OOP - Objektorientierte Programmierung

Kurze Einführung am Beispiel von Python

Klaus Mandel

23 Feb. 2011 / Computer-Stammtisch

Einführung

- OOP - Objekt Orientierte Programmierung
- Hauptbegriff ist das Objekt
- ein Datentyp, bei dem die Daten und die Funktionen, die auf die Daten angewandt werden, zusammengefasst werden
- Objekte (Instanzen) agieren unabhängig mit ihren Daten und Methoden
- Objekte, definiert durch ihre Klasse, stellen einen Form der Abstraktion dar
- Eigenschaften (Attribute) als auch die Fähigkeiten (Methoden) können privat sein und nach außen versteckt werden
- „BlackBox“oder Datenkapselung

Einführung

- OOP - Objekt Orientierte Programmierung
- Hauptbegriff ist das Objekt
- ein Datentyp, bei dem die Daten und die Funktionen, die auf die Daten angewandt werden, zusammengefasst werden
- Objekte (Instanzen) agieren unabhängig mit ihren Daten und Methoden
- Objekte, definiert durch ihre Klasse, stellen einen Form der Abstraktion dar
- Eigenschaften (Attribute) als auch die Fähigkeiten (Methoden) können privat sein und nach außen versteckt werden
- „BlackBox“oder Datenkapselung

Einführung

- OOP - Objekt Orientierte Programmierung
- Hauptbegriff ist das Objekt
- ein Datentyp, bei dem die Daten und die Funktionen, die auf die Daten angewandt werden, zusammengefasst werden
- Objekte (Instanzen) agieren unabhängig mit ihren Daten und Methoden
- Objekte, definiert durch ihre Klasse, stellen einen Form der Abstraktion dar
- Eigenschaften (Attribute) als auch die Fähigkeiten (Methoden) können privat sein und nach außen versteckt werden
- „BlackBox“oder Datenkapselung

Einführung

- OOP - Objekt Orientierte Programmierung
- Hauptbegriff ist das Objekt
- ein Datentyp, bei dem die Daten und die Funktionen, die auf die Daten angewandt werden, zusammengefasst werden
- Objekte (Instanzen) agieren unabhängig mit ihren Daten und Methoden
- Objekte, definiert durch ihre Klasse, stellen einen Form der Abstraktion dar
- Eigenschaften (Attribute) als auch die Fähigkeiten (Methoden) können privat sein und nach außen versteckt werden
- „BlackBox“oder Datenkapselung

Einführung

- OOP - Objekt Orientierte Programmierung
- Hauptbegriff ist das Objekt
- ein Datentyp, bei dem die Daten und die Funktionen, die auf die Daten angewandt werden, zusammengefasst werden
- Objekte (Instanzen) agieren unabhängig mit ihren Daten und Methoden
- Objekte, definiert durch ihre Klasse, stellen einen Form der Abstraktion dar
- Eigenschaften (Attribute) als auch die Fähigkeiten (Methoden) können privat sein und nach außen versteckt werden
- „BlackBox“oder Datenkapselung

Einführung

- OOP - Objekt Orientierte Programmierung
- Hauptbegriff ist das Objekt
- ein Datentyp, bei dem die Daten und die Funktionen, die auf die Daten angewandt werden, zusammengefasst werden
- Objekte (Instanzen) agieren unabhängig mit ihren Daten und Methoden
- Objekte, definiert durch ihre Klasse, stellen einen Form der Abstraktion dar
- Eigenschaften (Attribute) als auch die Fähigkeiten (Methoden) können privat sein und nach außen versteckt werden
- „BlackBox“oder Datenkapselung

Einführung

- OOP - Objekt Orientierte Programmierung
- Hauptbegriff ist das Objekt
- ein Datentyp, bei dem die Daten und die Funktionen, die auf die Daten angewandt werden, zusammengefasst werden
- Objekte (Instanzen) agieren unabhängig mit ihren Daten und Methoden
- Objekte, definiert durch ihre Klasse, stellen einen Form der Abstraktion dar
- Eigenschaften (Attribute) als auch die Fähigkeiten (Methoden) können privat sein und nach außen versteckt werden
- „BlackBox“oder Datenkapselung

Begriffe der OOP

- Schlüsselwort z.B. class wird einer Klasse vorangestellt
- Instanzen - reale Inkarnationen eines Objektes
- Attribute (Eigenschaften)
- Methoden (Fähigkeiten)
- Klassenattribute (klassenglobale Daten)
- Vererbung - Übernahme von Eigenschaften und Fähigkeiten (Ableiten)

Begriffe der OOP

- Schlüsselwort z.B. class wird einer Klasse vorangestellt
- Instanzen - reale Inkarnationen eines Objektes
- Attribute (Eigenschaften)
- Methoden (Fähigkeiten)
- Klassenattribute (klassenglobale Daten)
- Vererbung - Übernahme von Eigenschaften und Fähigkeiten (Ableiten)

Begriffe der OOP

- Schlüsselwort z.B. class wird einer Klasse vorangestellt
- Instanzen - reale Inkarnationen eines Objektes
- Attribute (Eigenschaften)
- Methoden (Fähigkeiten)
- Klassenattribute (klassenglobale Daten)
- Vererbung - Übernahme von Eigenschaften und Fähigkeiten (Ableiten)

Begriffe der OOP

- Schlüsselwort z.B. class wird einer Klasse vorangestellt
- Instanzen - reale Inkarnationen eines Objektes
- Attribute (Eigenschaften)
- Methoden (Fähigkeiten)
- Klassenattribute (klassenglobale Daten)
- Vererbung - Übernahme von Eigenschaften und Fähigkeiten (Ableiten)

Begriffe der OOP

- Schlüsselwort z.B. class wird einer Klasse vorangestellt
- Instanzen - reale Inkarnationen eines Objektes
- Attribute (Eigenschaften)
- Methoden (Fähigkeiten)
- Klassenattribute (klassenglobale Daten)
- Vererbung - Übernahme von Eigenschaften und Fähigkeiten (Ableiten)

Begriffe der OOP

- Schlüsselwort z.B. class wird einer Klasse vorangestellt
- Instanzen - reale Inkarnationen eines Objektes
- Attribute (Eigenschaften)
- Methoden (Fähigkeiten)
- Klassenattribute (klassenglobale Daten)
- Vererbung - Übernahme von Eigenschaften und Fähigkeiten (Ableiten)

weiter: Begriffe der OOP

- Polymorhie - Funktionen überladen
  verschiedene Objekte reagieren unterschiedlich auf die gleiche Fähigkeit
- „Late binding“- Wahl der Funktion zur Laufzeit
  welche Methode in einer Klassenhierachie gewählt wird, ist erst zur Laufzeit klar
- Persistenz - Gültigkeit eines Objektes

weiter: Begriffe der OOP

- Polymorhie - Funktionen überladen
  verschiedene Objekte reagieren unterschiedlich auf die gleiche Fähigkeit
- „Late binding“- Wahl der Funktion zur Laufzeit
  welche Methode in einer Klassenhierachie gewählt wird, ist erst zur Laufzeit klar
- Persistenz - Gültigkeit eines Objektes

weiter: Begriffe der OOP

- Polymorhie - Funktionen überladen
  verschiedene Objekte reagieren unterschiedlich auf die gleiche Fähigkeit
- „Late binding“- Wahl der Funktion zur Laufzeit
  welche Methode in einer Klassenhierachie gewählt wird, ist erst zur Laufzeit klar
- Persistenz - Gültigkeit eines Objektes

Besonderheiten von Klassen in Python

- Definition mit class KLASSE (ist_abgeleitet_von):
- Constructor: __init__(self, ...):
- kein Destructor; Entfernung überflüssiger Okjekte durch Garbage Collection
- self und cls - erster Parameter von Instanz-Methoden bzw. Klassenmethoden zeigen auf die eigene Instanz bzw. Klasse
- Decoratoren: @classmethod und @staticmethod

Besonderheiten von Klassen in Python

- Definition mit class KLASSE (ist_abgeleitet_von):
- Constructor: __init__(self, ...):
- kein Destructor; Entfernung überflüssiger Okjekte durch Garbage Collection
- self und cls - erster Parameter von Instanz-Methoden bzw. Klassenmethoden zeigen auf die eigene Instanz bzw. Klasse
- Decoratoren: @classmethod und @staticmethod

Besonderheiten von Klassen in Python

- Definition mit class KLASSE (ist_abgeleitet_von):
- Constructor: __init__(self, ...):
- kein Destructor; Entfernung überflüssiger Okjekte durch Garbage Collection
- self und cls - erster Parameter von Instanz-Methoden bzw. Klassenmethoden zeigen auf die eigene Instanz bzw. Klasse
- Decoratoren: @classmethod und @staticmethod

Besonderheiten von Klassen in Python

- Definition mit class KLASSE (ist_abgeleitet_von):
- Constructor: __init__(self, ...):
- kein Destructor; Entfernung überflüssiger Okjekte durch Garbage Collection
- self und cls - erster Parameter von Instanz-Methoden bzw. Klassenmethoden zeigen auf die eigene Instanz bzw. Klasse
- Decoratoren: @classmethod und @staticmethod

Besonderheiten von Klassen in Python

- Definition mit class KLASSE (ist_abgeleitet_von):
- Constructor: __init__(self, ...):
- kein Destructor; Entfernung überflüssiger Okjekte durch Garbage Collection
- self und cls - erster Parameter von Instanz-Methoden bzw. Klassenmethoden zeigen auf die eigene Instanz bzw. Klasse
- Decoratoren: @classmethod und @staticmethod

weiter: Besonderheiten von Klassen in Python

- Operator-Funktionen:
  __add__(self,other), __radd__(self,other)
  __sub__(self,other), __rsub__(self,other)
  __mul__(self,other), __rmul__(self,other)
  __div__(self,other), __rdiv__(self,other)
  definieren die Funktion +, -, * und /
  z = x + y
- andere Funktionen sind z.B.
  __cmp__, __len__, __repr__, __lt__,

weiter: Besonderheiten von Klassen in Python

- Operator-Funktionen:
  __add__(self,other), __radd__(self,other)
  __sub__(self,other), __rsub__(self,other)
  __mul__(self,other), __rmul__(self,other)
  __div__(self,other), __rdiv__(self,other)
  definieren die Funktion +, -, * und /
  z = x + y
- andere Funktionen sind z.B.
  __cmp__, __len__, __repr__, __lt__,

```
1   #!/usr/bin/env python
2   #-*- coding: utf-8 -*-
3
4   import os, sys
5
6   class Test1:      # Basisklasse
7       className = 'Test1'      # Attribut
8       classCount = 0           # Attribut
9       @classmethod             # Decorator
10      def addCount(cls):       # Klassen-Methode
11          cls.classCount += 1
12
13      def __init__(self, name = None):    # Constructor
14          if name == None:
15              self.addCount()
16              self.name = '%s_%i'%(self.className, self.classCount)
17          else:
18              self.name = name
19
20      def getName(self):                  # Methode
21          return self.name
22
23  class Test2(Test1):      # abgeleitete Klasse
24      className = 'Test2'
25      classCount = 0
```

```
1   def main():
2       # Instanzen bilden
3       t1 = Test1('myT1')
4       t2 = Test1()
5       t3 = Test2()
6       t4 = Test2()
7       t5 = Test1()
8       t6 = Test1()
9
10      # Namen der Instanzen ausgeben
11      print 'T1 = ', t1.getName()
12      print 'T2 = ', t2.getName()
13      print 'T3 = ', t3.getName()
14      print 'T4 = ', t4.getName()
15      print 'T5 = ', t5.getName()
16      print 'T6 = ', t6.getName()
17
18      # die Inhalte der Klassen und Instanzen ausgeben
19      print vars(Test1)
20      print vars(Test2)
21      print vars(t1)
22      print vars(t3)
23      print vars(t4)
24      print vars(t6)
25
26  if __name__ == "__main__":
27      main()
```

Beispiel: Module pyparsing

- pyparsing erzeugt auf einfache Weise Textparser
- angelehnt an die Backus-Naur-Form
- geeignet für einfache Textersetzung bis zum Parsen einer vollständigen Programmiersprache
- macht ausgiebig von den Möglichkeiten der OOP gebrauch
- kreativer Gebrauch von Operator-Überladung um Objekte zusammen zu fügen

+	Sequenz
|	erste Alternative
^	längste Alternative
*	Wiederholung
«	Zuweisung für Forward-Typ

Beispiel: Module pyparsing

- pyparsing erzeugt auf einfache Weise Textparser
- angelehnt an die Backus-Naur-Form
- geeignet für einfache Textersetzung bis zum Parsen einer vollständigen Programmiersprache
- macht ausgiebig von den Möglichkeiten der OOP gebrauch
- kreativer Gebrauch von Operator-Überladung um Objekte zusammen zu fügen

  `+` Sequenz
  `|` erste Alternative
  `^` längste Alternative
  `*` Wiederholung
  `«` Zuweisung für Forward-Typ

Beispiel: Module pyparsing

- pyparsing erzeugt auf einfache Weise Textparser
- angelehnt an die Backus-Naur-Form
- geeignet für einfache Textersetzung bis zum Parsen einer vollständigen Programmiersprache
- macht ausgiebig von den Möglichkeiten der OOP gebrauch
- kreativer Gebrauch von Operator-Überladung um Objekte zusammen zu fügen

+	Sequenz
|	erste Alternative
^	längste Alternative
*	Wiederholung
«	Zuweisung für Forward-Typ

Beispiel: Module pyparsing

- pyparsing erzeugt auf einfache Weise Textparser
- angelehnt an die Backus-Naur-Form
- geeignet für einfache Textersetzung bis zum Parsen einer vollständigen Programmiersprache
- macht ausgiebig von den Möglichkeiten der OOP gebrauch
- kreativer Gebrauch von Operator-Überladung um Objekte zusammen zu fügen

+	Sequenz
|	erste Alternative
^	längste Alternative
*	Wiederholung
«	Zuweisung für Forward-Typ

Beispiel: Module pyparsing

- pyparsing erzeugt auf einfache Weise Textparser
- angelehnt an die Backus-Naur-Form
- geeignet für einfache Textersetzung bis zum Parsen einer vollständigen Programmiersprache
- macht ausgiebig von den Möglichkeiten der OOP gebrauch
- kreativer Gebrauch von Operator-Überladung um Objekte zusammen zu fügen

+	Sequenz
|	erste Alternative
^	längste Alternative
*	Wiederholung
«	Zuweisung für Forward-Typ

Beispiel: Module pyparsing

- Komma ::= , $\Longrightarrow$ Komma = Literal(',')
- Ziffer ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
  Buchstabe ::= a | b | c ... X | Y | Z
  Int ::= <Ziffer> | <Ziffer> <Int>
  wird zu:
  Ziffer = Word(nums, max=1)
  Buchstabe = Word(alphas,max=1)
  Int = Ziffer | ZeroOrMore(Ziffer)

Beispiel: Module pyparsing

- Komma ::= , $\Longrightarrow$ Komma = Literal(',')
- Ziffer ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
  Buchstabe ::= a | b | c ... X | Y | Z
  Int ::= <Ziffer> | <Ziffer> <Int>
  wird zu:
  Ziffer = Word(nums, max=1)
  Buchstabe = Word(alphas,max=1)
  Int = Ziffer | ZeroOrMore(Ziffer)

```
1   def inifile_BNF():
2       # punctuation
3       lbrack = Literal("[").suppress()        # erscheint nicht in der Ausgabe
4       rbrack = Literal("]").suppress()
5       equals = Literal("=").suppress()
6       fence  = Literal("#")
7
8       comment = fence + Optional(restOfLine)
9
10      nonrbrack = "".join([c for c in printables if c != "]" ]) + " \täöüßÄÖÜ"
11      nonequals = "".join([c for c in printables if c != "=" ]) + " \täöüßÄÖÜ"
12
13      sectionDef = lbrack + Word( nonrbrack ) + rbrack
14      keyDef = ~lbrack + Word(nonequals) + equals + restOfLine
15
16      # using Dict will allow retrieval of named data fields as
17      # attributes of the parsed results
18      inibnf = Dict( ZeroOrMore(Group(sectionDef + Dict(ZeroOrMore(Group(keyDef))))))
19
20      inibnf.ignore(comment)
21
22      return inibnf
```

```
1   def test( strng ):
2       bnf = inifile_BNF()
3       tokens = bnf.parseString(strng)
4
5       return tokens
6
7   input = """[INIT]␣␣␣#␣Kommentar
8   var1=Test
9   vär2=34
10  var3=23.45
11  [Application]
12  start=1
13  dir=/home/vt4/ma
14  """
15
16  ini = test(input)
17  for item in ini:
18          print item
19
20  print 'Wert␣von␣INIT/var3␣=␣', ini['INIT']['var3']
```

```
1   def makeLogoParser():
2       Logo = Forward()
3       words = Word(alphanums+'!§$%/()=?\_.:,;#@~*^°<>—äüöÄÜÖß')
4       wordsall = Word(alphanums+'!§$%&/()=?\_.:,;#@~*^°<>—äüöÄÜÖß')
5       KOMMA = Suppress(',')
6       Punkt = Literal('.')
7       PLUS = Literal('+')
8       MINUS = Literal('—')
9
10      LINKS   = CaselessKeyword('links')
11      RECHTS = CaselessKeyword('rechts')
12      VOR     = CaselessKeyword('vor')
13      ZURUCK = CaselessKeyword('zuruck')
14      GEHEZU = CaselessKeyword('gehezu')
15      COMMANDS = LINKS | RECHTS | VOR | ZURUCK | GEHEZU
16
17      FOR     = CaselessKeyword('for')
18      ENDFOR = CaselessKeyword('endfor')
19
20      INT = Word(nums)
21      INTEGER = Combine(Optional(MINUS|PLUS) + INT).setParseAction(convInt)
22
23      ARNAME = Word(alphas+'_',alphanums+'_')
24      GLEICH = Literal('=')
25
26      VALUE = INTEGER | VARNAME
27      EXPR = Forward()
28      EXPR << (operatorPrecedence(VALUE, [
```

```
29                          (oneOf('! -'), 2, opAssoc.RIGHT),
30                          (oneOf('* / %'), 2, opAssoc.LEFT),
31                          (oneOf('+ -'), 2, opAssoc.LEFT),
32                          ]))
33
34      FUNCTION = Group(COMMANDS + Optional(Group(delimitedList(EXPR))))
35      DECL = Group(VARNAME + GLEICH + FUNCTION)
36
37      FORSTMNT = Forward()
38      FORSTMNT = Group(FOR + VARNAME + GLEICH + delimitedList(EXPR) + Group(ZeroOrMore(FUNCTION
39
40      Logo = OneOrMore(FUNCTION | DECL | FORSTMNT)
41
42      return Logo
```

Main

```
1   def main():
2       t = """gehezu 100,100
3   rechts 90
4   vor 100
5   rechts 90
6   vor 100
7   rechts 90
8   VOR 100
9   rechts 90
10  vor 100
11  for i=0,360,30
12  vor i/10+(3*i/10)
13  rechts i
14  endfor
15  """
16
17      lines = t.split('\n')
18      iniData = chr(10).join(lines)
19
20      logo = makeLogoParser()
21
22      res = logo.parseString(t)
```

Ergebnis

```
1   gehezu 100,100
2   rechts 90
3   vor 100
4   rechts 90
5   vor 100
6   rechts 90
7   VOR 100
8   rechts 90
9   vor 100
10  for i=0,360,30
11    vor i/10+(3*i/10)
12    rechts i
13  endfor
14
15  ['gehezu', [100, 100]]
16  ['rechts', [90]]
17  ['vor', [100]]
18  ['rechts', [90]]
19  ['vor', [100]]
20  ['rechts', [90]]
21  ['vor', [100]]
22  ['rechts', [90]]
23  ['vor', [100]]
24  ['for', 'i', '=', 0, 360, 30,
25    [['vor', [['i', '/', 10], '+', [3, '*', 'i', '/', 10]]], ['rechts', ['i']]], 'endfor']
```

Ergebnis

```
 1  COMMAND <gehezu> Parameter: 100, 100
 2  COMMAND <rechts> Parameter: 90
 3  COMMAND <vor> Parameter: 100
 4  COMMAND <rechts> Parameter: 90
 5  COMMAND <vor> Parameter: 100
 6  COMMAND <rechts> Parameter: 90
 7  COMMAND <vor> Parameter: 100
 8  COMMAND <rechts> Parameter: 90
 9  COMMAND <vor> Parameter: 100
10    COMMAND <vor> Parameter: 0
11    COMMAND <rechts> Parameter: 0
12    COMMAND <vor> Parameter: 12
13    COMMAND <rechts> Parameter: 30
14      .
15      .
16      .
17    COMMAND <vor> Parameter: 120
18    COMMAND <rechts> Parameter: 300
19    COMMAND <vor> Parameter: 132
20    COMMAND <rechts> Parameter: 330
```

Beispiel: Module kmcurses

- eine vollständige Klassenhierachie zur einfachen Nutzung des NCurses-Modules
- Eigenentwicklung, um einen einfacheren Zugang zu NCurses zu haben
- Durch Kapselung der NCurses-Funktionen lässt sich die fehlerträchtige Programmierung umgehen
- Viele Elemente auf Grafik-Gui's implementiert

Beispiel: Module kmcurses

- eine vollständige Klassenhierachie zur einfachen Nutzung des NCurses-Modules
- Eigenentwicklung, um einen einfacheren Zugang zu NCurses zu haben
- Durch Kapselung der NCurses-Funktionen lässt sich die fehlerträchtige Programmierung umgehen
- Viele Elemente auf Grafik-Gui's implementiert

Beispiel: Module kmcurses

- eine vollständige Klassenhierachie zur einfachen Nutzung des NCurses-Modules
- Eigenentwicklung, um einen einfacheren Zugang zu NCurses zu haben
- Durch Kapselung der NCurses-Funktionen lässt sich die fehlerträchtige Programmierung umgehen
- Viele Elemente auf Grafik-Gui's implementiert

Beispiel: Module kmcurses

- eine vollständige Klassenhierachie zur einfachen Nutzung des NCurses-Modules
- Eigenentwicklung, um einen einfacheren Zugang zu NCurses zu haben
- Durch Kapselung der NCurses-Funktionen lässt sich die fehlerträchtige Programmierung umgehen
- Viele Elemente auf Grafik-Gui's implementiert

```
1  #!/usr/bin/env python
2  #-*- coding: utf-8 -*-
3
4  from kmcurses import *
5
6  def main():
7      menu = ['&Erster Eintrag:Weiter zum 1. Dialog', \
8              '&Zweiter Eintrag:Weiter zu 2. Dialog', \
9              '&Dritter Eintrag:Weiter zu 3. Dialog',
10             '', "E &X I T:Ende des Menu's"]
11
12     app = Window(None, 0, 0)
13     menu = Menu(app, menu)
14
15     menuitem = menu.run()
16
17     print 'MenuItem : ', menuitem
18
19     app.exitCurses(app.scr)
20
21     print 'MenuItem = ', menuitem
22
23 if __name__ == "__main__":
24     SecureRun(main)
```

```
1   #!/usr/bin/env python
2   #-*- coding: utf-8 -*-
3
4   from kmcurses import *
5
6   def main():
7       app = Window(None, 0, 0)
8
9       li = ['kein', 'Dipl.-Ing.', 'Ms.Sc.', 'Dr.rer.nat.', 'Dr.-Ing.', 'Prof. Dr.', 'Blödmann']
10      user = Dialog(app, 65, 22)
11      user.insertItem(ComboBox(user, 'titel', 13, 1, 15, 15, 'Titel:', li, 'kein', 'Bitte den Ti
12      user.insertItem(dateEditBox(user, 'datum', 43, 1, 10, 10, 'Datum:', ''))
13      user.insertItem(editBox(user, 'vorname', 13, 3, 15, 40, 'Vorname:', '', 'Vorname des Users
14      user.insertItem(editBox(user, 'nachname', 43, 3, 20, 40, 'Nachname:', '', 'Nachname des Us
15      li = ['Mitarbeiter', 'WiMi', 'HiWi', 'Diplomant', 'Studienarbeiter', 'Professor', 'Gast']
16      user.insertItem(ComboBox(user, 'status', 13, 5, 15, 40, 'Status:', li, 'Mitarbeiter', 'Bitt
17      li = ['R207','R209', 'R210', 'R211', 'R212', 'R213', 'R214', 'R215', 'R216','R221']
18      user.insertItem(ComboBox(user, 'raum', 43, 5, 20, 45, 'Raum:', li, 'R210', 'Wo sitzt die P
19      user.insertItem(editBox(user, 'dienstTel', 13, 7, 15, 40, 'Tel.dienst.:', '', 'Telefon-Nr.
20      user.insertItem(editBox(user, 'privatTel', 43, 7, 20, 40, 'Tel.priv.:', '', 'private Telefo
21      user.insertItem(editBox(user, 'email', 13, 9, 30, 40, 'EMail: ', '@tu-harburg.de', 'EMail
22      user.insertItem(editBox(user, 'betreuer', 13, 11, 30, 40, 'Betreuer:', '', 'Ansprechpartne
23      user.insertItem(editBox(user, 'arbeit', 13, 13, 47, 100, 'Arbeit:', '', 'Welche Arbeit sol
24      user.insertItem(editBox(user, 'thema', 13, 15, 47, 100, 'Thema:', '', 'Thema der Arbeit'))
25      user.insertItem(SecretEditBox(user, 'passwd1', 13, 17, 15, 30, 'Password:', '', 'Password
26      user.insertItem(SecretEditBox(user, 'passwd2', 43, 17, 15, 30, 'wiederholen: ', '', 'Bitte
27
28      OK = Button(user, 'ok1', 19, 20, '&OK')
```

```
29      Cancel = Button(user, 'cancel1', 42, 20, '&Cancel')
30
31      user.insertItem(OK, 'o')
32      user.setRetItem(OK)
33      user.insertItem(Cancel, 'c')
34      user.setRetItem(Cancel)
35
36      user.drawWindow()
37      res = user.run()
38
39      return res
40
41  if __name__ == "__main__":
42      res = SecureRun(main)  # ist notwendig für ein geortetes Beenden von NCurses
43
44      print 'Result = ', res
45      print
46      print 'Password = ', res['passwd1']
```

Zusammenfassung

- Die OOP ist eine einfache Möglichkeit der Strukturierung
- Die OOP bietet Möglichkeiten den Programm-Code erheblich zu vereinfachen
- Objekte als Abstraktions-Schicht zur Kapselung fehlerträchtiger Programmteile
- Erzeugung von vielfach nutzbaren Klassen und Modulen

Zusammenfassung

- Die OOP ist eine einfache Möglichkeit der Strukturierung
- Die OOP bietet Möglichkeiten den Programm-Code erheblich zu vereinfachen
- Objekte als Abstraktions-Schicht zur Kapselung fehlerträchtiger Programmteile
- Erzeugung von vielfach nutzbaren Klassen und Modulen

Zusammenfassung

- Die OOP ist eine einfache Möglichkeit der Strukturierung
- Die OOP bietet Möglichkeiten den Programm-Code erheblich zu vereinfachen
- Objekte als Abstraktions-Schicht zur Kapselung fehlerträchtiger Programmteile
- Erzeugung von vielfach nutzbaren Klassen und Modulen

Zusammenfassung

- Die OOP ist eine einfache Möglichkeit der Strukturierung
- Die OOP bietet Möglichkeiten den Programm-Code erheblich zu vereinfachen
- Objekte als Abstraktions-Schicht zur Kapselung fehlerträchtiger Programmteile
- Erzeugung von vielfach nutzbaren Klassen und Modulen

Danke