bool checked)
void    pressed()
void    released()
```

QLineEdit

```
void    textChanged(const QString &text)
void    editingFinished()
void    inputRejected()
void    returnPressed()
void    selectionChanged()
```

Methoden

QLineEdit

Text auslesen
```
QString text() const
```

Text setzen
```
void setText(const QString &)
```

QAbstractButton

Auslesen: Ist Checkbox angehakt?
```
bool isChecked()
```

QString

Länge ermitteln
```
int size() const
```

Slots

QWidget

```
bool    close()
void    hide()
void    setDisabled(bool disable)
void    setEnabled(bool)
void    setFocus()
void    setHidden(bool hidden)
void    setVisible(bool visible)
void    setWindowModified(bool)
void    setWindowTitle(const QString &)
void    show()
void    update()
```

QAbstractButton

```
void    animateClick(int msec = 100)
void    click()
void    setChecked(bool)
void    setIconSize(const QSize &size)
void    toggle()
```

QPushButton *(erbt von QAbstractButton)*

```
void    showMenu()
```

QLineEdit

```
void    clear()
void    copy() const
void    cut()
void    paste()
void    redo()
void    selectAll()
void    setText(const QString &)
void    undo()
```

Layout-Manager

QHBoxLayout
QVBoxLayout
QFormLayout
QBoxLayout
QGridLayout
QStackedLayout

Thread-Synchronisation

QMutex
QReadWriteLock
QSemaphore