1. Abstrakte Basisklassen

Eine abstrakte Klasse kann als Blaupause für andere Klassen betrachtet werden. Sie ermöglicht es, eine Reihe von Methoden zu erstellen, die **in allen untergeordneten Klassen implementiert werden müssen** (= Vertrag für die Implememtation), die von der abstrakten Klasse ableiten. Eine Klasse, die eine oder mehrere abstrakte Methoden enthält, wird als abstrakte Klasse bezeichnet. Eine abstrakte Methode ist eine Methode, die eine Deklaration, aber keine Implementierung hat.
Wenn eine gemeinsame Schnittstelle für verschiedene Implementierungen einer Komponente bereitgestellt werden soll, verwenden wir eine abstrakte Klasse.
In der UML wird eine abstrakte Klasse durch `{abstract}` markiert, während eine abstrakte Methode *kursiv* angeschrieben wird.

Beispiel 7.1: Übergeordneter Begriff `Medium` als Träger einer Niederschrift.

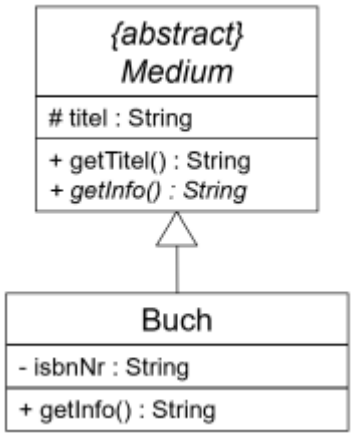


Abb 7.1: UML-Diagramm mit abstrakter Klasse
Die Klasse `Medium` kann nicht instanziert werden, da sie als abstract deklariert ist. Es bedingt immer eine konkrete Ableitung (hier `Buch`) der Klasse.
Die Methode `getInfo` wird als abstrakte Methode deklariert und muss in der abgeleiteten Klasse konkretisiert werden.

Wozu abstrakte Basisklassen verwenden?

Indem eine abstrakte Basisklasse definiert wird, können gemeinsame Anwendungsprogrammschnittstelle (API) für eine Reihe von Unterklassen definiert werden. Diese Funktion ist besonders nützlich in Situationen, in denen ein Drittanbieter Implementierungen bereitstellt, z.B. mit Plugins. Sie kann aber auch helfen, wenn in einem grossen Team oder mit einer grossen Codebasis gearbeitet wird. Eine Situation in der es schwierig oder gar nicht möglich ist, alle Klassen im Kopf zu behalten.

Wie funktionieren abstrakte Basisklassen in Python?

Standardmässig stellt Python keine abstrakten Klassen bereit. Python wird mit einem Modul `abc` geliefert, das die Basis für die Definition von Abstract Base Classes ABC bereitstellt. ABC funktioniert,

indem Methoden der Basisklasse als “abstrakt dekoriert” und dann konkrete Klassen als Implementierungen der abstrakten Basis registriert werden. Eine Methode wird abstrakt, wenn sie mit dem Schlüsselwort `@abstractmethod` dekoriert wird. Zum Beispiel so:

Beisiel 7.2: Abstrakte Klasse für Vielecke (Polygon)

```
from abc import ABC, abstractmethod

class Polygon(ABC):
    """
    Ein unbestimmtes Polygon, das
    a) weiss, dass es eine gewisse Anzahl Seiten hat
    b) aber nicht weiss, wie viele es wirklich sind.
    """
    @abstractmethod
    def __init__(self):
        """
        Konstruktor: Abstrakte Methode um die Instantiierung zu verhindern
        """
        pass


    @abstractmethod
    def my_sides(self):
        """
        Abstrakte Methode ohne Implementierung. Diese muss
        zwingend in einer abgeleiteten Klasse erfolgen.
        """
        pass

class Triangle(Polygon):
    # overriding abstract method
    def my_sides(self):
        print('I have 3 sides')

class Quadrilateral(Polygon):
    # overriding abstract method
    def my_sides(self):
        print('I have 4 sides')


if __name__ == '__main__':
    polygons = [
        Triangle(),
        Quadrilateral(),
    ]
    for p in polygons:
```

```
        p.my_sides()
```

Output:

```
I have 3 sides
I have 4 sides
```

Beachten Sie hier die Nutzung der Polymorphie, um gleichartige Objekte (Triangle, Quadrilateral) gleich zu behandeln - konkret durch den Aufruf der (überschriebenen) Methode `my_sides`.

© Danie Fahrni, René Probst

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/archiv/m320/learningunits/lu99/theorie/lu8-kapitel_2

Last update: **2026/08/30 14:09**