# Grafische Benutzeroberflächen: Einführung

Nun lernen wir grafische Benutzeroberflächen kennen, mit denen ein Benutzer ein Programm selbst steuern kann.

## Grundprinzip moderner Softwareentwicklung

Trennung von

* Benutzeroberfläche (GUI-Schicht)
* und Fachlichem (Fachkonzept-Schicht)

|  |  |  |
| --- | --- | --- |
| GUI-Schicht |  | Fachkonzept-Schicht |
| * Benutzerführung (Ein- und Ausgabe der Daten) * Präsentation der Daten |  | * Darstellung und Lösung der fachlichen Probleme * unabhängig von der GUI |

|  |  |  |
| --- | --- | --- |
| GUI-Klasse  MeinDialog |  | Fachklasse  Konto |
| beschreibt den Aufbau der GUI:   * Bezeichnungsfelder (Labels) * Textfelder * Befehlsschaltflächen (Buttons) * … |  |  |

|  |  |  |
| --- | --- | --- |
| Objekt der Klasse  MeinDialog |  | Objekt der Klasse  Konto |
|  |  |  |

## Klassenübersicht von Qt

Qt ist ein Plattform-übergreifendes Framework für verschiedenste Geräte (Desktop, mobil, eingebettet) und Plattformen (Linus, Windows, OS X, Android etc.). Qt selbst ist keine Programmiersprache, sondern ein in C++ geschriebenes Framework. Weitere Informationen sind unter <http://qt.io> zu finden.

Für unsere grafischen Benutzeroberflächen spielen Widgets eine zentrale Rolle. Ein Widget ist ein funktionales Element der GUI. Es hat einen klar definierten Funktionsumfang. Eine Übersicht über zur Verfügung stehende Widgets können der folgenden Klassenbibliothek entnommen werden:

Beschrif- tung

Text- eingabe

Schiebe-regler

Text-ausgabe

Schaltflächen

Dialog-

fenster

## Integration der GUI in das Python-Programm

Jedes Programm, das eine GUI enthält, besteht immer aus mindestens zwei Dateien. Speichern Sie alle Dateien in einen gemeinsamen Ordner.

* eine Datei, in der die GUI beschrieben ist (hier: *MeinDialog1.ui*). Nähere Informationen zu dieser Datei sind im *Infoblatt L4\_5 Info Eigene GUI erstellen.docx* zu finden.
* eine Python-Datei, die diese GUI-Datei einbindet und das Hauptprogramm beinhaltet (hier: L4\_1\_1\_GUI\_Einführung.py)
* Da wir objektorientiert arbeiten, ist meistens noch eine Datei mit der Fachklasse enthalten (hier: Konto.py)

Öffnen Sie zunächst das Programm L4\_1\_1\_GUI\_Einführung.py. Mit diesem Programm soll später ein Bankkunde sein Konto verwalten können. Zum aktuellen Zeitpunkt ist das Programm noch eine leere Hülle und tut noch nichts. Dies wird sich aber in den nächsten Schritten schnell ändern. Der Programmaufbau ist für alle Programme mit GUI der gleich. Wir betrachten kurz den Quellcode.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import sys  from PyQt5 import QtWidgets, uic  from Konto import \*  class MeinDialog(QtWidgets.QDialog):  def \_\_init\_\_(self, parent=None):  super().\_\_init\_\_(parent)  self.ui = uic.loadUi("MeinDialog1.ui", self)    app = QtWidgets.QApplication(sys.argv)  dialog = MeinDialog()  dialog.show()  sys.exit(app.exec\_()) |

Zeile 1: sys wird importiert

Zeile 2: Zwei benötigte Module werden importiert:

* In QtWidgets sind alle Widget-Klassen des Qt-Frameworks gekapselt.
* Mit Hilfe des Moduls uic kann die erstellte UI-Datei gelesen werden.

Zeile 3: Wird eine Fachklasse verwendet (hier: Konto), so müssen wir diese ebenfalls einbinden.

Zeile 5: Die Klasse MeinDialog wird definiert. Sie erbt die wesentlichen Eigenschaften von der Klasse QDialog.

Zeile 6: Der Konstruktor der Klasse MeinDialog wird definiert.

Zeile 7: Darin wird der Konstruktor der Oberklasse aufgerufen.

Zeile 8: Anschließend wird die Benutzeroberfläche aus der Datei MeinDialog1.ui geladen.

Zeile 10: Die App wird erstellt.

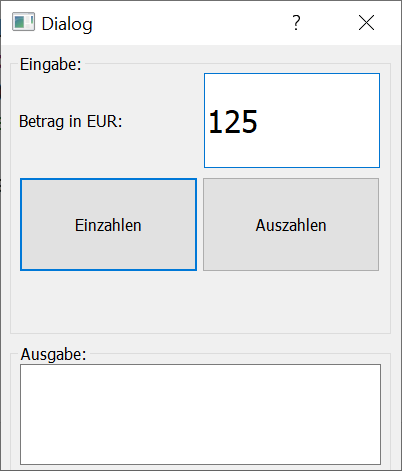
Zeile 11: Ein neues Dialogfenster wird deklariert und initialisiert.

Zeile 12: Das Dialogfenster wird angezeigt.

Zeile 13: Sobald der Dialog vom Benutzer geschlossen wird, wird das Programm beendet.

## Die ersten Widgets

Beim Start des Programms sehen wir bereits die erstellte GUI. In dieser sind bereits die ersten Widgets erkennbar. Wie solche Dialogfenster erstellt werden, werden wir in Einheit L4\_5 kennen lernen.



Schaltfläche

Klasse **QPushButton**

Es kann eine Aktion eingerichtet werden, die bei Klick auf die Schaltfläche durchgeführt wird.

Im Beispiel: *bt\_einzahlen*

Bezeichnungsfeld

Klasse **QLabel**

Hiermit werden Beschriftungen vorgenommen.

Im Beispiel: *lb\_betrag*

Texteingabefeld

Klasse **QLineEdit**

Hiermit kann kurzer Text eingelesen werden.

Im Beispiel: tf\_betrag

Textausgabefeld

Klasse **QPlainTextEdit**

Hiermit kann Text ausgegeben werden.

Im Beispiel: ta\_ausgabe

## Wie erstelle ich ein Objekt und wie gebe ich Text aus?

Öffnen Sie nun das Programm L4\_1\_2\_GUI\_Einführung.py. Wir sehen eine erste Ausgabe. Dabei werden die Informationen aus einem Objekt der Klasse Konto ausgelesen.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class MeinDialog(QtWidgets.QDialog):  def \_\_init\_\_(self, parent=None):  super().\_\_init\_\_(parent)  self.ui = uic.loadUi("MeinDialog.ui", self)  # Objekt anlegen  global k  k = Konto(66124457, 450.00, "Peter Schmidt")  self.ui.ta\_ausgabe.setPlainText("Hallo " + k.inhaber + "!")  self.ui.ta\_ausgabe.appendPlainText("Ihre Kontonummer: " + str(k.kontonummer))  self.ui.ta\_ausgabe.appendPlainText("Aktueller Kontostand: " + str(k.kontostand) + " Euro") |

Zeile 6: Es wird zunächst eine Variable k deklariert. Durch das Schlüsselwort global kennzeichnen wir, dass diese Variable überall sichtbar sein soll.

Zeile 7: Die Variable k wird als Objekt der Klasse *Konto* mit der Kontonummer 66124457, dem Kontostand von 450,00€ und dem Inhaber Peter Schmidt initialisiert.

Zeile 8-10: Mit self.ui können wir auf unsere GUI zugreifen. Darin greifen wir auf das Objekt ta\_ausgabe, also unser Textausgabefeld zu. Dieses Textausgabefeld stellt unter anderem zwei Methoden zur Verfügung:

setPlainText(…): Leert eventuell vorhandene Ausgabe und schreibt neuen Inhalt

appendPlainText(…): Fügt neuen Inhalt zur vorhandenen Ausgabe hinzu

Bitte beachten: Diese Methode akzeptieren nur Zeichenketten. Daher müssen Zahlen stets mit der Methode str(…) in Zeichenketten umgewandelt werden.

## Wie weise ich einer Schaltfläche eine Reaktion zu und wie lese ich Text ein?

Öffnen Sie nun das Programm L4\_1\_3\_GUI\_Einführung.py und testen Sie es. Endlich passiert etwas. Wir können einen Geldbetrag eingeben und auf eine Schaltfläche klicken. Der Geldbetrag wird dann je nach Schaltfläche ein- oder ausbezahlt. Die Benutzersteuerung erfolgt nach dem Konzept von Signalen und Slots.

**Signale und Slots**

Signale sind „Botschaften“, die abgegeben werden, sobald ein bestimmtes Ereignis eintritt. Ein Slot ist eine Funktion, die auf eine bestimmte Art mit einem Signal verknüpft werden kann. Slots und Signale „wissen“ zunächst nichts voneinander. Erst durch die Verknüpfung entsteht die eigentliche Programmlogik: Jedes Mal, wenn das Signal abgegeben wird, wird anschließend der verbundene Slot aufgerufen. Ein Signal kann auch mit mehreren Slots verbunden werden, so dass bei Eintreten eines Ereignisses mehrere Funktionen aufgerufen werden. Genauso kann ein Slot mit mehreren Signalen verbunden werden, wodurch dieselbe Funktion bei Auftreten unterschiedlicher Ereignisse aufgerufen wird.

**Verknüpfung von Signal und Slots**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | def \_\_init\_\_(self, parent=None):  super().\_\_init\_\_(parent)  self.ui = uic.loadUi("MeinDialog.ui", self)  ...  # Slot einrichten  self.ui.bt\_einzahlen.clicked.connect(self.on\_einzahlen)  self.ui.bt\_auszahlen.clicked.connect(self.on\_auszahlen)  def on\_einzahlen(self):  ...  def on\_auszahlen(self):  ... |

Zeilen 6-7: Mit self.ui können wir auf unsere GUI zugreifen. Darin greifen wir auf die Objekte bt\_einzahlen und bt\_auszahlen, also unsere beiden Schaltflächen zu. Und darin wiederum auf das Signal clicked. Mit der Methode connect() wird also das Signal clicked mit der Reaktion, also dem Slot on\_einzahlen() bzw. on\_auszahlen() verbunden.

Zeilen 9-10: Hier wird der Slot on\_einzahlen() definiert. Alles, was später im Rumpf dieser Methode steht, wird später bei Klick auf den Button „Einzahlen“ ausgeführt.

Zeilen 12-13: Hier wird der Slot on\_auszahlen() definiert. Alles, was später im Rumpf dieser Methode steht, wird später bei Klick auf den Button „Auszahlen“ ausgeführt.

**Definition des Slots on\_einzahlen()**

|  |  |
| --- | --- |
| 1  2  3  4  5 | def on\_einzahlen(self):  betrag = float(self.ui.tf\_betrag.text())  k.einzahlen(betrag)  self.ui.ta\_ausgabe.appendPlainText("Es wurden: " + str(betrag) + " Euro eingezahlt.")  self.ui.ta\_ausgabe.appendPlainText("Neuer Kontostand: "+ str(k.kontostand) + " Euro") |

Zeile 1: Der Slot, also die Funktion on\_einzahlen() wird definiert.

Zeile 2: Mit self.ui greifen wir auf unsere GUI zu. Darin greifen wir auf die Objekte tf\_betrag zu, also unser Texteingabefeld. Mit der Methode text() können wir den Geldbetrag einlesen. Dieser wird dann in der Variablen betrag gespeichert. Bitte beachten: Wir wollen eine Dezimalzahl einlesen, daher müssen wir die eingelesene Zeichenkette mit float() umwandeln.

Zeilen 3-5: Wir rufen die Methode einzahlen() für das Objekt k auf und erstellen wie gewohnt unsere Ausgabe.

**Definition des Slots on\_auszahlen()**

|  |  |
| --- | --- |
| 1  2  3  4  5 | def on\_auszahlen(self):  betrag = float(self.ui.tf\_betrag.text())  k.auszahlen(betrag)  self.ui.ta\_ausgabe.appendPlainText("Es wurden: " + str(betrag) + " Euro ausgezahlt.")  self.ui.ta\_ausgabe.appendPlainText("Neuer Kontostand: "+ str(k.kontostand) + " Euro") |

Zeile 1: Der Slot, also die Funktion on\_auszahlen() wird definiert.

Zeile 2: Mit self.ui greifen wir auf unsere GUI zu. Darin greifen wir auf die Objekte tf\_betrag zu, also unser Texteingabefeld. Mit der Methode text() können wir den Geldbetrag einlesen. Dieser wird dann in der Variablen betrag gespeichert. Bitte beachten: Wir wollen eine Dezimalzahl einlesen, daher müssen wir die eingelesene Zeichenkette mit float() umwandeln.

Zeilen 3-5: Wir rufen die Methode auszahlen() für das Objekt k auf und erstellen wie gewohnt unsere Ausgabe.