Einführung in die objektorientierte Programmierung (SEK II)

OCG Informatiktag 2019

Wien, 3. Oktober 2019

Rony G. Flatscher (Rony.Flatscher@wu.ac.at)

Wirtschaftsuniversität Wien, Austria (http://www.wu.ac.at)

Einführung in die objektorientierte Programmierung (SEK II)

- Motivation und Genese der Lehrveranstaltung an WU
  - In einem Semester von Null zur OO-Programmierung
    - Windows, Linux und MacOS!
- Ziele, Infrastruktur und LV-Organisation
  - Kritische Erfolgsfaktoren
- Punktation einer Lehrveranstaltung für SEK II
  - Von Null über OO-Programmierung hin zu MS Windows (MS Office), Linux und MacOS (OpenOffice, GUIs, ...)
- Zusammenfassung und Ausblick, URLs

Motivation und Genese, WU, 1

- Background
  - Handelsakademie (COBOL)
  - Habilitation für *Betriebswirtschaftslehre insbesondere Wirtschaftsinformatik*
    - U.a. eine Reihe von Programmiersprachen (Mainframes, Workstations, PCs) von COBOL, über Assembler, BASIC, (Turbo) Pascal, Prolog, (Object) Rexx, Perl, Python, Tcl, C, C++, C#, Java, ...
  - Seit Mitte der 80er universitäre Lehre
    - Selbstbestimmte Inhalte
    - Selbstbestimmter Unterricht

Motivation und Genese, WU, 2

- Background
  - WU-Studierende mit Interesse an IT
  - Ziele
    - Wettbewerbsvorteile für Studierende schaffen
      - Von Null weg in einem Semester objektorientiertes Programmieren vermitteln
        - Vier Stunden, acht ECTS
      - Effizientere Nutzung der eigenen PC-Ressourcen
      - Bessere Nutzung der IT-Infrastruktur in Betrieben
        - Automatisieren von Geschäftsabläufen

Motivation und Genese, WU, 3

- Background
  - Beginn der Entwicklung der LV mit Windows
    - LV für das Automatisieren mit Hilfe von Makros von MS Word, MS Excel und Sybase RDBMS (Ende 80er!)
    - VBScript erstaunlich komplex
      - Besonderheiten der Syntax müssen verstanden werden,
      - Kosten viel Zeit für Erklärungen
        - z.B. Aufruf mit benannten Argumenten ("a:=1")

Motivation und Genese, WU, 4

- Background
  - Experimenteller Einsatz von REXX in der Lehre
    - Von Mainframe bekannt, unter DOS und OS/2 verfügbar
    - Sprache mit einfacher Syntax, Programme lesen sich wie Pseudocode
      - Zeitersparnis bei Syntax, die für Konzeptvermittlung eingesetzt werden konnte
  - Objektorientierte Version (Object Rexx) ab 1996
    - Unterstützung für Windows (COM, OLE)
    - Ansteuern von Windows und MS Office

Motivation und Genese, WU, 5

- Background
  - Opensource Version seit 2005 von RexxLA
    - IBM übergibt 2004 Quellcode von Object Rexx an den gemeinnützigen, internationalen Verein RexxLA
    - Seit 2005 dauerhaft gratis als "ooRexx" verfügbar
    - Aktive Weiterentwicklung
  - ooRexx-Java-Brücke
    - Proof-of-concept 2000 (!), seitdem voll entwickelt und stabil
    - Verkleidet Java als die einfache Sprache ooRexx :-)

Motivation und Genese, WU, 6

- Background
  - ooRexx-Java-Brücke, Fortsetzung
    - Nutzung von Java für portable GUIs
    - Nutzung von Java für das Ansteuern von Softwarepaketen
    - Programmieren mit ooRexx, Programme laufen unverändert unter Windows, Linux und MacOS !
  - 2019: Inhalt der LV "Business Programming" an der WU
    - 1. Semesterhälfte
      - Von Null weg ins objektorientierte Programmieren
      - Anwendung auf Windows (COM, OLE) und MS Office

Motivation und Genese, WU, 7

- Background
  - 2019: Inhalt der LV "Business Programming" an der WU, Fortsetzung
    - 2. Semesterhälfte unter Nutzung der ooRexx-Java-Brücke
      - Anwendung auf Windows, Linux, MacOS
      - Programme unverändert ausführbar
      - Konzepte und Anwendungen
        - GUIs (awt, swing, JavaFX)
        - Internetprogrammierung (Sockets, Secure Sockets)
        - OpenOffice/LibreOffice
        - XML-Texte verarbeiten (SAX, DOM)

Ziele, Infrastrukur, Organisation, 1

- Ziele
  - Grundlagen der objektorientierten Programmierung
  - Anwendung auf Windows und MS Office
  - Nutzung von Java, ohne Java zu können
    - Anwendung auf Windows, Linux und MacOS
    - Erstellung von OpenOffice/LibreOffice-Programmen
    - Erstellung von GUI-Programmen

Ziele, Infrastrukur, Organisation, 2

- Infrastrukur
  - PC mit Windows, Linux und/oder MacOS
  - **ooRexx** (opensource, frei für alle Systeme verfügbar)
    - Dynamisch typisiert
    - Nachrichtenorientiert (wie Smalltalk)
    - Groß-/Kleinschreibung irrelevant
    - Einfache Syntax, fast wie Pseudocode

```
str="hallo, OCG Informatiktag 2019 an der TU Wien!"
do i=1 to 3
   say "Durchlauf" i":" str
end
```

```
Durchlauf 1: hallo, OCG Informatiktag 2019 an der TU Wien!
Durchlauf 2: hallo, OCG Informatiktag 2019 an der TU Wien!
Durchlauf 3: hallo, OCG Informatiktag 2019 an der TU Wien!
```

Ziele, Infrastrukur, Organisation, 3

Beispiel "zapfen.rex"

```
say "Gib eine Zahl ein:"
parse pull zahl          /* Tastatureingabe                              */
say                      /* Leerzeile                                    */

tmp=zahl                 /* Ausgangswert                                 */
obergrenze=5             /* Obergrenze für Multiplikation und Division   */
ziffern=9                /* maximale Zahl an Ziffern im Ergebnis         */
do i=1 to obergrenze     /* Multiplikationsschleife                      */
   tmp=tmp*i             /* Multiplikation                               */
   say "Multiplikation mit" i":" right(tmp,ziffern)   /* rechtsbündig   */
end
say                      /* Leerzeile                                    */

do i=2 to obergrenze     /* Divisionsschleife                            */
   tmp=tmp/i             /* Division                                     */
   say "Division durch" i":    " right(tmp,ziffern)   /* rechtsbündig   */
end
```

```
Gib eine Zahl ein:
191

Multiplikation mit 1:       191
Multiplikation mit 2:       382
Multiplikation mit 3:      1146
Multiplikation mit 4:      4584
Multiplikation mit 5:     22920

Division durch 2:         11460
Division durch 3:          3820
Division durch 4:           955
Division durch 5:           191
```

Ziele, Infrastrukur, Organisation, 4

- Infrastrukur
  - **Java** (z.B. OpenJDK)
  - **BSF4ooRexx** (opensource, frei für alle Systeme verfügbar)
    - ooRexx-Java-Brücke
    - Verkleidet Java als ooRexx
      - Keine Programmierkenntnisse in Java notwendig!
      - Einfach, Java von ooRexx aus zu nutzen
      - Keine strikte Typisierung, stattdessen dynamisch!
      - Groß-/Kleinschreibung irrelevant!

Ziele, Infrastrukur, Organisation, 5

Beispiel "nutzeJava.rex"

```
dim=.bsf~new("java.awt.Dimension", 100, 200) /* Instanz von Java-Klasse    */
    -- dim: ooRexx-Objekt referenziert das Java-Objekt
say "dim              ="dim                  /* Referenz auf Java-Objekt   */

    -- ooRexx-Nachrichten an das Java-Objekt
say "Breite ('width')="dim~width             /* Attribut/Feld abfragen     */
say "Höhe ('height') ="dim~height            /* Attribut/Feld abfragen     */
say "dim~toString    ="dim~toString          /* Java erzeugte Zeichenkette */

    -- ändern der Java-Felder "width" und "height" von ooRexx aus
dim~width=987                                /* Breite ändern              */
dim~height=654                               /* Höhe ändern                */
say "dim~toString    ="dim~toString          /* Java erzeugte Zeichenkette */

::requires "BSF.CLS"                         /* ooRexx-Java-Brücke laden   */
```

```
dim              =java.awt.Dimension@d8cfe0
Breite ('width')=100
Höhe ('height') =200
dim~toString    =java.awt.Dimension[width=100,height=200]
dim~toString    =java.awt.Dimension[width=987,height=654]
```

Dokumentation der Java-Klassen im Internet verfügbar! Suchen Sie z.B. einfach mit den Schlüsselwörtern:
javadoc java.awt.Dimension

Ziele, Infrastrukur, Organisation, 6

- Organisation
  - 8 ECTS, "teaching load" netto 200 Stunden!
    - Davon 56-60 Kontaktstunden in der Lehrveranstaltung
      - Theoretischer Vortrag mit "*Nutshellbeispielen*"
    - Restliches Zeitbudget
      - Zwei wöchentliche, *möglichst einfache* Hausübungen
      - Ausarbeitung eines *Abschlussprojekts*
  - Gemeinsame Nachrichtengruppe für alle Studierende
    - Erstellte HÜs werden von allen dort verteilt
    - Jeder kann die HÜs der anderen studieren (und lernen)

Ziele, Infrastrukur, Organisation, 7

- Organisation
  - *Selbständiges Recherchieren* nach Lösungen *im Internet*
    - Übernahme von Lösungen ausdrücklich erlaubt!
    - Im MS-Office-Bereich sehr viele hilfreiche Lösungen!
    - Viel an OLE/CLR(.Net)/Java-Dokumentationen verfügbar
  - Bildung von Gruppen aus *zwei* Studierenden
    - Ab drei nimmt die *"U-Boot"-Problematik* stark zu!
  - Fragen ausdrücklich erwünscht
    - "*Dumme* Fragen" ausdrücklich erwünscht!
    - Fragen von KollegInnen beantworten lassen

Ziele, Infrastrukur, Organisation, 8

- Organisation
  - Inhalte aufeinander aufbauend
    - Nutshellbeispiele zur Veranschaulichung
  - Inhalte als "autarke Module" konzipieren
    - Konzentration auf zusammengehörende Konzepte
    - Entfernen ohne Seiteneffekte leichter möglich
  - Folien aufeinander aufbauend
    - Über die Zeit Fragen systematisch Auslöser für
      - Änderung von Reihenfolgen
      - Änderung von Erklärungen
      - Einfügen oder Entfernen von Inhalten/Folien

Zwischenhalt

- Fragen soweit?

Inhalte 1 (SEK II, 1)

- Inhalte ("OO+Windows")
  - Grundlagen der Programmierung
  - Grundlagen der objektorientierten Programmierung
  - Grundlagen: Prozess, "Umgebung", Standarddateien
  - Modul "Windows"
    - Grundlagen der Programmierung über OLE/COM
    - Grundlagen der Programmierung der Windows Shell
    - Grundlagen der Programmierung von MS Office

Inhalte 1 (SEK II, 1)

Grundlagen Programmierung, 1

- Inhalte
  - Erlaubte Zeichen
  - Verarbeitung durch den ooRexx-Interpreter
  - Variable, Zuweisung, Konstante
  - Kommentare (Block /*...*/, zeilenbezogen -- ...)
  - Operationen (Verknüpfung, arithmetisch)
  - Anweisung: Ende (Zeilenende oder ;), Umbruch (-)
  - Blöcke (DO)

Inhalte 1 (SEK II, 1)
Grundlagen Programmierung, 2

- Inhalte
  - Vergleiche, Boole'sche Werte und Verknüpfungen
  - Bedingte Ausführung (IF)
  - Mehrfachauswahl (SELECT)
  - Wiederholungen, Kontrollvariablen
  - Verknüpfungen mit und ohne Leerzeichen
  - Kommandos
    - Ausführung von Befehlen in der Umgebung

Inhalte 1 (SEK II, 1)
Grundlagen Programmierung, 3

- Inhalte
  - Sprungmarken, Routinen, Funktionen
  - Keine reservierten Bezeichner, drei Bezeichner, die vom Interpreter verwendet werden: result, rc, sigline
  - Eingebaute Funktionen, externe Funktions-bibliotheken, Suchreihenfolgen
  - Geltungsbereiche ("scopes")
  - Stammvariable (assoziative Felder/Arrays)
  - Zerlegungsanweisung (PARSE)

Inhalte 1 (SEK II, 1)
Grundlagen Programmierung, 4

- Inhalte
  - Ausnahmen, Fehlerbehandlung
  - Argumente als Referenzen übernehmen
  - Direktiven
    - Anweisungen an den Interpreter, die vor dem Start eines Programms ausgeführt werden
      - ROUTINE-Direktive
      - REQUIRES-Direktive
      - Einbinden von anderen Programmen

Inhalte 1 (SEK II, 1)

Grundlagen OO-Programmierung, 1

- Inhalte
  - Datentyp, abstrakter Datentyp (ADT)
    - Attribute, Methoden/Operationen
    - Implementierung über die Direktiven CLASS, ATTRIBUTE, METHOD
  - Nachrichten, kaskadierende Nachrichten
  - Instanzen/Werte/Objekte erzeugen
    - Konstruktoren (Methode mit dem Namen INIT)

Inhalte 1 (SEK II, 1)
Grundlagen OO-Programmierung, 2

- Inhalte
  - Müllsammler ("garbage collector")
    - Destruktoren (Methode mit dem Namen UNINIT)
  - Klassifikationsbaum, Einfach-, Mehrfachvererbung
    - Laufzeitvariable self und super in Methoden
    - Vererbung
      - konzeptionell durch die Suche des Empfängerobjekts nach Methoden, die so heißen wie die empfangene Nachricht in seiner Klasse oder allen übergeordneten Klassen

Inhalte 1 (SEK II, 1)
Grundlagen OO-Programmierung, 3

- Inhalte
  - UNKNOWN-Mechanismus
    - Möglichkeit, unbekannte Nachrichten abzufangen
  - Wiederholung
    - Implementierung von Klassen, Attributen, Methoden
    - Nachrichten
    - Konstruktoren, Destruktoren, Unknown-Mechanismus
    - Geltungsbereiche
    - Vererbung

Inhalte 1 (SEK II, 1)
Grundlagen OO-Programmierung, 4

- Inhalte
  - Überblick über die mitgelieferten Klassen
    - Fundamentale Klassen
    - Utility-Klassen
    - Sammelklassen ("collection classes")
      - Gemeinsame Methoden
      - Abarbeiten aller gesammelten Werte/Objekte/Instanzen

Inhalte 1 (SEK II, 1)

Beispiel "geburtstag.rex"

```
geb1=.Geburtstag~new        /* erzeuge einen Geburtstag                */
geb1~datum="2015-09-01"     /* weise Geburtsdatum zu                   */
geb1~zeit ="16:00"          /* weise Geburtszeit zu                    */

geb2=.Geburtstag~new        /* erzeuge einen Geburtstag                */
geb2~datum="2018-02-29"     /* weise Geburtstdatum zu                  */
geb2~zeit ="19:19"          /* weise Geburtszeit zu                    */

say "Geburtstag 1:" geb1~datum geb1~zeit  /* zeige Attributwerte       */
say "Geburtstag 2:" geb2~datum geb2~zeit  /* zeige Attributwerte       */

::CLASS     Geburtstag      /* Name des Datentyps/der Klasse           */
::ATTRIBUTE datum           /* definiere das Attribut "datum"          */
::ATTRIBUTE zeit            /* definiere das Attribute "zeit"          */
```

```
Geburtstag 1: 2015-09-01 16:00
Geburtstag 2: 2018-02-29 19:19
```

Inhalte 1 (SEK II, 1)

Beispiel 2 "person.rex"

```
p1=.person~new("Albert", "Einstein", 45000)  /* erzeuge eine Person           */
p2=.person~new("Vera", "Doe", 25000)         /* erzeuge eine Person           */

say "Person 1:    " p1~vorname p1~nachname p1~gehalt  /* zeige Attributwerte  */
say "Person 2:    " p2~vorname p2~nachname p2~gehalt  /* zeige Attributwerte  */
say "Gehaltssumme:" p1~gehalt + p2~gehalt             /* summiere Gehälter    */

::CLASS Person              /* Name des Datentyps/der Klasse                  */
::ATTRIBUTE vorname         /* definiere das Attribut "vorname"               */
::ATTRIBUTE nachname        /* definiere das Attribut "nachname"              */
::ATTRIBUTE gehalt          /* definiere das Attribut "gehalt"                */

::METHOD INIT               /* Konstruktormethode                             */
  EXPOSE vorname nachname gehalt    /* direkter Zugriff auf Attribute         */
  USE ARG vorname, nachname, gehalt /* Zuweisung der Argumente                */

::METHOD erhoeheGehalt      /* Methode "erhoeheGehalt"                        */
  EXPOSE gehalt             /* direkter Zugriff auf Attribut "gehalt"         */
  USE ARG erhoehung         /* Erhöhungsbetrag entgegennehmen                 */
  gehalt=gehalt+erhoehung /* neuen Gehalt berechnen und Attribut zuweisen     */
```

```
Person 1:     Albert Einstein 45000
Person 2:     Vera Doe 25000
Gehaltssumme: 70000
```

Inhalte 1 (SEK II, 1)

Grundlagen Windows OLE/COM, 1

- Inhalte
  - Geschichte und Konzepte von COM, OLE, ActiveX
    - IUnknown, IDispatch, ...
  - Windows Registry
    - Zentral für Windows ("single point of failure")
    - Organisation in "hives"
    - Universally/global unique identifiers (UUID/GUID)
      - CLSID
    - ProgID
  - Monikers

Inhalte 1 (SEK II, 1)
Grundlagen Windows OLE/COM, 2

- Inhalte
  - OLEObject
    - Stellvertreterklasse ("proxy class")
  - OLEObject-Instanzen repräsentieren Windows-Objekte
    - ooRexx-Nachrichten an OLEObject-Instanzen verursachen
      - Abfrage/Setzen von Windows-Attributen
      - Ausführen von Windows-Methoden
      - Auskunft über welche öffentlichen Attribute, Konstante, Ereignisse und Methoden das Windows-Objekt verfügt

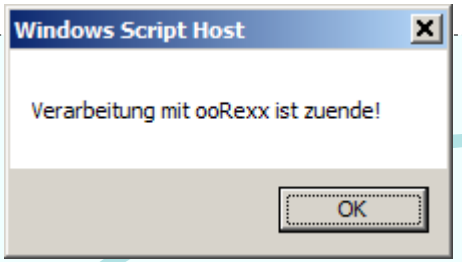
Inhalte 1 (SEK II, 1)
Grundlagen Windows Shell

- Inhalte
  - Klasse "WScript.Shell"
    - Erlaubt die Windows-Benutzerschnittstelle über OLE anzusteuern
    - Z.B. Methoden "Environment", "SpecialFolders", "CreateShortcut", ...
  - Zahlreiche OLE-Nutshellbeispiele mit ooRexx für Windows mitgeliefert
    - Vgl. "%REXX_HOME%\ooRexx\samples\ole\"

Inhalte 1 (SEK II, 1)

Beispiel "wscript.shell.rex"

```
wscript=.OLEObject~New("WScript.Shell")       /* Instanz der Shell erzeugen */
say "Folgende besondere Verzeichnisse stehen zur Verfügung:"
do verzeichnis over wscript~specialFolders    /* über Array iterieren       */
  say "   " verzeichnis                       /* Verzeichnis zeigen         */
end
wscript~popup("Verarbeitung mit ooRexx ist zuende!")  /* Popup zeigen       */
```




```
Folgende besondere Verzeichnisse stehen zur Verfügung:
    C:\Users\Public\Desktop
    C:\ProgramData\Microsoft\Windows\Start Menu
    C:\ProgramData\Microsoft\Windows\Start Menu\Programs
    C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Startup
    C:\Users\Administrator\Desktop
    C:\Users\Administrator\AppData\Roaming
    C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Printer Shortcuts
    C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Templates
    C:\Windows\Fonts
    C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Network Shortcuts
    C:\Users\Administrator\Desktop
    C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu
    C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\SendTo
    C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Recent
    C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
    C:\Users\Administrator\Favorites
    C:\Users\Administrator\Documents
    C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Start Menu\Programs
```

Inhalte 1 (SEK II, 1)
Grundlagen Windows MS Office

- Inhalte
  - Internetrecherche zu MS Office
    - msdn – Microsoft Developer Network
      - Ausgezeichnete Ressourcen und Dokumentationen!
      - URL: https://msdn.microsoft.com
  - Erläuterung der mit ooRexx-mitgelieferten Nutshellbeispiele
    - MS Access: "samples\ole\apps\samp14.rex", "samples\ole\apps\MSAccessDemo.rex"
    - MS Excel: "samples\ole\apps\samp09.rex"
    - MS Word: "samples\ole\apps\samp08.rex"

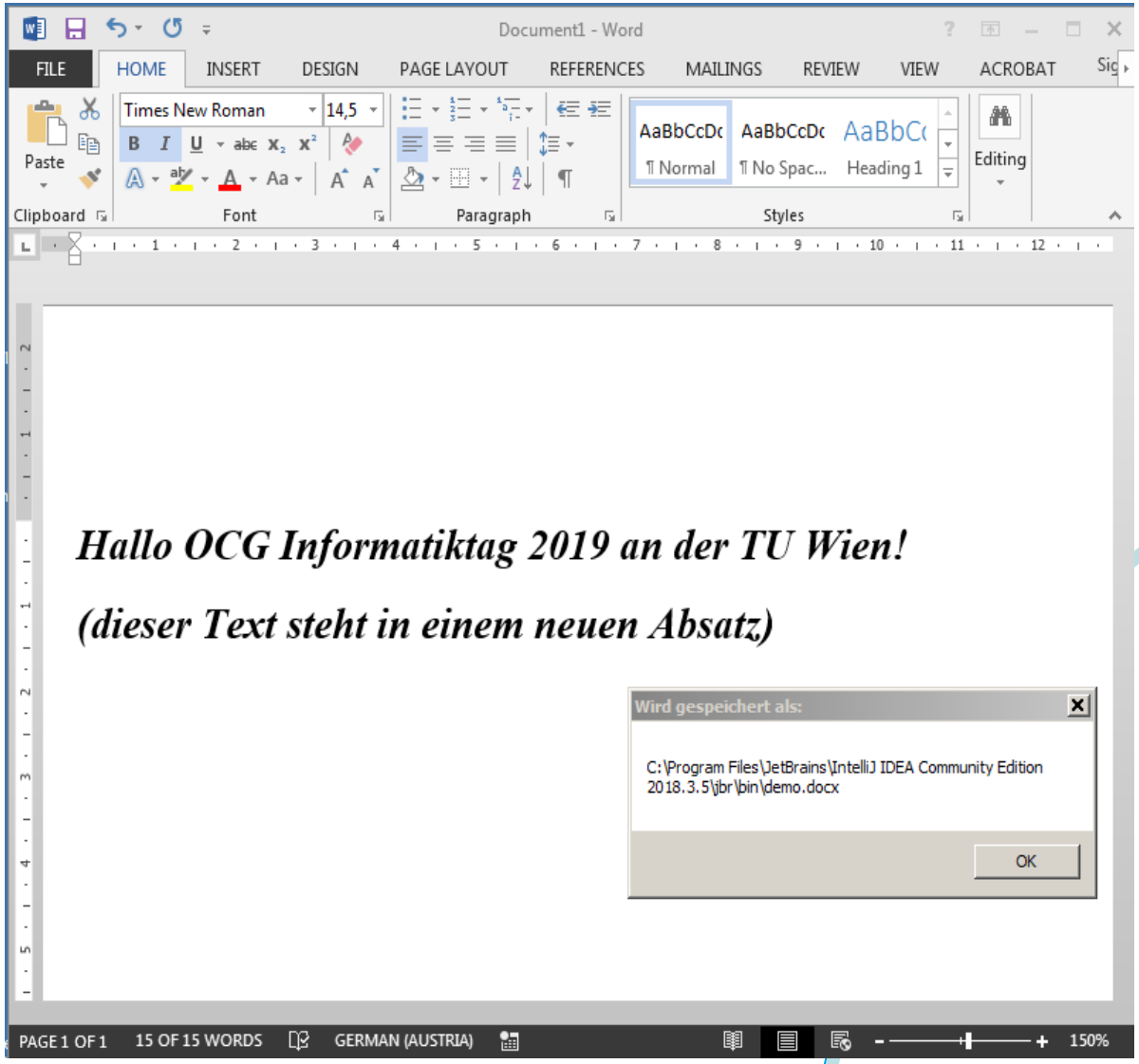
Inhalte 1 (SEK II, 1)

Beispiel "word.rex"

```
word=.OLEObject~New("Word.Application")   /* erzeuge eine Word-Instanz      */
word~visible=.true                        /* zeige das Wordfenster          */
wordDokument=word~documents~add           /* erzeuge ein neues Dokument     */

auswahl=word~selection                    /* Zugriff auf Dokumentbereich    */
auswahl~typeText("Hallo OCG Informatiktag 2019 an der TU Wien!")
auswahl~typeParagraph                     /* neuer Absatz                   */
auswahl~typeText("(dieser Text steht in einem neuen Absatz)")

auswahl~wholeStory                        /* alles auswählen                */
font=auswahl~font                         /* Font-Objekt der Auswahl abrufen*/
font~name  ="Times New Roman"             /* Schriftfamilie festlegen       */
font~size  =14.5                          /* Schriftgröße festlegen         */
font~bold  =.true                         /* fett setzen                    */
font~italic=.true                         /* kursiv setzen                  */

   /* speichere Worddokument im aktuellen Verzeichnis                       */
dateiName=directory()"\demo.docx" /* Dateinamen festlegen                   */
call rxMessageBox dateiName, "Wird gespeichert als:"  /* Dateinamen zeigen */
wordDokument~saveAs(dateiName)    /* Dokument speichern                    */
word~quit                         /* Word beenden                          */
```

Inhalte 1 (SEK II, 1)

Beispiel "word.rex", Resultat



Zwischenhalt

- Fragen soweit?

Inhalte 2 (SEK II, 2)

- Inhalte ("Java+coole Sachen")
  - Modul "Portable Programmierung"
    - Grundlagen von Java
    - Grundlagen der GUI-Programmierung
    - Grundlagen der Internet-Programmierung
    - Grundlagen der Programmierung von Apache OpenOffice (AOO)/LibreOffice (LO)
    - Grundlagen der Verarbeitung von XML-Dateien
    - Grundlagen der JavaFX-Programmierung

Inhalte 2 (SEK II, 2)
Grundlagen von Java, 1

- Inhalte
  - Geschichte
  - "Write once, run everywhere"
    - Kompilierte Java-Klassen laufen unverändert auf allen Betriebssystemen, auch GUI-Klassen!
  - Klassen, Felder (= Attribute), Methoden
  - Sichtbarkeitsregeln
  - Strikte Datentypen
    - Besonderheit: "primitive Datentypen"
  - JavaDoc (HTML-Dokumentation aller Klassen)

Inhalte 2 (SEK II, 2)
Grundlagen von Java, 2

- Inhalte
  - BSF4ooRexx mit "BSF.CLS"-ooRexx-Paket
    - Brücke zwischen ooRexx und Java
    - Verkleidet Java als ooRexx
      - Nachrichtenbasiert
      - Dynamisch (!)
      - Groß- und Kleinschreibung (fast) nicht wichtig
      - Interpretiert
    - Erlaubt es, Java-Unkundigen *alle Java-Klassen direkt in ooRexx auf allen Plattformen* zu nutzen!

Inhalte 2 (SEK II, 2)
Grundlagen von Java, 3

- Inhalte
  - Besonderheiten
    - Java-Arrays sind von fixer Kapazität und strikt typisiert!
      - Können wie ooRexx-Arrays angesprochen werden!
      - Können in ooRexx auch mit "do ... over"-Schleifen abgearbeitet werden
    - Interface- und abstrakte Java-Klassen
      - Abstrakte Java-Methoden können in ooRexx einfach implementiert werden!

Inhalte 2 (SEK II, 2)
Grundlagen von Java, 4

- Inhalte
  - Besonderheiten
    - ooRexx kann als Java-Skriptsprache eingesetzt werden
      - Beinhaltet die Implementierung von "javax.script"
        - Autor war als Experte in JSR-223 an der Definition beteiligt
      - JavaFX kann ooRexx als Controllersprache verwenden
    - Anbindung an Apache OpenOffice (AOO)/LibreOffice (LO) über deren Java-Schnittstellen
      - ooRexx kann auch als Makrosprache in AOO installiert werden!

Inhalte 2 (SEK II, 2)
Grundlagen der GUI-Programmierung

- Inhalte
  - GUI-Komponenten, GUI-Container
  - Anordnung der GUI-Komponenten
    - Layoutmanager
  - Ereignisverarbeitung
    - GUI-Ereignisse (events)
    - GUI-Thread
    - Event-Handler (Callbacks)
  - Synchronisierung des Haupt- mit dem GUI-Thread

Inhalte 2 (SEK II, 2)

Beispiel "gui.rex", 1

```
reh=.RexxHandler~new        -- Rexx event handler for Java events
jeh=BsfCreateRexxProxy(reh, , "java.awt.event.WindowListener", -
                              "java.awt.event.ActionListener")
frame=.bsf~new("java.awt.Frame", "Demonstrating the .BSF.Dialog Class")

frame~addWindowListener(jeh)
frame~setLayout( .bsf~new("java.awt.FlowLayout") ) -- set layout manager
button1=.bsf~new('java.awt.Button', 'BSF.Dialog without parent')
frame~add(button1~~addActionListener(jeh))
button2=.bsf~new('java.awt.Button', 'BSF.Dialog with this frame as parent')
frame~add(button2~~addActionListener(jeh))
frame ~~pack ~~setVisible(.true) ~~toFront -- layout the Frame object, show it

reh~waitForExit          -- wait until we are allowed to end the program

::requires BSF.cls       -- load ooRexx-Java bridge

    /* Rexx event handler which handles Window and Action events  */
::class RexxHandler
::attribute closeApp     -- control variable
::method init            -- constructor
  expose closeApp count  -- direct access to attributes
  closeApp  = .false     -- control variable
  count     = 0          -- set counter to 0

::method waitForExit     -- blocking (waiting) method
  expose closeApp        -- direct access to control variable
  guard on when closeApp=.true   -- blocks until control variable is .true


… cut Fortsetzung auf nächster Folie ...
```

Inhalte 2 (SEK II, 2)

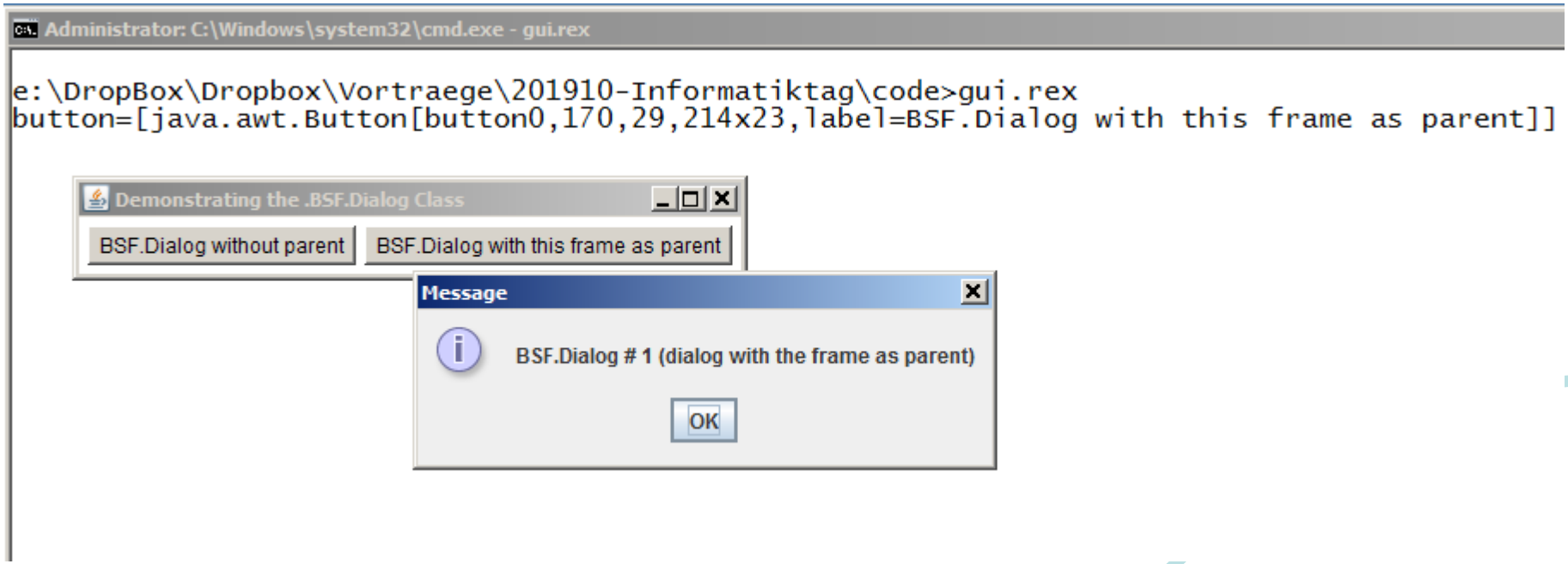
Beispiel "gui.rex", 2

… Fortsetzung von vorhergehender Seite ….

```
::method windowClosing  -- event method (from WindowListener)
  expose closeApp       -- direct access to control variable
  closeApp=.true        -- change control variable to unblock

::method unknown        -- intercepts unhandled WindowListener events

::method actionPerformed   -- event method (from ActionListener)
  expose count          -- direct access to count variable
  use arg eventObject

  button=eventObject~source   -- get the button object
  say "button="pp(button~toString)
  count+=1              -- increase counter
  if button~label="BSF.Dialog without parent" then   --  parentless modal dialog
      .bsf.dialog~messageBox("BSF.Dialog #" count "(centered on screen)")
  else      -- create a modal dialog for the frame itself
  do
     bd=.bsf.dialog~new(button) -- or directly the Frame object of the button
     bd~messageBox("BSF.Dialog #" count "(dialog with the frame as parent)")
  end
```

Inhalte 2 (SEK II, 2)

Beispiel "gui.rex", 3



Inhalte 2 (SEK II, 2)
Grundlagen der Internet-Programmierung, 1

- Inhalte
  - Konzepte aus der Telefonie, "Socket"-Programmierung
    - "Socket" (Steckköffnung), "patch cord" (Steckerkabel)
  - Client-/Serverprogramme
    - Können auf verschiedenen Rechnern laufen
    - Können in unterschiedlichen Sprachen programmiert sein
  - "Protokoll" zur Kommunikation notwendig
    - "RFC" (request for comment) z.B. http, ftp, telnet, ...
      - URL: https://www.ietf.org/standards/rfcs/

Inhalte 2 (SEK II, 2)

Beispiel Socketprogrammierung

```
-- socketServer.rex: run in its own terminal window
   -- create server socket listening on port 8888
srvSock=.bsf~new("java.net.ServerSocket", 8888)
say "SERVER: starting to accept clients..."
   -- accept client, returns socket to client
socket2client=srvSock~accept
say "SERVER: client" pp(socket2client~toString) "accepted."
   -- receive data from client, create a Java "byte"-array
barr=.bsf~bsf.createArray('byte.class', 2048)
   -- get bytes from client, returns number of bytes read
n=socket2client~getInputStream~read(barr)
   -- turn received byte array to Rexx string
say "SERVER: received:" pp(BsfRawBytes(barr,n))
   -- send data to client
os=socket2client~getOutputStream
msg="Hello from the ooRexx-server!"
say "SERVER: sending" pp(msg) "to client..."
   -- turn Rexx string to byte array before sending
socket2client~getOutputStream~write(BsfRawBytes(msg))
say "SERVER: about to end."

::requires BSF.cls      -- load ooRexx-Java bridge
```

```
-- socketClient.rex: run in its own terminal window
   -- get server's (this computer) Internet address
lh=.bsf~bsf.import('java.net.InetAddress') ~getLocalHost
   -- connect to server at port 8888
socket2server=.bsf~new('java.net.Socket', lh, 8888)
say "CLIENT: connected to server" pp(socket2server~toString)
   -- send data to server
msg="Hello, this is from your ooRexx-client!"
say "CLIENT: sending" pp(msg) "to server..."
   -- turn Rexx string to byte array
socket2server~getOutputStream~write(BsfRawBytes(msg))
   -- receive data from server, create a Java "byte"-array
b=.bsf~bsf.createArray('byte.class', 100)
   -- get bytes from server, returns number of bytes read
received=socket2server~getInputStream~read(b)
say "CLIENT: received from server: ["||BsfRawBytes(b,received)"]"
say "CLIENT: about to end."

::requires BSF.cls      -- load ooRexx-Java bridge
```

```
SERVER: starting to accept clients...
SERVER: client [Socket[addr=/137.208.114.24,port=60851,localport=8888]] accepted.
        CLIENT: connected to server [Socket[addr=T450sRGF/137.208.114.24,port=8888,localport=60851]]
        CLIENT: sending [Hello, this is from your ooRexx-client!] to server...
SERVER: received: [Hello, this is from your ooRexx-client!]
SERVER: sending [Hello from the ooRexx-server!] to client...
SERVER: about to end.
        CLIENT: received from server: [Hello from the ooRexx-server!]
        CLIENT: about to end.
```

Inhalte 2 (SEK II, 2)

Grundlagen der Internet-Programmierung, 2

- Inhalte
  - Kommunikation im Klartext
    - Potentielle Sicherheitsprobleme
  - Kryptografierte Kommunikation
    - SSL (secure socket layer)
      - In Java der Java-Socketprogrammierung nachempfunden
      - Programmierung daher praktisch identisch
      - Erstellung und Einbindung von Zertifikaten für den Kommunikationsaufbau notwendig
        - Entsprechendes Tool (keytool) wird im JDK mitgeliefert!

Inhalte 2 (SEK II, 2)
Grundlagen der Programmierung AOO/LO, 1

- Inhalte
  - Geschichte, Client-/Serverarchitektur
  - UNO (unified network objects)
    - Architektur, unterschiedliche Arten von Klassen
      - Services mit Interfaces, Properties, ...
    - Bedeutung der UNO-Interfaceklassen
  - "UNO.CLS"-Paket (Teil von BSF4ooRexx)
    - Erleichtert das Interagieren mit AOO/LO massiv
    - Ermöglicht auch die reflektive Dokumentation von UNO-Klassen und UNO-Objekten (!) zur Laufzeit

Inhalte 2 (SEK II, 2)
Grundlagen der Programmierung AOO/LO, 2

- Inhalte
  - Module mit Architektur und Nutshell-Beispielen
    - "swriter" - Textverarbeitung
    - "scalc" - Tabellenkalkulation
    - "sdraw" - Zeichenmodul
    - "simpress" - Präsentationsmodul

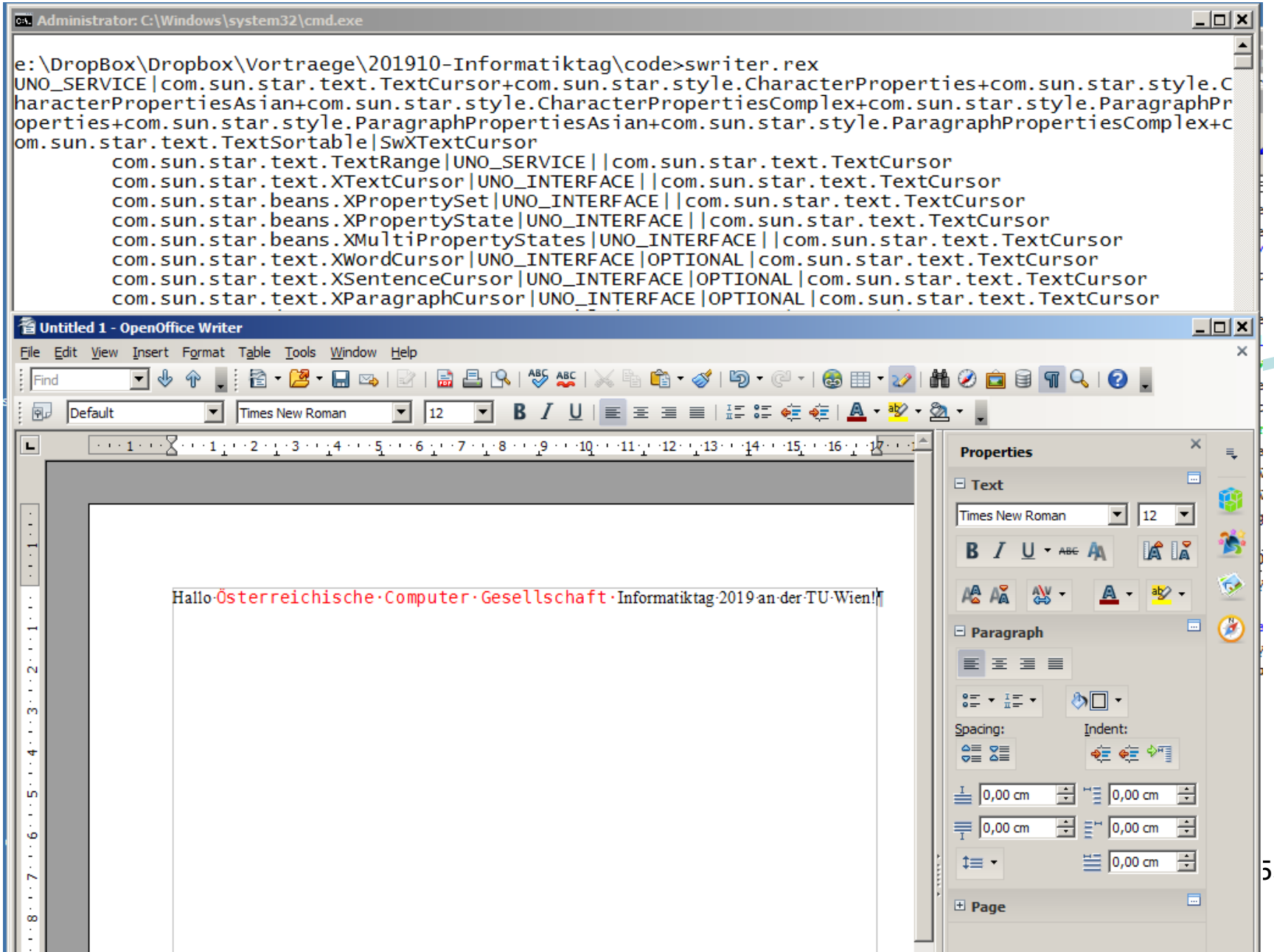
Inhalte 2 (SEK II, 2)

Beispiel "swriter.rex"

```
xDesktop=uno.createDesktop()          -- bootstrap & get access to XDesktop
xcl=xDesktop~XComponentLoader         -- get XComponentLoader interface
uri="private:factory/swriter"         -- new swriter document
doc=xcl~loadComponentFromURL(uri,"_blank",0,.uno~noProps)
   -- add text
xText=doc~XTextDocument~getText       -- get text object
xText~setString("Hallo OCG Informatiktag 2019 an der TU Wien!")
   -- change second word
xTextCursor=xText~createTextCursor    -- character based cursor
xTextCursor~gotoStart(.false)         -- make sure we are at start
   -- replace acronym "OCG" with long form
xWordCursor=xTextCursor~XWordCursor -- get the XWordCursor interface
xWordCursor~gotoNextWord(.false)      -- XTextRange represents first word
xWordCursor~gotoNextWord(.true)       -- select second word, includes blank!
xWordCursor~setString("Österreichische Computer Gesellschaft ")
   -- change color
red=box("int", "FF 00 00"x ~c2d)      -- red (RGB color) as java.lang.Integer
xWordCursor~XPropertySet~setPropertyValue("CharColor", red)
   -- change font
fontName="DejaVu Sans Mono"
xWordCursor~XPropertySet~setPropertyValue("CharFontName", fontName)
say ppd(xWordCursor~uno.getDefinition) -- show UNO definitions

::requires "UNO.CLS" -- get UNO support which requires "BSF.CLS"
```

Inhalte 2 (SEK II, 2)

Beispiel "swriter.rex", Screenshot



Inhalte 2 (SEK II, 2)

Grundlagen der Verarbeitung von XML-Dateien

- Inhalte
  - Grundlagen der Verarbeitung von XML-Dateien
  - SAX (Simple API for XML)
    - Callbacks während des Parsens (Zerlegen)
  - DOM (Document Object Model, W3C)
    - Rekursive Abarbeitung des Zerlegungsbaums

Inhalte 2 (SEK II, 2)
Grundlagen der JavaFX-Programmierung

- Inhalte
  - Grundlagen der JavaFX-Programmierung
  - Tools (SceneBuilder)
    - Interaktive Erstellung auch komplexer GUIs
    - Erweiterbar mit Community JavaFX-Komponenten
  - FXML-Definition mit Skriptsprachensupport

Inhalte 2 (SEK II, 2)

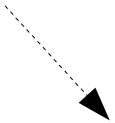
Beispiel "fxml_01.fxml"

```
-<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.AnchorPane?>

<!-- the processing instruction (PI) defines the Java script engine named 'rexx'
     to be used for executing code in this FXML file: running "fxml_01_controller.rex"
     and the code in the 'onAction' event attribute for the Button element -->
<?language rexx?>

<AnchorPane id="AnchorPane" prefHeight="200" prefWidth="400"
            xmlns:fx="http://javafx.com/fxml/1">

    <!-- call Rexx program, its public routine "buttonClicked" is known afterwards -->
    <fx:script source="fxml_01_controller.rex" />

    <children>
        <!-- the Rexx code in the 'onAction' attribute will be invoked by JavaFX;
             note: last argument is the slotDir argument from BSF4ooRexx
        -->
        <Button fx:id="idButton1" layoutX="170.0" layoutY="89.0"
                onAction="call buttonClicked arg(arg())"
                text="Click Me!" textFill="GREEN" />

        <Label fx:id="idLabel1" alignment="CENTER" contentDisplay="CENTER"
                                layoutX="76.0"     layoutY="138.0"
                                minHeight="16"     minWidth="49"
                                prefHeight="16.0"  prefWidth="248.0"
                                textFill="GREEN" />
    </children>
</AnchorPane>
```

Inhalte 2 (SEK II, 2)

Beispiel "fxml_01_controller.rex"

```
    -- "fxml_01.controller.rex" called from the FXMLLoader
    -- the routine will be called each time the button gets pressed
::routine buttonClicked public
  use arg slotDir
  now=.dateTime~new -- time of invocation
  say now": arrived in routine 'buttonClicked' ..."
     -- fetch idLabel JavaFX object using a RexxScript annotation:
  /* @get(idLabel1) */
  say '... current value of label='pp(idLabel1~getText)
  idLabel1~text="Clicked at:" now          -- set text property
  say '...     new value of label='pp(idLabel1~getText)
  say
```



```
REXXout>2019-10-02T18:44:03.658000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[]
REXXout>...     new value of label=[Clicked at: 2019-10-02T18:44:03.658000]
REXXout>
REXXout>2019-10-02T18:44:04.234000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[Clicked at: 2019-10-02T18:44:03.658000]
REXXout>...     new value of label=[Clicked at: 2019-10-02T18:44:04.234000]
REXXout>
```

Inhalte 2 (SEK II, 2)

Beispiel "fxml_01.rex"

```
   -- "fxml_01.rex"
   -- change directory to program location
parse source  . . pgm                -- get full path to this program
call directory filespec('L', pgm)    -- change to its directory

rxApp=.RexxApplication~new -- create Rexx object that will set up JavaFX
jrxApp=BSFCreateRexxProxy(rxApp, ,"javafx.application.Application")
jrxApp~launch(jrxApp~getClass, .nil)    -- launch the application

::requires "BSF.CLS"    -- get ooRexx-Java bridge

 -- Rexx class implements "javafx.application.Application" abstract method "start"
::class RexxApplication

::method start          -- Rexx method implements the abstract method
  use arg primaryStage  -- fetch the primary stage (window)
  primaryStage~setTitle("Hello JavaFX from ooRexx! (Green Version)")

   -- create an URL for the FMXLDocument.fxml file (hence the protocol "file:")
  fxmlUrl=.bsf~new("java.net.URL", "file:fxml_01.fxml")
   -- use FXMLLoader to load the FXML and create the GUI graph
  rootNode=bsf.loadClass("javafx.fxml.FXMLLoader")~load(fxmlUrl)
   -- create a scene for our document from the GUI graph
  scene=.bsf~new("javafx.scene.Scene", rootNode)
  primaryStage~setScene(scene)  -- set the stage to our scene
  primaryStage~show             -- show the stage (and thereby our scene)
```

Inhalte 2 (SEK II, 2)

Beispielsausgabe

Administrator: C:\Windows\system32\cmd.exe - fxml_01.rex

```
e:\DropBox\Dropbox\Vortraege\201910-Informatiktag\code>fxml_01.rex
REXXout>2019-10-02T18:44:03.658000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[]
REXXout>...     new value of label=[Clicked at: 2019-10-02T18:44:03.658000]
REXXout>
REXXout>2019-10-02T18:44:04.234000: arrived in routine 'buttonClicked' ...
REXXout>... current value of label=[Clicked at: 2019-10-02T18:44:03.658000]
REXXout>...     new value of label=[Clicked at: 2019-10-02T18:44:04.234000]
REXXout>
```

Zusammenfassung und Ausblick

- Möglich, in einem Fach von Null weg OO-Programmieren für Windows, Linux, MacOS in einem Schuljahr, wenn
  - kritische Erfolgsfaktoren beachtet werden
- Todo
  - Rahmenbedingungen für SEK II definieren
  - Organisation und Inhalte entsprechend anpassen
  - Abgestimmte Lehrunterlagen für LehrerInnen und Schüler erstellen

URLs

- RexxLA-Homepage (gemeinnütziger Verein, Eigentümer von ooRexx, BSF4ooRexx)

  <http://www.rexxla.org/>
- ooRexx 5.0 beta auf Sourceforge

  <https://sourceforge.net/projects/oorexx/files/oorexx/5.0.0beta/>
- BSF4ooRexx auf Sourceforge (ooRexx-Java-Brücke)

  <https://sourceforge.net/projects/bsf4oorexx/>
- Einführung in ooRexx (254 Seiten), wesentlich billiger, wenn direkt am WU-Campus im Management Book Store gekauft

  <https://www.facultas.at/Flatscher>
- JetBrains "IntelliJ IDEA", professionelle, mächtige Entwicklungsumgebung in Java für alle Betriebssysteme
  - <https://www.jetbrains.com/idea/download>, freie "Community-Edition"
  - Alexander Seik's IntelliJ's *ooRexx-Plugin* mit README.txt (Abruf: 2019-10-01)
    - <https://sourceforge.net/projects/bsf4oorexx/files/Sandbox/aseik/ooRexxIDEA/beta/1.0.5/>
- WU-LV-Übersichten WS 2019/20 zu "Business Programming": <http://vvz.wu.ac.at/cgi-bin/vvz.pl?C=S&LANG=DE&S=19W&LV=3&L2=S&L3=S&T=business+programming&L=&I=&JOIN=AND>