Johannes Gutenberg-Universität Mainz
Institut für Informatik
Einführung in die Softwareentwicklung
Prof. Dr. Michael Wand
Dr. Christian Ali Mehmeti-Göpel
Visual Computing Group

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Einführung in die Softwareentwicklung

Übungsblatt 01: Einrichten der Systemumgebung und Benutzung der Programmiersprachen

15. April 2025

Über dieses Übungsblatt

Dieses Übungsblatt dient zur Vorbereitung der Programmierumgebung für die Übungen. Wir werden alle vier in der Vorlesung relevanten Programmiersprachen installieren (Python, JAVA, C++ und SCALA) und eine kleine Aufgabe in jeder der vier Programmiersprachen lösen. Damit es spannend bleibt, messen wir am Ende die Laufzeitunterschiede.

Hinweise zur Umgebung

Man kann alle vier Programmiersprachen auf allen gängigen Betriebssystemen (Unix-Varianten, Windows, MacOS) installieren. Ich zeige auf diesem Übungsblatt als Beispiel die Installation auf dem aktuellen Ubuntu Linux (24.04 LTS), da es hier durch den eingebauten Paketmanager besonders einfach zu erklären ist. Für Neueinsteiger/innen mit wenig Erfahrung kann es sinnvoll sein, dieses Betriebssystem zu benutzen, z.B. in einer virtuellen Maschine. Die Installation von Ubuntu in z.B. VirtualBox (frei verfügbare VM mit schönem, einfachen GUI; empfehlenswert – aber, Achtung, es gibt eine Erweiterungen, die wir nicht brauchen, die kostenpflichtig sein kann) ist auf den meisten Systemen sehr einfach (einfach googlen – im Wesentlichen lädt man das „ISO"-Abbild der Installations-DVD von Ubuntu runter, startet damit die VM, und folgt den Anweisungen am Bildschirm); wichtig ist nur, dass der eingesetzte Computer genug Arbeitsspeicher hat. Empfehlenswert ist, der VM genug RAM, Video-RAM und Cores zuzuordnen sowie die „Guest-Extensions" zu installieren.

Abgabe: (Abgabe 4. SW, Beratung 2.+3.SW)

Das Übungsblatt muss bis zum **04. Mai 2025, 23:59h** in LMS abgegeben werden. Laden Sie dazu den Source-Code von Aufgabe 2/3 hoch. Aufgabe 1 muss nicht abgegeben und nicht präsentiert werden. Die Ergebnisse müssen in den Übungen zwischen dem 5.5-9.5.2025 (4. SW) vorgestellt werden. In den Übungen vom 21.4-2.5.2025 (2-3. Woche) gibt es Gelegenheit für Diskussionen und Tipps.

Aufgabe 1: Installieren Sie die Tools für die folgenden Programmiersprachen auf Ihrem Rechner

- **C und C++**
- **Python 3** mit MyPy.
- **JAVA**
- **SCALA 3** (benötigt eine JAVA Umgebung)

Außerdem sollten Sie die notwendigen Tools für die Entwicklung installieren. Sie brauchen jedem Fall einen Texteditor; ich empfehle Ihnen aber, eine richtige integrierte Entwicklungsumgebung („integrated development environment, IDE") zu installieren. Eine einfach zu nutzende Umgebung, die alle vier Sprachen (halbwegs) unterstützt, wäre z.B. Visual Studio Code. Für JAVA und SCALA werden oft die Werkzeuge von JetBrains (Intelli-J) empfohlen.

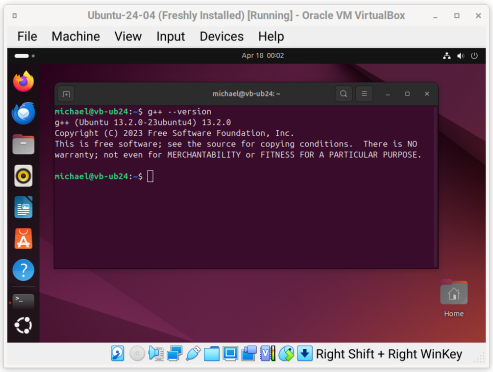
Bewertung: – *keine Punkte* –

Anleitung zu Aufgabe 1 (Beispielinstallation auf Ubuntu 24.04)

C bzw. C++

Als Compiler empfehlen wir auf allen Plattformen `gcc` / `g++` oder `clang`; auf Windows ist auch der Microsoft Visual C++-Compiler verfügbar. Als Version der Sprache ist C++ ab C++11 oder höher geeignet, also alle modernen Varianten (wir werden keine komplexen Features benutzen). Alle diese Tools werden in vollständigen Paketen bereitgestellt, die auch die notwendigen Systembibliotheken und den Linker, und was man sonst noch so alles braucht, enthalten.

Tipp: eine Umfangreiche Doku findet sich in den Unterlagen zu EIS aus früheren Semestern, siehe hier: https://luna.informatik.uni-mainz.de/eis21/cpp-tips.md



Ubuntu-Installation von gcc/g++:

```
sudo apt update
sudo apt install build-essential
```

Installation testen z.B. via `g++ --version`

Compiler-Aufruf: `g++ MeinProgram.cpp`

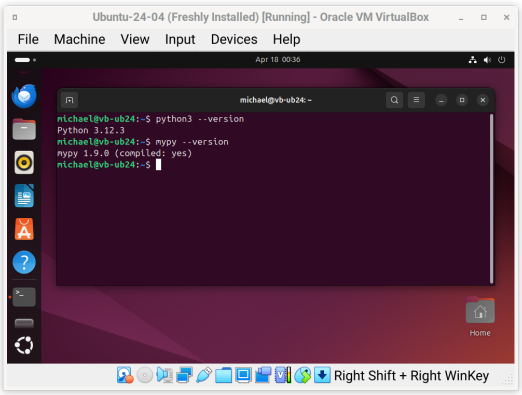
Ausgabe starten mit: `./a.out`
(die Datei `a.out` im aktuellen Verzeichnis enthält das übersetzte Programm)

Hinweis: C und C++ werden in EIS im Sommersemester 2025 nur eine kleine Rolle spielen. Sie benötigen aber trotzdem eine funktionierende C++-Umgebung, um alle Aufgaben bearbeiten zu können.

Python 3 und MyPy Installieren

Für Python gibt es fertige Installationspakete für alle gängigen Plattformen. Wir empfehlen die neuste stabile Version (derzeit 3.13). MyPy kann dann in der Regel über PIP („`pip install mypy`“) nachinstalliert werden (nur Ubuntu möchte hier lieber, dass man den eingebauten Paketmanager benutzt, zumindest solange man keine „virtuellen Evinroments“ nutzt).

Ubuntu-Installation von Python 3 mit MyPy




```
sudo apt update
sudo apt install python3-dev  (…-dev ist wichtig)
sudo apt install mypy
```

Installation testen z.B. via

```
python3 --version
mypy --version
```

Typchecker-Aufruf: `mypy mein_program.py` (optional)

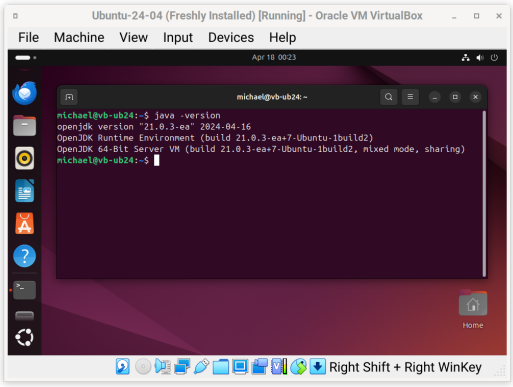
Program starten mit: `python3 mein_programm.py` (keine Übersetzung nötig)

JAVA

Auch wenn die Programmiersprache JAVA nur eine untergeordnete Rolle in EIS spielen wird, ist eine JAVA-Umgebung notwendig, um mit SCALA arbeiten zu können. Da die Sprachen eng aufeinander aufbauen, ist ein Grundverständnis von JAVA auch durchaus nützlich. Daher probieren wir es hier auch gleich mal aus.

Auf den meisten Platformen installieren sie die virtuelle Maschine als „JAVA Runtime Environment (JRE)“ und die Entwicklungswerkzeuge (Compiler etc.) als „JAVA Development Toolkit (JDK)“

Ubuntu-Installation des JAVA JRE+JDKs




```
sudo apt update
sudo apt install default-jre
sudo apt install default-jdk
```

Installation testen z.B. via `java -version`

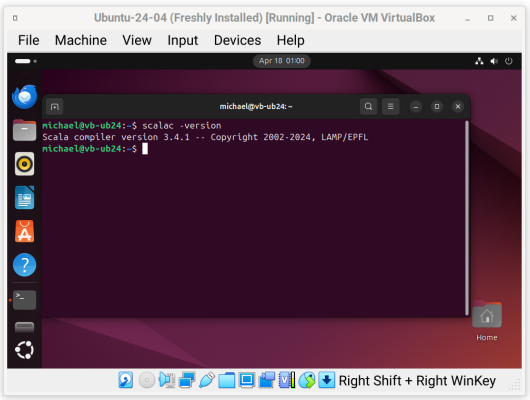
Compiler-Aufruf: `javac MeinProgram.java`

Ausgabe starten mit: `java MeinProgramm.class`

(Die `.class`-Datei enthält den übersetzten JVM-Code, „`java`“ startet die JVM. Die Endungen kann man weglassen bei `javac` und `java` auch weglassen.)

SCALA 3

Nachdem JAVA erfolgreich installiert wurde, ist es einfach, Scala nachzuinstallieren. Auch hier gibt es Installer-Pakete, z.B. für Windows oder Ubuntu. Bei Ubuntu ist das eingebaute „`scala`"-Paket allerdings noch auf dem Stand von Scala 2. Für Scala 3 muss man daher etwas zusätzlichen Aufwand betreiben.



Ubuntu-Installation der Scala Umgebung

```
sudo apt update
sudo apt install curl
```

Danach etwas unkonventionell (nach Empfehlung von https://docs.scala-lang.org/getting-started/)

```
curl -fL https://github.com/coursier/coursier/releases/latest/download/cs-x86_64-pc-linux.gz | gzip -d > cs
chmod +x cs
./cs setup
```

(Dies sind drei separate Zeilen, jeweils mit return am Ende; man kann die auch mit „**&&**" aneinanderhängen und nacheinanderausführen, wie auf der angegebenen Webseite gemacht.)

Installation testen z.B. via `scalac -version` (vorher Rechner bzw. VM neu starten!)

Compiler-Aufruf: `scalac MeinProgram.scala`

Ausgabe starten mit: `scala MeinProgramm`

(„`scala`" startet die JVM mit zusätzlichen Konfigurationen für die Ausführung von SCALA-Programmen. Die Endung `.class` entfällt beim Aufruf von `scala`.)

Installation einer IDE

Installieren Sie am besten eine geeignete IDE für Ihre Arbeit. Eine gute „Universallösung" ist z.B. Visual Studio Code. In Ubuntu kann man das recht einfach via

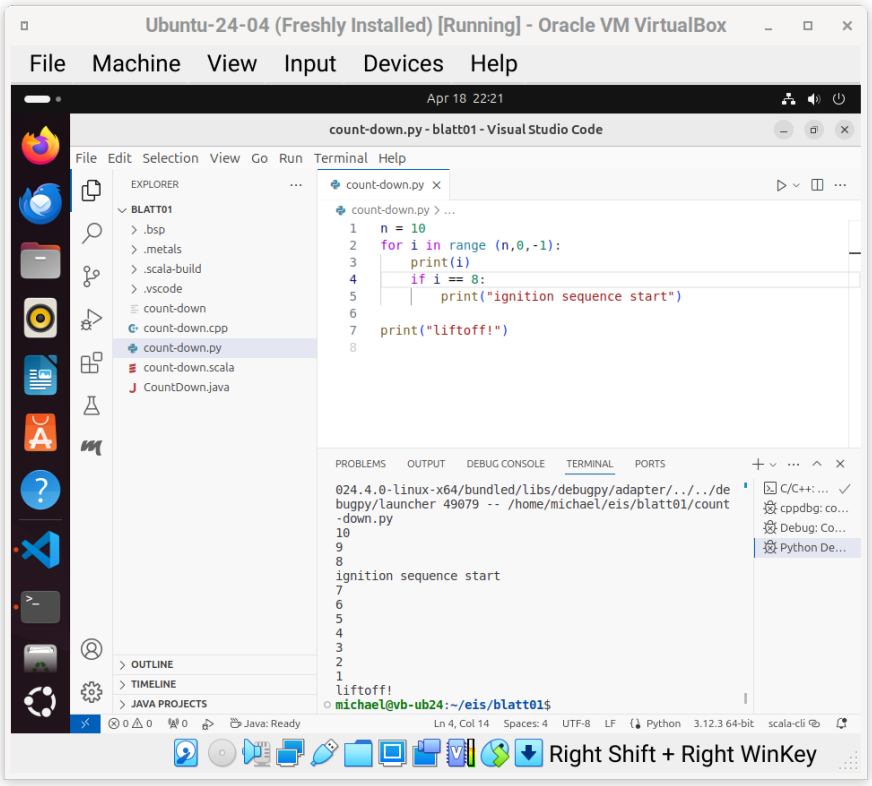
```
sudo snap install --classic code
```

erledigen. Für die verschiedenen Programmiersprachen gibt es Plugins, die bei der Entwicklung helfen und die i.d.R. automatisch zur Installation vorgeschlagen werden, sobald man z.B. eine Python oder C++-Datei anlegt. Für Scala empfiehlt sich das Scala-„Metals" Plugin (ggf. manuell auswählen). Auch Versionsverwaltung via git (oder anderer VCS/DVCS) wird per Plugin erleichtert.

Es gibt aber auch viele Alternativen, wie z.B. die Produkte von JetBrains, QTCreator für C++ und m.E. Python, oder das kommerzielle „Visual Studio" von Microsoft. Es gibt auch viele alternativen Editoren wie Sublime, Atom, Notepad++ oder EMACS / VI / VIM (für Hartgesottene **:-)**).

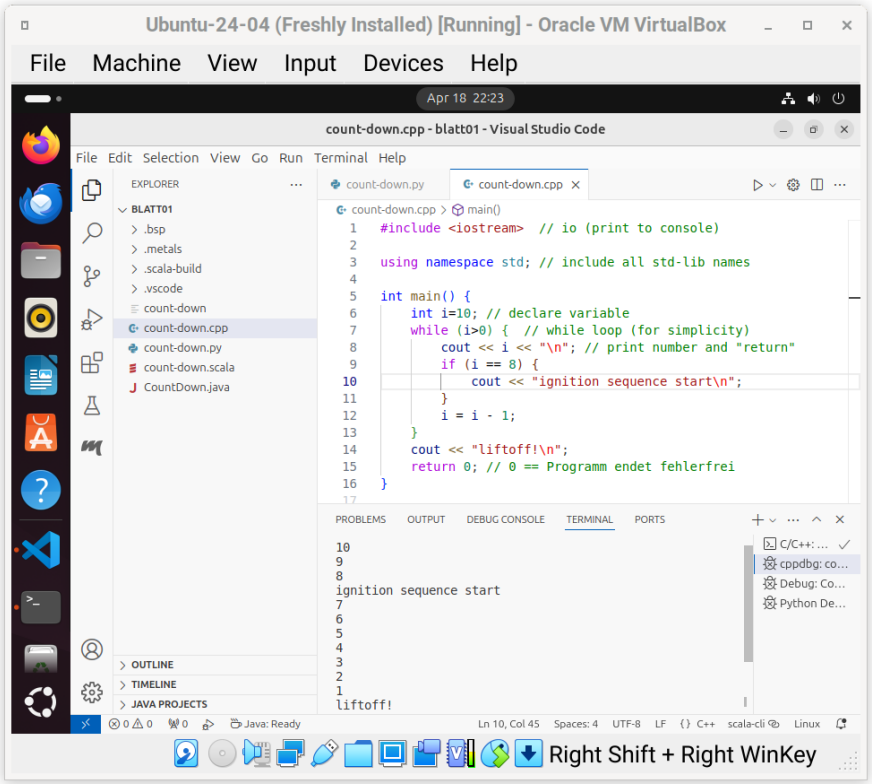
Aufgabe 2: Ein einfaches Programm

a) Zum Einstieg testen Sie das folgende einfache Programm in allen vier Sprachen, welches einen Count-Down auf der Konsole ausgibt. Diese Aufgabe braucht nicht mit abgegeben / präsentiert werden; es dient als Vorlage für Aufgabe b.



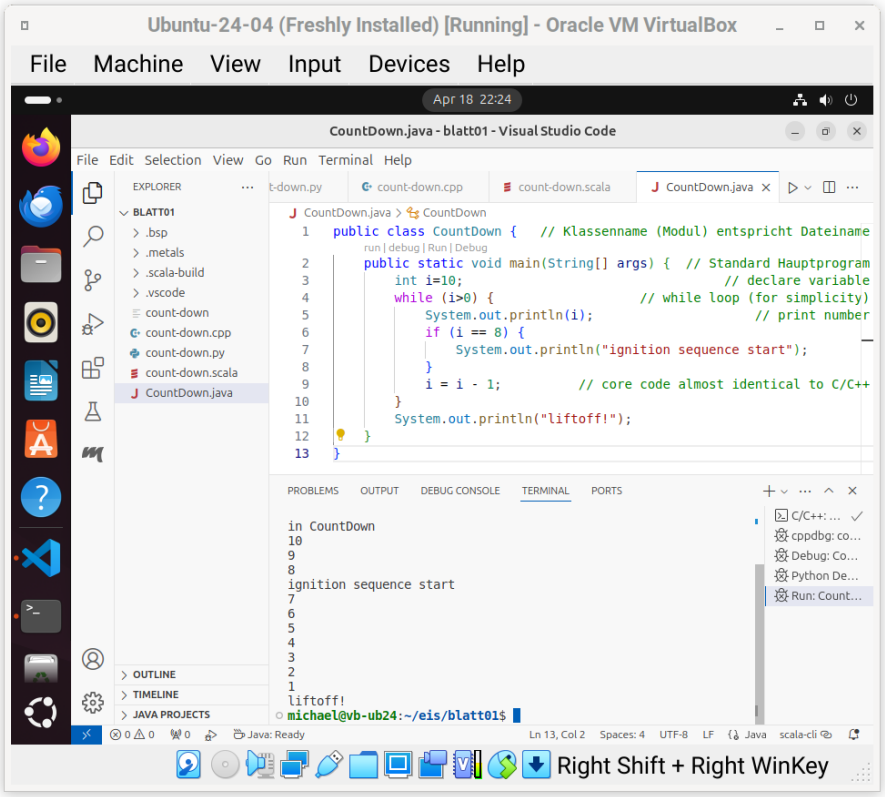
Count-Down in Python

```python
n = 10
for i in range (n,0,-1):
    print(i)
    if i == 8:
        print("ignition sequence start")

print("liftoff!")
```



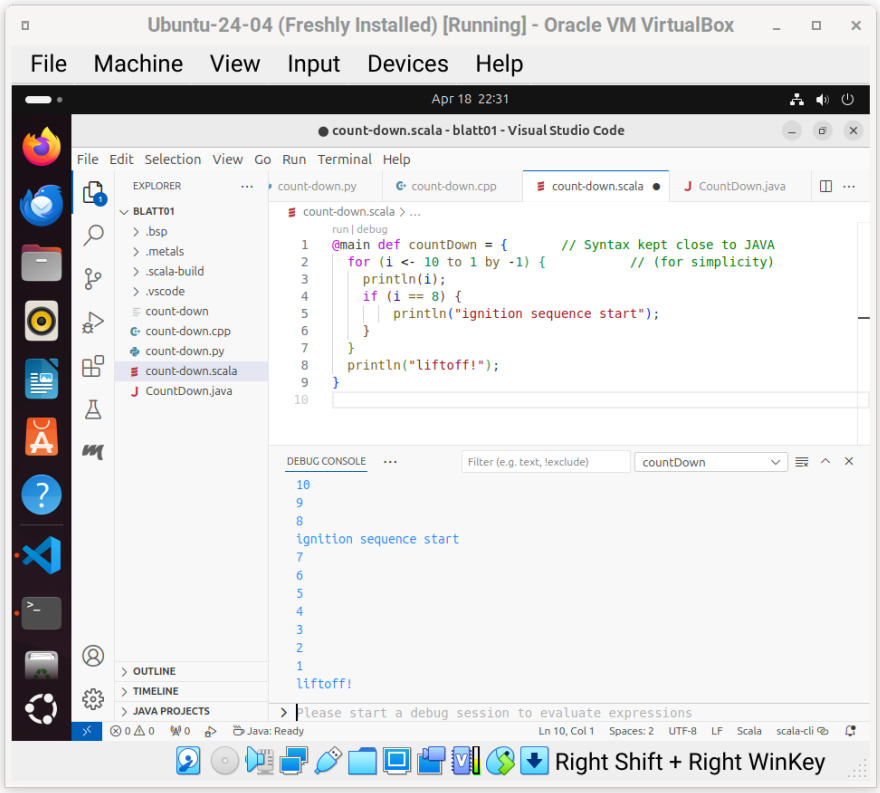
Count-Down in C++

```cpp
#include <iostream>  // io (print to console)

using namespace std; // include all std-lib names

int main() {
    int i=10; // declare variable
    while (i>0) {  // while loop (for simplicity)
        cout << i << "\n"; // print number and "return"
        if (i == 8) {
            cout << "ignition sequence start\n";
        }
        i = i - 1;
    }
    cout << "liftoff!\n";
    return 0; // 0 == Programm endet fehlerfrei
}
```



Count-Down in JAVA

```java
public class CountDown {   // Klassenname (Modul) entspricht Dateiname
    public static void main(String[] args) {  // Standard Hauptprogram
        int i=10;                                    // declare variable
        while (i>0) {                       // while loop (for simplicity)
            System.out.println(i);                    // print number
            if (i == 8) {
                System.out.println("ignition sequence start");
            }
            i = i - 1;              // note: code very similar to C/C++
        }
        System.out.println("liftoff!");
    }
}
```



Count-Down in Scala

```
@main def countDown = {        // Syntax kept close to JAVA
  for (i <- 10 to 1 by -1) {            // (for simplicity)
    println(i);
    if (i == 8) {
        println("ignition sequence start");
    }
  }
  println("liftoff!");
}
```

Anmerkung: Man darf in Scala viele Klammern und Semikola auch weglassen; wir machen es hier genauso wie in C/C++/JAVA um die Sache an dieser Stelle zu vereinfachen.

b) Programmieren Sie in allen vier Programmiersprachen ein einfaches Programm, dass testet, ob eine ganze Zahl n eine Primzahl ist. Die Zahl n kann dabei im Quelltext fest kodiert sein. Testen Sie Ihre Lösung auf der Zahl $10\,007 \cdot 100\,003$ (Hinweis: die beiden fünf und sechsstelligen Faktoren sind jeweils Primzahlen, das Produkt damit nicht. Außerdem überschreitet das Produkt nicht die max. Größe des Standard 32Bit Ganzzahltyps in C/Java, was das Programmieren vereinfacht).

Als Algorithmus sollten Sie die naive Lösung benutzen, die einfach alle möglichen Teiler zwischen $2 \ldots \lfloor\sqrt{n}\rfloor$ ausprobiert und abbricht, falls ein Teiler gefunden wurde.

Messen Sie jeweils, wie lange die Ausführung des Programms dauert. Auf Linux-Systemen kann man dies z.B. machen, indem man einem Befehl auf der Kommandozeile den Befehl „time“ voranstellt, z.B. `time python3 count-down.py`

Hier sind übrigens meine Messungen für den einfachen „Count-Down“ (man sieht in dem Beispiel eher den Überhang für das Laden der Laufzeitumgebung) auf einem Linux System in einer VirtualBox-VM auf einem intel Skylake i7-6700K 4GHz Prozessor:

	C++	Python	JAVA	SCALA
Count Down	4 ms	42 ms	42 ms	3590 ms

Beim Primzahltest sah die Liste deutlich anders aus (wird aber noch nicht verraten)…

c) Schreiben Sie das gleiche Programm wie in Aufgabe b, aber nutzen Sie nun Rekursion um die Schleife über die Teiler zu programmieren. Testen Sie Ihren Code mit kleinen Zahlen (z.B. 21) und großen Zahle (wie zuvor) als Eingabe. Was beobachten Sie in Bezug auf Laufzeit? Können Sie größere Zahlen testen, ohne dass der Stack überläuft? Berichten Sie über Ihre Erfahrungen in der Übung (es ist zu erwarten, dass nicht alle Programme mit größeren Zahlen funktionieren).

Bewertung: 0+50+20 = 70 Punkte

- *Teil (a) unbewertet,*
- *Teil (b) 4x15 Punkte für die Implementation und 10 Punkte für den Vergleich*
- *Teil (c) 4x5 Punkte für die verschiedenen Sprachen. Wenn die Implementation nur für kleine Zahlen funktioniert (wie tlw. zu erwarten), gibt es trotzdem die volle Punktzahl*

Aufgabe 3: Ein sehr einfaches GUI

Zum Schluss installieren wir noch eine GUI-Bibliotheken für Python – in JAVA/SCALA ist das schon im Standardinventar eingebaut (man kann das auch noch für C++ machen, wenn man möchte). Dann schreiben wir damit in Python wie auch in SCALA je ein Programm mit zwei Knöpfen nach Vorlage, und erweitern es selbst um einen dritten Knopf.

a) Installieren Sie das Packet PySide6 für Python3.

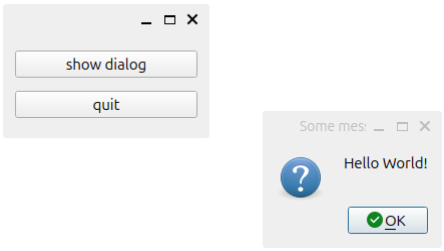
Anleitung: Auf den meisten Systemen geht das ganz einfach mit „`pip install PySide6`“. Lediglich die neueren Ubuntu Versionen mögen es lieber, wenn man alles mit apt installiert:

```
sudo apt update
sudo apt install python3-pyside6.qtcore
sudo apt install python3-pyside6.qtgui
sudo apt install python3-pyside6.qtwidgets
sudo apt install python3-pyside6.qtuitools
```

Tipps / Hinweise: Man kann auch noch mehr Pakete hinzufügen, wenn man möchte. Mit virtualenvironments kann man die Sachen auch direkt mit PIP installieren (z.B. recht einfach mit PyCharm umsetzbar); dann ist es nur ein PIP-Kommando.

b) Probieren Sie die folgenden beiden Code-Schnipsel aus, um eine einfache Anwendung zu erzeugen. Diese hat zwei Knöpfe. Der erste öffnet ein Dialogfenster mit einer Meldung, der zweite beendet das Programm.

Minimale Qt-GUI-Anwendung mit Python 3 + PySide 6 (Qt 6)



(Ergebnis der Aufgabe für PySide6 / Qt für Python)

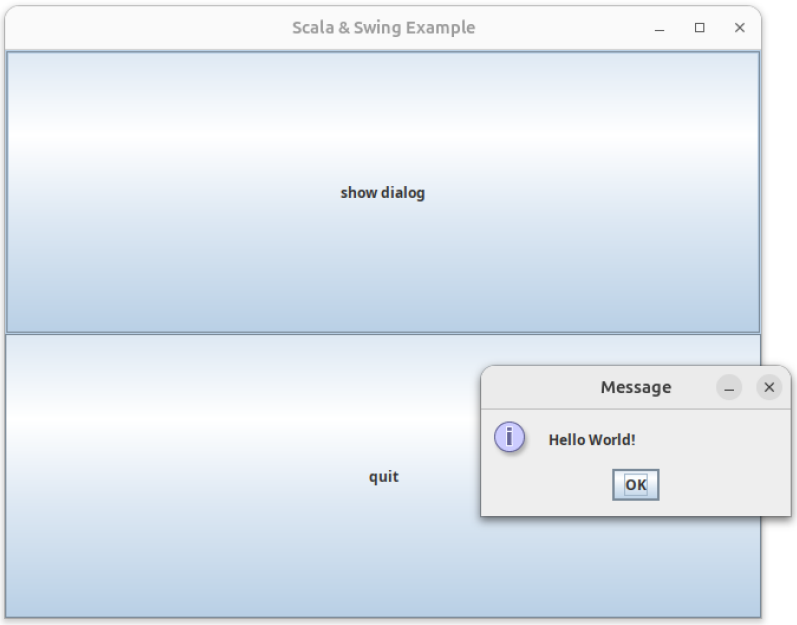
```
# import all the widgets
from PySide6.QtWidgets import QApplication, QPushButton
from PySide6.QtWidgets import QMainWindow, QVBoxLayout, QFrame, QMessageBox

# only needed for command line arguments
import sys
```

```
# we define our own window type to build a custom UI
class MyWindow(QMainWindow):

    def quit_app(self):
        # closing the last (and only) window ends the application
        self.close()

    def show_dialog(self):
        # open a simple dialog window to say hello
        QMessageBox.question(self, "Some message...", "Hello World!", QMessageBox.Ok)

    def __init__(self, parent):
        # call parent constructor
        super().__init__(parent)

        # a bit of housekeeping...

        # set a frame object (empty container) that fills the whole window
        self.frame = QFrame(self)
        # make this the content of the main window
        self.setCentralWidget(self.frame)
        # create a layout object
        self.my_layout = QVBoxLayout(self.frame)
        # tell the frame (just created) to use this layout
        self.frame.setLayout(self.my_layout)

        # make a button!

        # create a button, remember in a member variable
        self.btn_dialog = QPushButton("show dialog")
        # connect button to method of self object
        self.btn_dialog.clicked.connect(self.show_dialog)
        # add the button to the UI via the layout object
        self.my_layout.addWidget(self.btn_dialog)

        # same procedure again...
        self.btn_quit = QPushButton("quit")
        self.btn_quit.clicked.connect(self.quit_app)
        self.my_layout.addWidget(self.btn_quit)
```

Anmerkung: Alle Source-Code-Beispiele für dieses Übungsblatt liegen auch in einer ZIP-Datei „Blatt01-Beispielcode.zip“ im LMS bereit.

```
# our main program starts here, Python-style
if __name__ == "__main__":

    # create an application object (needs cmd-line arguments)
    app: QApplication = QApplication(sys.argv)

    # Create the main window.
    main_window: MyWindow = MyWindow(None)
    main_window.show()

    # Start the event loop.
    # Ends only after closing the main window
    app.exec_()
```

Minimale AWT/SWING-GUI-Anwendung mit Scala



(Ergebnis der Aufgabe für SWING / Scala)

```
// get all the AWT/SWING components we need
import java.awt.BorderLayout
import java.awt.Dimension
import java.awt.GridLayout
import javax.swing.JFrame
import javax.swing.JPanel
import javax.swing.JButton
import javax.swing.WindowConstants
import javax.swing.JOptionPane
```

```
object SwingHellowWorld extends App { // a fancy way of creating an application
    // some member functions, for later use
    def showSomeDialog(): Unit = {
        JOptionPane.showMessageDialog(null, "Hello World!");
    }
    def doQuit(): Unit = {
        mainWindow.setVisible(false)
        mainWindow.dispose()
    }

    // member variables to remember all the components we need
    // first, two new buttons
    var buttonDialog = new JButton("show dialog")
    var buttonQuit = new JButton("quit")
    // then, a panel, where the buttons are put into
    var panel = new JPanel()
    // and a layout, for the panel to arrange the buttons
    var layout = new GridLayout(2,1)
    panel.setLayout(layout)

    // we now add the buttons to the panel
    panel.add(buttonDialog)
    panel.add(buttonQuit)

    // we connect member functions as events to be called to the buttons
    buttonDialog.addActionListener(e => showSomeDialog())
    buttonQuit.addActionListener(e => doQuit())

    // now we create the main window (simple window via type "JFrame")
    var mainWindow = new JFrame("Scala & Swing Example")
    // it also needs a layout
    mainWindow.getContentPane.add(panel, BorderLayout.CENTER)
    // when it is closed, the whole app should quit
    mainWindow.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)
    // set some default size, why not VGA?
    mainWindow.setSize(new Dimension(640, 480))
    // center the new window on the screen
    mainWindow.setLocationRelativeTo(null)
    // show the window
    mainWindow.setVisible(true)

}
```

c) Erweitern Sie Ihre Lösung so, dass ein dritter Button erscheint, der die Zahl „42“ auf der Konsole ausgibt.

Optional (unbewertet): Falls Sie Lust haben, experimentieren Sie ruhig etwas mit anderen Widgets, wie z.B. Menüleisten u.ä.

Hinweis: Wir empfehlen für Neueinsteiger in den folgenden Übungsblättern, wo immer freie Wahl der Sprache besteht, Python + PySide6 für die GUIs zu benutzen, da hier mehr Voraussetzungen aus EIP bekannt sind. Wiederholer können vielleicht eher zwischen Scala+AWT/SWING und Python+PySide6 wählen.

Bewertung: 0+0+30 = 30 Punkte

- *Teil (a) unbewertet,*
- *Teil (b) unbewertet*
- *Teil (c) 2x15 Punkte jeweils für die beiden Sprachen*