Refactoring Raumplaner-Modell

Das Raumplanerprojekt hat einen Stand, bei dem man interaktiv im ShellFrame Objekte erzeugen und manipulieren kann. Es fehlen aber noch zwei Anforderungen an eine sinnvolle Anwendung:

- Das Projekt merkt sich die erzeugten Objekte und kann sie abspeichern – spätestens bevor sie beendet wird – damit diese später bei einem erneuten Start geladen werden können.
- Es gibt eine Benutzeroberfläche, die eine Interaktion mit der laufenden Anwendung ohne den ShellFrame ermöglicht.

Für eine erfolgreiche Modellierung und Umsetzung der Kommunikation zwischen den beteiligten Objekten ist eine klare Trennung zwischen den Modellkomponenten und den Visuellen(View-) Komponenten wichtig. Wir orientieren uns dabei am ***model-view-controller*** – Konzept (MVC).

Modell

- enthält die vollständige Zustands-, Daten- und Anwendungslogik
  Diese Aussage zeigt, dass unserer Anwendung im Modellteil noch die Fähigkeiten zum Abspeichern und Laden hinzugefügt werden müssen.
  Das Modell informiert die Viewkomponente (ggf den Controller), wenn sich sein Zustand geändert hat[1].
  Die Umsetzung des Beobachtermusters ermöglicht eine klare Entkopplung des Modells von der Art der Viewkomponente: Es muss nur wissen, wer informiert werden muss.

View

- enthält alles, was zur Darstellung des Zustands und zur Benutzerinteraktion mit der laufenden Anwendung notwendig ist
  Das Grafikfenster ist also Teil der Viewkomponente, die zu entwickelnde GUI, entweder integriert oder gesondert, gehört ebenfalls dazu.
  Die Viewkomponente reicht ihre Anforderungen an den Controller weiter.
  Sie holt sich Informationen über dessen Zustand vom Modell.

Controller

- stellt einen konsistenten Zustand des Systems sicher, indem er die Anforderungen der Viewkomponente (und ggf weitere) an das Modell weiterreicht
  Er fordert ggf zur Änderung der Darstellung auf.

Refactoring der Modellkomponente

Ob eine kleine Anwendung tatsächlich eine Controllerklasse realisiert, muss von Fall zu Fall entschieden werden. Durch eine Überarbeitung unseres bisherigen Projekts ermöglichen wir eine saubere Schnittstelle. Sie hat den Charakter eines Controllers, übernimmt aber auch beispielsweise die Verwaltung der aktuell verwendeten Objekte und die Datensicherung und Wiederherstellung.
Wir bezeichnen die Klasse als MoebelModell und fügen sie unserem Projekt hinzu[2].

1 Zu klären ist, ob sie auch zur Aktualisierung der Darstellung auffordert oder das dem Controller überlässt.
2 Zu einer anderen Variante siehe die Präsentation ***Raumplaner MVC Programm***

Die Klasse MoebelModell

In der Klasse werden alle Modellaspekte realisiert, also auch

- das Verwalten der erzeugten Moebelobjekte
- das Auswählen eines Objektes
- das Speichern und Laden des gesamten Zustands

Konstruktor

Die Klasse wird wieder als Singleton modelliert. Im Konstruktor benötigen wir außerdem die beiden Zeilen

```
        self.__alleMoebel = []
        self.__ausgewaehlt = -1
```

also eine zunächst leere Liste für die aktuell erzeugten Moebelobjekte und eine Zahl zur Kennzeichnung des aktuell ausgewählten Moebelobjekts. Der Wert -1 zeigt an, dass keines ausgewählt ist.

Erzeugen eines Moebelobjekts

Die methode Neu ist im Kopf so definiert, dass sie eine beliebige (sinnvolle) Anzahl von Parameterwerten (arguments) und Schlüsselwortparamtern (keyword-arguments) verarbeiten kann, um alle wünschenswerten Aufrufe verarbeiten zu können.

```
    def Neu(self, klasse, *par, **kwargs):
        if klasse.__class__==str:
            klasse=eval(klasse)
        neu=klasse(*par, **kwargs)
        self.__alleMoebel.append(neu)
        return neu
```

Ein Aufruf **`Neu("Tisch", 20, 20, farbe="green")`** ist dadurch zulässig.

Speichern als Objektdaten mit dem Modul pickle

Um das möglich zu machen, muss das Modul pickle importiert werden. Wie bei allen Zugriffen auf Dateien, muss auch hier gewährleistet werden, dass mögliche Fehler verarbeitet werden und darauf reagiert werden kann.

```
    def Speichere(self, dateiname):
        """Objektdaten werden in Datei geschrieben"""
        try:
            with open(dateiname, 'wb') as objekt_datei:
                pickle.dump(self.__alleMoebel, objekt_datei)
                return 'OK'
        except IOError as ioerr:
            return 'Dateifehler: ' + str(ioerr)
        except pickle.PicklingError as perr:
            return 'Pickle-Fehler: ' + str(perr)
```

`dump` ist die pickle-Anweisung zum Speichern, with sichert das ordnungsgemäße Schließen der Datei auch nach Fehlern. Es ist möglich, die Liste der Objekte in einem Schreibvorgang abzuspeichern.

Lesen aus der Datei und Erzeugen der Objekte

In dieser Version werden zunächst erst alle vorhandenen Moebelobjekte verborgen, anschließend gelöscht.
Dann erst werden die in der Datei abgespeicherten Moebelobjekte wieder hergestellt. Dazu ist es zwingend notwendig, dass

- die Klassen im aktuellen Ordner existieren
- die Klassendefinition inzwischen nicht geändert wurde.

```
def Lade(self, dateiname):
    """Objektdaten werden aus der Datei gelesen"""
    for moebel in self.__alleMoebel:
        moebel.Verberge()
        del(moebel)
    self.__ausgewaehlt = -1
    try:
        with open(dateiname, 'rb') as objekt_datei:
            self.__alleMoebel = pickle.load(objekt_datei)
            for moebel in self.__alleMoebel:
                if moebel.GibSichtbar():
                    moebel.Zeige()
            return 'OK'
    except IOError:
        return 'Datei nicht auffindbar!'
    except AttributeError as ae:
        return 'AttributeError: ' + str(ae)
    except pickle.UnpicklingError as perr:
        return 'Pickle-Fehler: ' + str(perr)
```

Methoden für die Auswahl

Die Methode Waehle sieht kompliziert aus, sollte aber selbst erklärend sein.

```
    def Waehle(self,index=None):
        '''wählt das Moebel aus;
        None zum Weitersetzen der Auswahl, -1 zum Beenden der Auswahl,
        nach dem letzten wird die Auswahl ebenfalls beendet'''
        if len(self.__alleMoebel)==0: # nichts auszuwaehlen
            return
        if index!=None: # Indexfehler pruefen
            if index<-1: return
            if index>=len(self.__alleMoebel): return
        if index==-1: # Beenden der Auswahl
            aktuell = self.__ausgewaehlt
            self.__ausgewaehlt = -1
            if aktuell!=-1: # Darstellung aktualisieren
                self.__alleMoebel[aktuell].Waehle(False)
                self.__alleMoebel[aktuell].Zeige()
            return
        if index==None: # Weitersetzen
            if self.__ausgewaehlt!=-1:
                self.__alleMoebel[self.__ausgewaehlt].Waehle(False)
                self.__alleMoebel[self.__ausgewaehlt].Zeige()
            self.__ausgewaehlt+=1
            if self.__ausgewaehlt==len(self.__alleMoebel):
                self.__ausgewaehlt=-1
                return
            self.__alleMoebel[self.__ausgewaehlt].Waehle(True)
            self.__alleMoebel[self.__ausgewaehlt].Zeige()

    def GibAusgewaehltes(self):
        if self.__ausgewaehlt<0: return None
        else: return self.__alleMoebel[self.__ausgewaehlt]

    def LoescheAusgewaehltes(self):
```

```
        if self.__ausgewaehlt>=0:
            self.GibAusgewaehltes().Verberge()
            self.__alleMoebel.remove(self.GibAusgewaehltes())
            self.__ausgewaehlt=-1

    def BearbeiteAusgewaehltes(self, **kwargs):
        """die kwargs dx,dy,dw,f erzeugen ein Dictionary"""
        if self.__ausgewaehlt>=0:
            gewaehlt=self.GibAusgewaehltes()
            if 'dx' in kwargs:
                gewaehlt.BewegeHorizontal(int(kwargs['dx']))
            if 'dy' in kwargs:
                gewaehlt.BewegeVertikal(int(kwargs['dy']))
            if 'dw' in kwargs:
                gewaehlt.Drehe(int(kwargs['dw']))
            if 'f' in kwargs:
                gewaehlt.AendereFarbe(kwargs['f'])
```

GibAlleMoebel

Die einzige nun noch fehlende Methode ist die Get-Methode für alle Moebelobjekte.

```
    def GibAlleMoebel(self):
        return self.__alleMoebel
```

Schnittstelle

Die Klasse MoebelModell definiert die einzige Schnittstelle zum Modell. Über die Methode GibAusgewahltes() kann der Benutzer auf von ihr bereit gestellte Objekte mit deren Methoden zugreifen.
Ein Beispiel für eine Interaktion über die PyShell:

```
>>> modell=app.modell
>>> modell.Neu(Schrank,280, 40, 100, 60, 90, 'black', True)
>>> modell.Waehle()
>>> modell.GibAusgewaehltes().BewegeHorizontal(-100)
```

Der Schrank wird erzeugt, ausgewählt und bewegt.