HOCHSCHULE RUHR WEST
UNIVERSITY OF APPLIED SCIENCES
INSTITUT INFORMATIK

18.03.2019

Klausur

MMI und GUI-Programmierung

Wintersemester 2018/2019

Studiengänge: AI, MTI, WI

Prüfer: Prof. Dr. Malte Weiß

Persönliche Angaben

Nachname	Vorname	Matrikelnummer

Bewertung

Frage	Punkte	Erreicht
1	3	
2	4	
3	8	
4	12	
5	13	
6	8	
7	5	
8	3	
9	4	

Frage	Punkte	Erreicht
10	4	
11	2	
12	15	
13	5	
14	8	
15	6	
16	0	
Gesamt:	100	

Ablauf der Prüfung

- **Zur Person**: Tragen Sie Namen, Matrikelnummer auf der ersten Seite ein. Legen Sie Ihren Studentenausweis auf den Tisch.
- **Matrikelnummer**: Schreiben Sie **auf jede Seite der Klausur** Ihre Matrikelnummer. Seiten ohne Matrikelnummer können nicht bewertet werden.
- **Zeit für die Lösung dieses Aufgabenblatts**: 120 Minuten.
- **Erlaubte Hilfsmittel**: Cheat Sheet am Ende der Klausur. Darüber hinaus sind keine Hilfsmittel erlaubt.
- **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 – 23 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). (3)

2. Erläutern Sie die beiden Begriffe **User Experience Design (UX-Design)** und **User Interface Design (UI-Design)** und grenzen Sie sie voneinander ab. (4)

3. Nachdem die Klingonen die Erde erobert haben, haben sie beschlossen, die Planeten des Sonnensystems in ihrem brandneuen Web-Shop Planetenshop 24™ zu verkaufen. Der UI-Designer K'Ratak hat die folgende Webseite entworfen: (8)

Identifizieren Sie **vier** Gestaltgesetze, die K'Ratak eingesetzt hat. Nennen Sie für jedes Gestaltgesetz den Namen und wie es im Webshop **eingesetzt** oder **bewusst verletzt** wurde.

(Bitte benutzen Sie die nächste Seite)

Gesetz 1:

Gesetz 2:

Gesetz 3:

Gesetz 4:

4. (a) Beschreiben Sie die drei menschlichen Deadlines, die beim responsiven Design eine Rolle spielen. (6)

(b) Betrachten Sie nun das folgende Szenario: (6)

> In der Entwicklungsumgebung CodingHero können Java-Projekte wie folgt importiert werden: Der Nutzer wählt den Eintrag "Importieren" im Menü "Datei". Nun öffnet sich Fenster, in der Nutzer verschiedene Angaben zum Import tätigen kann. Diese Angaben sind auf mehrere Fenster verteilt. Mit "Weiter" kommt man jeweils auf die nächste Ansicht. In der letzten Ansicht befindet sich der Knopf "Jetzt importieren", der den Eingabeprozess abschließt und anschließend den Import durchführt.

Wie man das Wissen über die menschlichen Deadlines einsetzen, um dieses Szenario möglichst benutzerfreundlich zu gestalten? Geben Sie **für jede der drei menschlichen Deadlines ein konkretes Beispiel** an.

C++

5. Betrachten Sie die folgende Klasse, die einen 3D-Vektor der Form $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ darstellt.

```
class Vector3D
{
public:
    Vector3D(float x, float y, float z) : m_x(x), m_y(y), m_z(z) {}

    inline float x() { return m_x; }
    inline float y() { return m_y; }
    inline float z() { return m_z; }

private:
    float m_x, m_y, m_z;
};
```

(a) Das **Kreuzprodukt** zwischen zwei Vektoren $\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$ und $\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}$ ist definiert als (6)

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \begin{pmatrix} y_1 \cdot z_2 - z_1 \cdot y_2 \\ z_1 \cdot x_2 - x_1 \cdot z_2 \\ x_1 \cdot y_2 - y_1 \cdot x_2 \end{pmatrix}.$$

Überladen Sie den Operator `%` zur Berechnung des Kreuzprodukts in der Klasse `Vector3D`, so dass folgender Code möglich ist:

```
Vector3D a(1,2,3);
Vector3D b(4,3,3);

Vector3D c = a % b;   // Kreuzprodukt berechnen

qDebug() << c.x(); // -3
qDebug() << c.y(); // 9
qDebug() << c.z(); // -5
```

Ergänzen Sie den Code auf der nächsten Seite.

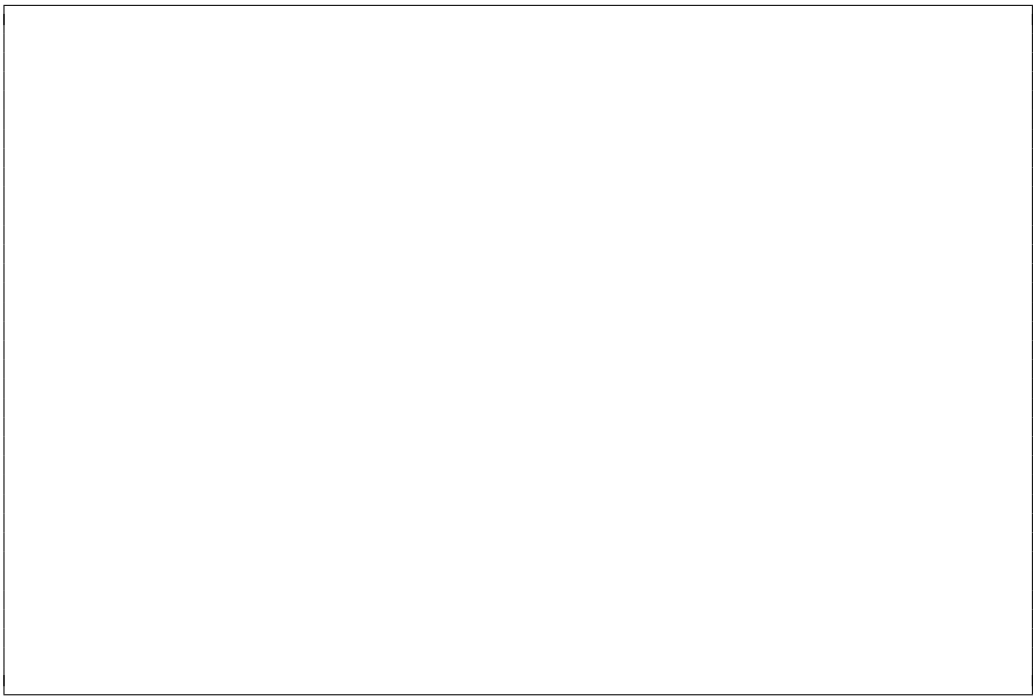
Headerdatei vector3d.h

```
class Vector3D
{
public:
    Vector3D(float x, float y, float z) : m_x(x), m_y(y), m_z(z) {}

    inline float x() { return m_x; }
    inline float y() { return m_y; }
    inline float z() { return m_z; }

private:
    float m_x, m_y, m_z;
```

```
};
```

Sourcedatei vector3d.cpp

```
#include "vector3d.h"
```



(b) Wir würden jetzt gerne die Möglichkeit haben, einen Vektor einfach über `std::cout` auszugeben: (7)

```
Vector3D v(1,2,3)
std::cout << c;
```

Überladen Sie dazu den Operator `<<`. Greifen Sie dabei direkt auf die privaten Attribute der Klasse Vector3D zu (keine Getter-Aufrufe!). Vervollständigen Sie den folgenden Code.

Tipp: ostream und Freundschaft.

Headerdatei vector3d.h

```
class Vector3D
{
public:
    Vector3D(float x, float y, float z) : m_x(x), m_y(y), m_z(z) {}

private:
    float m_x, m_y, m_z;
```

```

```

```
};
```

Sourcedatei vector3d.cpp

```
#include "vector3d.h"
```

```

```

6. Die folgenden Codezeilen verwenden unsichere C-Casts. Wählen Sie jeweils einen sichereren C++-Cast und korrigieren Sie die betroffene Codezeile.

(a) (2)

```
float x = 2.5f;
int i = (int) x;
```

Korrigierte Codezeile:

(b) (2)

```
void hamsterStreicheln(Lebewesen* lebewesen) {
  Hamster* hamster = (Hamster*) lebewesen;
  // Future Work: Hamster tatsächlich streicheln
}
```

Korrigierte Codezeile:

(c) (2)

```
// 32-Bit-Float-Kodierung als 32-Bit-Zahl ausgeben
float f = 1.25f;
qint32* lp = (qint32*) &f;
qDebug() << *lp;
```

Korrigierte Codezeile:

(d) (2)

```
int getSquared(int *p) { return *p * *p; }

int main()
{
    const int x = 5;
    qDebug() << getSquared(&x);
    return 0;
}
```

Korrigierte Codezeile:

Qt

7. Qt gilt als **plattformunabhängiges** Framework.

   (a) Was bedeutet das? (2)

   (b) Beschreiben Sie einen Vor- und einen Nachteil von plattformunabhängigen Frameworks. (3)

   Vorteil:

   Nachteil:

8. Nennen Sie drei Eigenschaften, die eine von QObject abgeleitete Klasse von einer normalen C++-Klasse unterscheidet. (3)

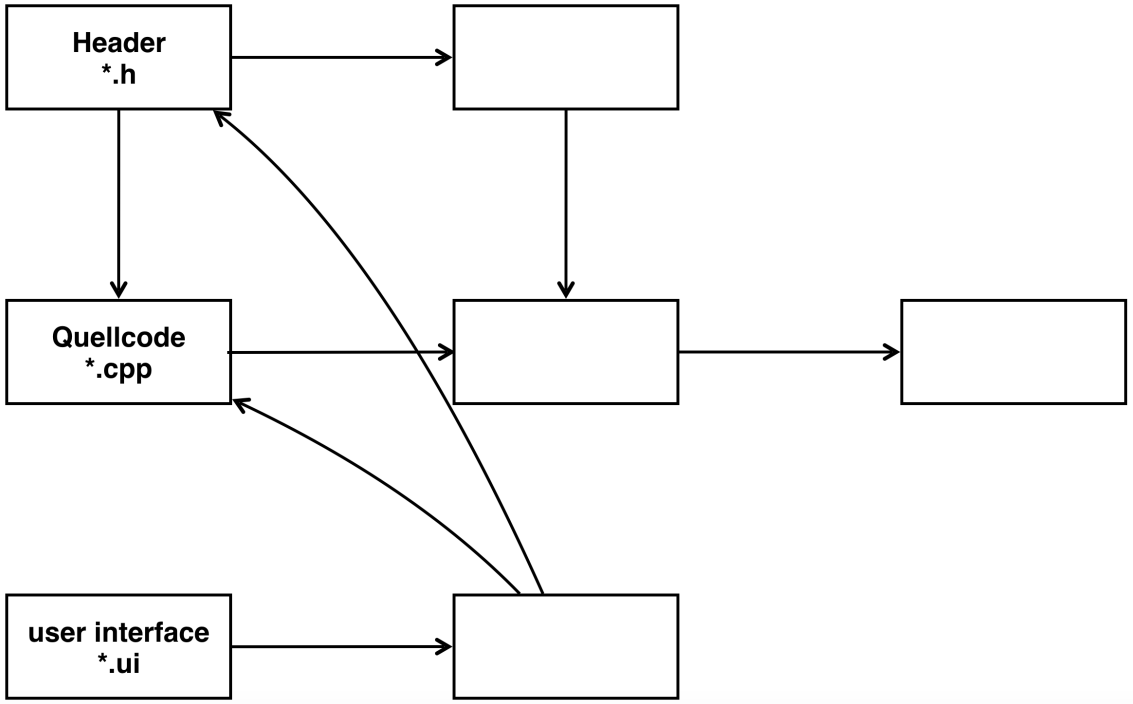
   1.

   2.

   3.

9. Wie wird ein Qt-Projekt kompiliert, das das Meta Object System verwendet *und* den UI-Designer einsetzt? Zeichen Sie den Build-Prozess als Diagramm. (4)

   Beschriften Sie im folgenden Diagramme **alle Pfeile** und füllen Sie die **leeren Kästchen** aus.



10. Nennen Sie zwei Vorteile, die das Konzept der Signals und Slots gegenüber klassischen C-Funktionszeigern (Callbacks) besitzt. (4)

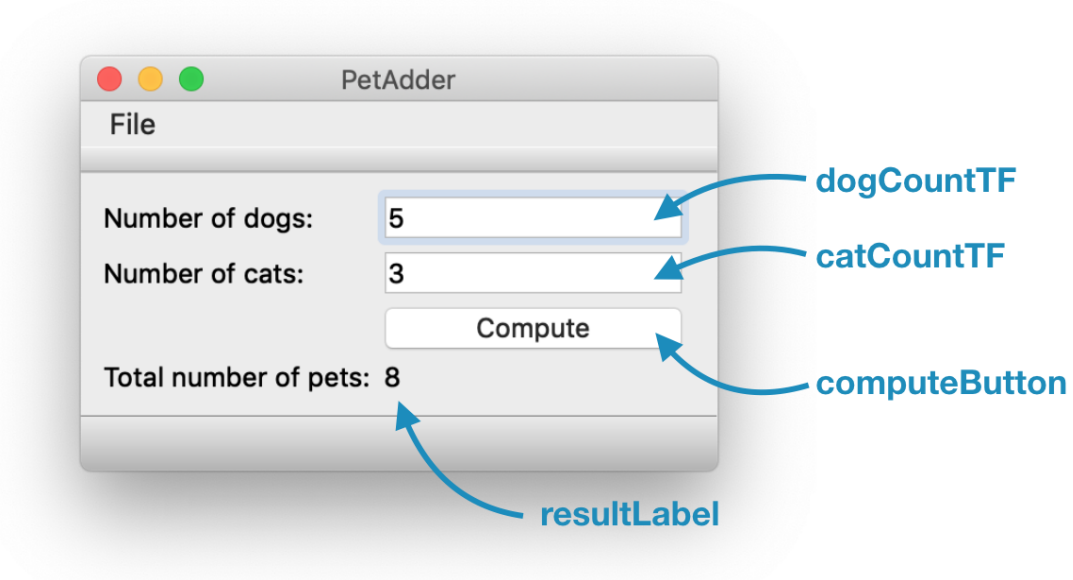
    Vorteil 1:

    Vorteil 2:

11. Was ist der Unterschied zwischen einem modalen und einem nicht-modalen Dialog? (2)

12. Sie sind der Chef-Entwickler der *Bankrupt Software AG* und sollen das Unternehmen vor der Insolvenz retten, indem Sie eine bahnbrechende GUI-Anwendung schreiben: das Programm PetAdder™. (15)

PetAdder™ erlaubt es einem Nutzer, die Anzahl seiner Hunde und Katzen einzugeben. Auf Knopfdruck wird dann die Summe aus beidem berechnet und angezeigt. Der UX-Designer K'Ratak hat schon mal die GUI gebaut:



Die Objektnamen der GUI-Elemente befinden sich ebenfalls in der Abbildung (blau).

Das Hauptfenster (Klasse MainWindow) soll folgendes können:

- Der Knopf "Compute" ist genau dann aktiviert, wenn in jedem Eingabefeld mindestens ein Zeichen eingegeben ist. Ist also mindestens ein Eingabefeld leer, ist der Knopf deaktiviert.
- Bei Klick auf "Compute" passiert folgendes:
  - Die Summe aus beiden Eingabefeldern `dogCountTF` und `catCountTF` wird berechnet und im Ergebnis-Label `resultLabel` ausgegeben.
  - Zusätzlich soll ein Signal `sumComputed` ausgelöst werden. Es erhält das Ergebnis als Parameter.

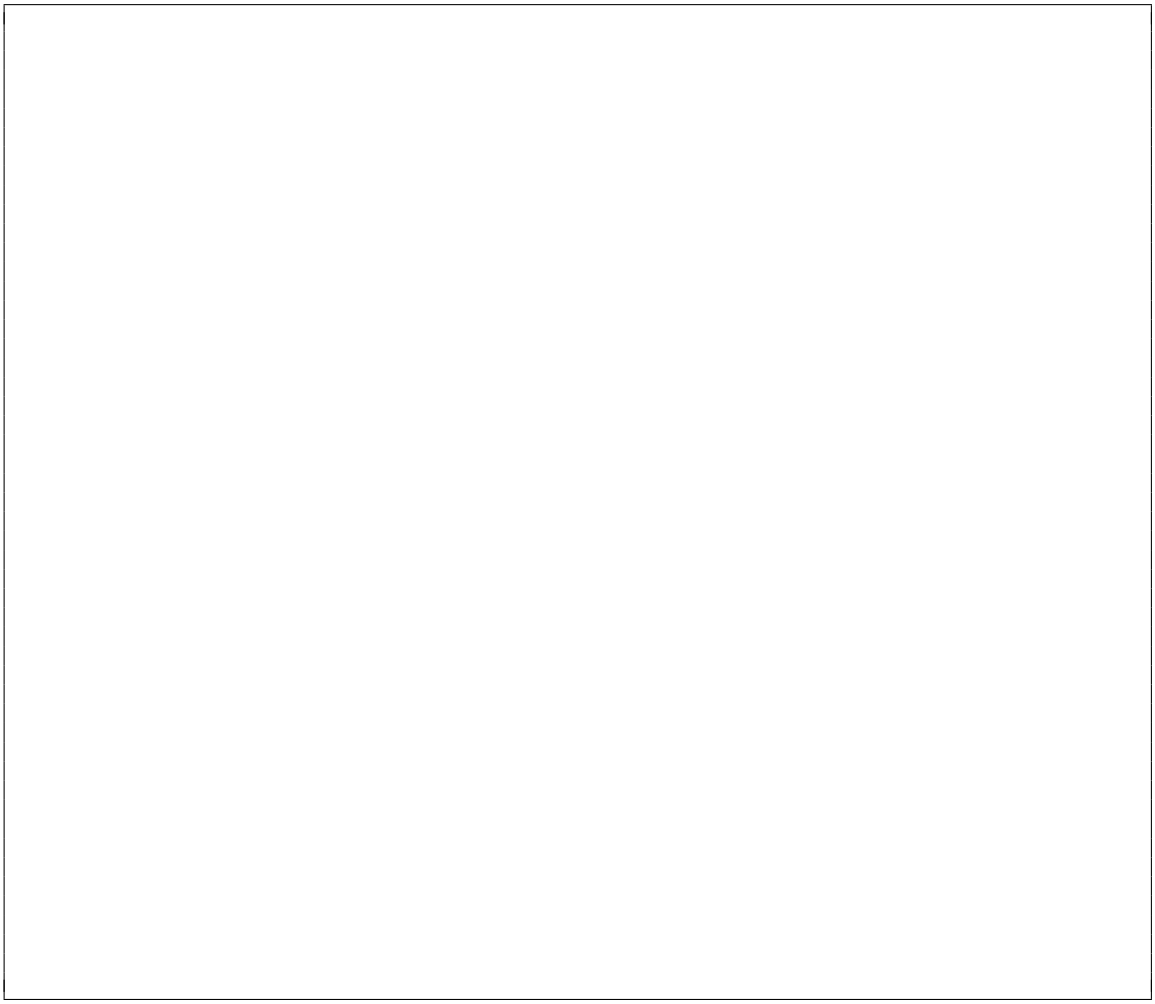
Hinweise:

- Beim Programmstart sind die beiden Eingabefelder leer.
- Sie können davon ausgehen, dass ein Nutzer immer korrekte Zahlen eingibt.
- Das Qt Cheat Sheet auf der letzten Seite enthält hilfreiche Informationen.

Ergänzen Sie die Dateien auf den nächsten Seiten.

Headerdatei mainwindow.h

```
#include <QMainWindow>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private:
    Ui::MainWindow *ui;
```



```
};
```

Sourcedatei mainwindow.cpp (Seite 1)

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

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

```
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

(Fortsetzung auf der nächsten Seite)

Sourcedatei mainwindow.cpp (Seite 2)

Nebenläufigkeit

13. (a) Wenn eine Anwendung den GUI-Thread blockiert, ist die Anwendung aus Sicht des Nutzers "eingefroren". Erklären Sie anhand der Event-Loop, was technisch dabei passiert. (3)

(b) Wie lässt sich die Blockierung des GUI-Threads prinzipiell vermeiden? (2)

14. (a) Beschreiben Sie die **beiden Arten**, mit dem sich in Qt nebenläufige Threads umsetzen lassen. (4)

(b) Wie beendet man einen Qt-Thread sauber? Wie lässt sich dieser Prozess sauber beschleunigen? (4)

15. Nehmen wir an, es gibt ein Programm, das Sensordaten verarbeitet (z.B. Temperaturdaten). Diese Sensordaten stammen aus 10 unterschiedlichen Quellen. Zwei Funktionen stehen zur Verfügung, um mit diesen Sensordaten zu arbeiten: (6)

    - `acquireNewSensorData()` liest neue Sensordaten aus den 10 Quellen und schreibt sie in ein Array. Diese Funktion wird einmal pro Sekunde von einem einzigen Thread aufgerufen.
    - `getAverageSensorData()` gibt den Durchschnittswert der Sensordaten zurück. Diese Funktion wird jede Sekunde von über 1000 Threads aufgerufen.

    Schreiben Sie den folgenden Code so um, dass keine Konflikte beim Zugriff auf die geteilte Sensordaten auftreten. Achten Sie auf eine effiziente Programmierung.

```
const int g_numChannels = 10;

float sensorData[g_numChannels];


float getAverageSensorData() {


    float sum = 0;

    for(int i = 0; i < g_numChannels; i++) {
        sum += sensorData[i];
    }


    return sum / g_numChannels;
}


void acquireNewSensorData() {


    for(int i = 0; i < g_numChannels; i++) {
        sensorData[i] = readSensorFromExternalSource(1);
    }


}
```

Bonusfrage (+2 Punkte)

16. Ein wichtiger Leitsatz der Mensch-Maschine-Interaktion lautet “Wir sind nicht die Nutzer”. Was ist damit gemeint? (2 (bonus))

Qt Cheat Sheet

Signale

QPushButton / QAbstractButton

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

QLineEdit

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

Methoden

QLineEdit

Text auslesen
QString text() const

Text setzen
void setText(const QString &)

QString

Länge ermitteln
int size() const

String in int umwandeln:
int toInt() const

String aus int erzeugen:
static QString number(int n)

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()
```