7.2 Abstrakte Klassen und Methoden

Freitag, 12. Mai 2023 17:31

7.2. Abstrakte Klassen & Methoden

Abstrakte Klasse:
Eine abstrakte Klasse ist eine eingeschränkte Klasse, da sie nicht instanziiert werden kann - Sie können sie nicht zum Erstellen von Objekten verwenden. Es kann nur von einer anderen Klasse geerbt werden.

Eine abstrakte Klasse zielt darauf ab, eine gemeinsame Funktion/ein gemeinsames Verhalten zu definieren, das mehrere Unterklassen erben können, ohne die gesamte abstrakte Klasse implementieren zu müssen.

Abstrakte Klasse erstellen

1. Modul importieren

```
from abc import ABC, abstracmethod
```

2. Deklarieren der abstrakten Klasse

```
class class_name(ABC):
    ...
```

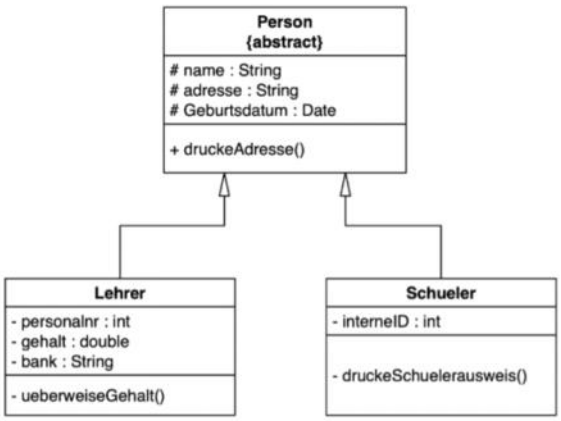
Abstrakte Methoden

Eine **abstrakte Methode** besitzt keinen Methodenrumpf und dementsprechend auch keine Anweisungen. Ihr Zweck liegt ausschließlich darin, jede Unterklasse dazu zu zwingen diese Methoden zu überschreiben. Damit ist sichergestellt, dass alle Unterklassen eine Methode mit dieser Signatur implementieren.

Eine Klasse, die mindestens eine abstrakte Methode enthält, muss selbst als abstrakte Klasse gekennzeichnet werden.

Im **UML-Klassendiagramm** wird eine abstrakte Methode entweder **kursiv** dargestellt oder um die Eigenschaft **{abstract}** ergänzt.

➔ **Darstellung im Klassendiagramm:**



7.3 Einfachvererbung

Freitag, 12. Mai 2023 17:31

7.3. Einfachvererbung

Wiederholung OOP:

Konto
– kontoNr: int – besitzer: string – kontostand: double
+ Konto() + Einzahlen(double): void + Auszahlen(double): void + Überzogen(): bool

Handlungsauftrag:

1. Erstellen Sie die oben dargestellte Klasse.
2. Für die Erzeugte Klasse, soll nun in einem Hauptprogramm ein Kontoerzeugt werden.
3. Mit Hilfe eines Menüs sollen Ein- und Auszahlungen vorgenommen sowie Kontodaten abgefragt werden.
    a. Zuerst wird das Konto erzeugt und mit Daten initialisiert. Auf das Konto wird dann über den Namen und die entsprechende Methode zugegriffen.

Hinweis: Sie können die Klasse Konto mit dem import Befehl nutzen. from Konto import Konto

Das Konzept der Vererbung erlaubt es uns spezialisiertere Klassen einer allgemeinen Klasse zu erstellen. Die spezialisierte Klasse soll dabei alle Eigenschaften der allgemeinen Klasse besitzen, so dass nur noch wenige Eigenschaften hinzugefügt werden müssen. Die Klasse, von welcher geerbt wird, nennt man **Oberklasse**, **Superklasse** oder **Basisklasse** und die Klasse, welche erbt, wird **Unterklasse**, **abgeleitete Klasse** oder **Subklasse** genannt.

So könnte z.B. die Klasse **Fahrzeug** eine Oberklasse der Unterklassen **Personenwagen** und **Lastwagen** sein. Jedes **Fahrzeug** ist durch die marke, den hubraum und die leistung charakterisiert. Bei den **Personenwagen** möchte man noch zusätzlich wissen, wie viele Sitzplätze es hat und beim **Lastwagen** wie schwer die Fracht sein darf.

Um im Programm zu deklarieren, dass die Klasse **Personenwagen** von der Klasse **Fahrzeug** erbt, setzt man beim Namen der Unterklasse einfach den Namen der Oberklasse in Klammern (**Personnenwagen(Fahrzeug)**). In einem Programm könnte das folgendermaßen aussehen:

```
1   class Fahrzeug:
2       def __init__(self, marke, hubraum, leistung):
3           self.marke = marke
4           self.hubraum = hubraum
5           self.leistung = leistung
6
7       def get_infos(self):
8           return "Marke: " + self.marke + ", Hubraum: " + \
9       str(self.hubraum) + ", Leistung: " + str(self.leistung)
10
11  class Personenwagen(Fahrzeug):
12      pass
13
14  class Lastwagen(Fahrzeug):
15      pass
```

Testen wir die Klasse in der Konsole, dann könnte das so aussehen:

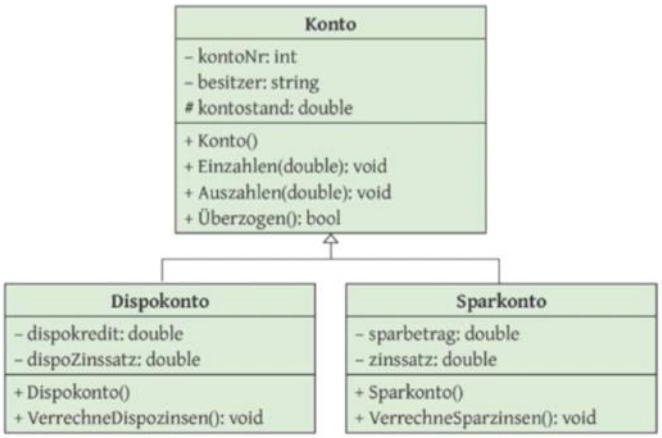
```
>>> pw = Personenwagen("Opel", 222, 100)
>>> lkw = Lastwagen("Mercedes", 5000, 300)
>>> print(pw.get_infos())
Marke: Opel, Hubraum: 222, Leistung: 100
>>> print(lkw.get_infos())
Marke: Mercedes, Hubraum: 5000, Leistung: 300
```

Wir sehen also, dass die beiden Unterklassen alle Eigenschaften, d.h. alle Instanzvariablen und alle Methoden der Oberklasse geerbt haben.

Übung

Handlungsauftrag:

1. Implementieren Sie die untenstehenden Klassen.



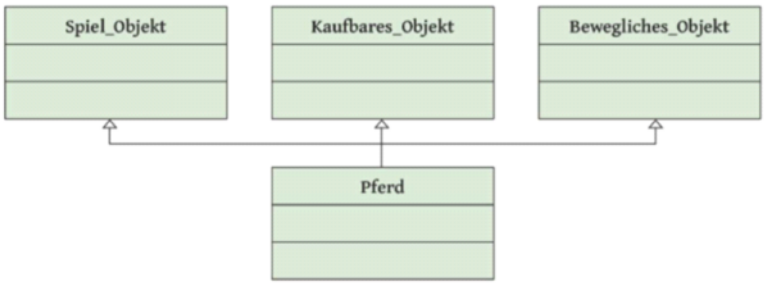
Schon fertig? Erweitern Sie das Hauptprogramm. Es soll nun ein Dispokonto / Sparkonto angelegt werden. Implementieren Sie Menüpunkte für die Verrechnung der Zinsen.

7.4 Mehrfachvererbung

Freitag, 12. Mai 2023 17:31

7.4. Mehrfachvererbung

Eine Klasse kann in Python auch von zwei oder mehreren Klassen gleichzeitig abgeleitet sein. Sie erbt in diesem Fall alle Eigenschaften und Methoden aller Elternklassen.



Im Beispiel wird die Klasse Pferd von der Klasse Spiel_Objekt abgeleitet. Außerdem soll das Pferd ein bewegliches und ein kaufbares Objekt sein. Deswegen erbt die Klasse auch noch von diesen beiden Klassen.

Auftrag:
Implementieren Sie die oben dargestellte Mehrfachvererbung. Jede Klasse soll eine Print-Anweisung ausgeben, die zeigt, um welche Klasse es sich handelt.

7.5 Überschreiben und Überladen

Freitag, 12. Mai 2023 17:31

7.5. Überschreiben und Überladen von Methoden

Neben der Kapselung und Vererbung ist die **Polymorphie** das dritte Basiskonzept der OOP. Das Wort stammt aus dem Griechischen und bedeutet: Vielgestaltigkeit. In diesem Zusammenhang werden zwei Konzepte vorgestellt: Das Überladen und das Überschreiben.

Überladen von Methoden

Im Gegensatz zu anderen Programmiersprachen können Methoden und Funktionen in Python die überladen werden. Wenn eine Methode mehrfach definiert wird, dann gilt immer die letzte Definition dieser Methode. Dies gilt auch, wenn die Methoden unterschiedliche Übergabeparameter besitzen. Methoden können abder in abgeleiteten Klassen überschrieben werden.

Überladen bedeutet:

Derselbe Methodenname kann mehrfach in einer Klasse verwendet werden. Das Überladen findet häufig bei Konstruktoren oder Konvertierungsmethoden statt. Damit das Überladen möglich ist, muss wenigstens eine der folgenden Voraussetzungen erfüllt sein:

1. Der Datentyp mindestens eines Übergabeparameters ist anders als in den übrigen gleichnamigen Methoden.
2. Die Anzahl der Übergabeparameter ist unterschiedlich.

Überschreiben von Methoden

Überschreiben bedeutet:

Überschreiben von Methoden bedeutet, dass eine Methode einer Elternklasse in der vererbten Klasse neu implementiert (überschrieben) wird. Durch Vererbung übernimmt eine neue Klasse alle öffentlichen (public) und geschützten (protected) Eigenschaften und Methoden der Elternklasse.

Der neuen Klasse können nun beliebige Methoden und Eigenschaften hinzugefügt werden.

Wenn eine Methode der neuen Klasse den gleichen Namen wie eine geerbte Methode aufweist und neu implementiert wird, dann spricht man vom Überschreiben

Beispiel:

```
1  class Rechteck():
2  def zeichne(self):
3  pass

4  class AusgefülltesRechteck(Rechteck):
5  def zeichne(self):
6  pass
```

Im Beispiel erbt die Klasse „AusgefülltesRechteck“ die Methode zeichne() von der Klasse „Rechteck“. Diese wird nun überschrieben und neu implementiert, sodass anstatt eines nicht ausgefüllten Rechteckes ein ausgefülltes Rechteck gezeichnet wird.

Beispiel 2: Das Beispiel zeigt, dass die abgeleiteten Klassen die Eigenschaften der Oberklassen geerbt haben. Nun möchten wir aber noch die speziellen Eigenschaften der Unterklassen im Programm einbauen. Damit sind bei der Klasse **Personenwagen** die Anzahl Sitzplätze und bei der Klasse **Lastwagen** die Schwere der Fracht gemeint.

Um diese Eigenschaften zu implementieren führen wir in den Unterklassen eigene **__init__()** - Methoden ein. Dies hat zur Folge, dass wenn wir ein Objekt der Klasse **Personenwagen** erstellen, nun nicht mehr die **__init__()** - Methoden der Klasse **Fahrzeug** aufgerufen wird, sondern die der Klasse **Personenwagen**.

Um aber Code Duplizität zu vermeiden, können wir in der Unterklasse die **__init__()** - Methode der Oberklasse aufrufen. Dies geschieht mit dem Keyword **super()**.

Ebenfalls *überschreiben* wir die Methode **get_info()** und passen sie auf die entsprechende Unterklasse an.

```
1  class Fahrzeug:
2      def __init__(self, marke, hubraum, leistung):
3          self.marke = marke
4          self.hubraum = hubraum
5          self.leistung = leistung
6
7      def get_infos(self):
8          return "Marke: " + self.marke + ", Hubraum: " + \
9      str(self.hubraum) + ", Leistung: " + str(self.leistung)
10
11 class Personenwagen(Fahrzeug):
12     def __init__(self, marke, hubraum, leistung, anz_plaetze):
13     super().__init__(marke, hubraum, leistung)
14         self.anz_plaetze = anz_plaetze
15
16     def get_infos(self):
17     return super().get_infos() + ", Anzahl Plaetze: " + str(self.anz_plaetze)
18
19 class Lastwagen(Fahrzeug):
20     def __init__(self, marke, hubraum, leistung, last):
21     super().__init__(marke, hubraum, leistung)
22         self.last = last
23
24     def get_infos(self):
25     return super().get_infos() + ", Lastgewicht: " + str(self.last)
26
27 if __name__ == "__main__":
28     pw = Personenwagen("Opel", 222, 100, 5)
29     lkw = Lastwagen("Mercedes", 5000, 300, 2000)
30     print(pw.get_infos())
31     print(lkw.get_infos())
```

Das Programm liefert folgenden Output:

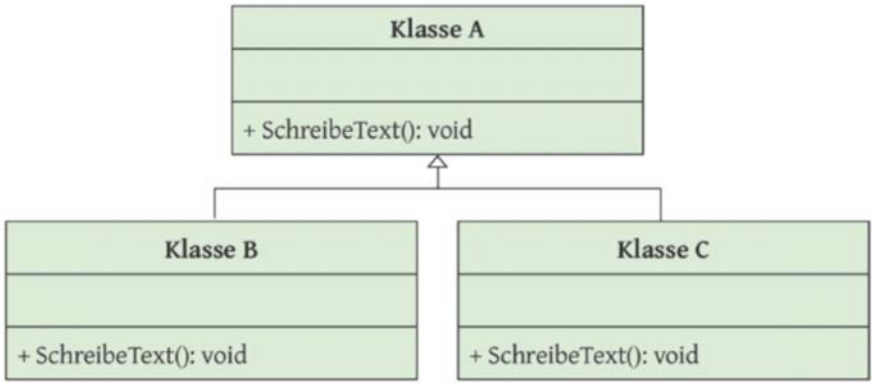
```
>>>
Marke: Opel, Hubraum: 222, Leistung: 100, Anzahl Plaetze: 5
Marke: Mercedes, Hubraum: 5000, Leistung: 300, Lastgewicht: 2000
```


Alternativ kann man auch über den Klassennamen anstelle von `super()` auf die Methoden der Oberklasse zugreifen. Zeile 13 von oben könnte dann folgendermaßen aussehen:

```
Fahrzeug.__init__(self, marke, hubraum, leistung)
```

Auftrag:

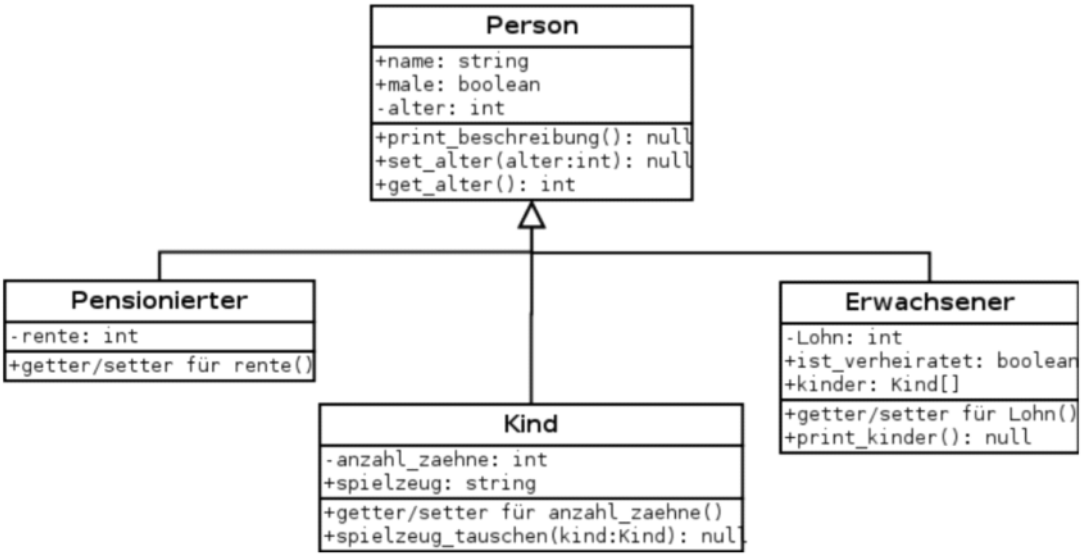
1. Schreiben Sie ein Programm, welches das folgende Klassendiagramm abbildet.
2. Dabei soll die Methode schreibe_text() in der Klasse „A“ definiert werden.
3. Dn den Klasen „B“ und „C“ wird die Methode mit neuen Texten überschrieben.
    1. Klasse B: „Methode der Klasse B
    2. Klasse C: Methode der klasse C



7.6 Übung zur Vererbung

Freitag, 12. Mai 2023 17:31

7.6. Übung zur Vererbung



Hinweise zum Diagramm:

- Der Pfeil bedeutet: *„Erbt von“*
- Im obersten Teilkästchen wird jeweils der Klassenname notiert. Anschließend folgen die Instanzvariablen und zum Schluss die Methoden.
- Instanzvariablen werden gemäß der folgenden Notation illustriert:

  <+/-> <name> : <type>

  wobei ein + für *public* und ein - für *private* steht.
- Analog bei Methoden:

  <+/-> <name>(<parameter>) : <return-type>

a) Implementiere alle Klassen gemäß dem Klassendiagramm.

- Die Methode **print_beschreibung()** soll eine kurze Beschreibung ausgeben:
- *„Ich heiße Hans Muster, bin männlich, 70 Jahre alt und Pensionär(in).“*
- Achte darauf, dass das Alter eines Erwachsenen zwischen 18 und 61 Jahre beträgt. Kinder sind jünger als 18 Jahre und die Pensionierten älter als 61 Jahren.
- Die Funktion **print_kinder()** in der Klasse **Erwachsener** gibt die Namen der Kinder auf der Konsole aus, falls sie überhaupt Kinder besitzt.
- Für die getter und Setter Methoden gilt:
    - Setter: setze den Wert beim Methodenaufruf.
    - Getter: Wert wird zurückgegeben.
- Vermeide Code Duplizität.

8. Klassendiagramm

Das Klassendiagramm bildet das Herzstück der UML. Es basiert auf den Prinzipien der Objektorientierung und ist durch seine Vielseitigkeit in allen Phasen eines Projekts einsetzbar. In der Analysephase tritt es in Erscheinung und versucht ein Abbild der Wirklichkeit darzustellen. In der Designphase wird damit die Software modelliert und in der Implementierungsphase daraus Sourcecode generiert.

In Klassendiagrammen werden Klassen und die Beziehungen von Klassen untereinander modelliert. Bei den Beziehungen kann man grob drei Arten unterscheiden. Die einfachste und allgemeinste Variante ist die Assoziation. Eine zweite modellierbare Beziehung ist die Aufnahme einer Klasse in eine zweite Klasse, die sogenannte Containerklasse. Solche Beziehungen werden Aggregation oder Komposition genannt. Eine dritte Möglichkeit ist die Spezialisierung bzw. Generalisierung.

Da eine Klasse die Struktur und das Verhalten von Objekten, die von dieser Klasse erzeugt werden, modellieren muss, können diese mit Methoden und Attributen versehen werden. Weiterhin ist die Modellierung von Basisklassen und Schnittstellen über Stereotypen möglich.