Kompetent objektorientiert programmieren

V1.01- Juli 2019

Aegidius Plüss, Bern, www.aplu.ch

Eine objektorientierte Programmiersprache steht auf drei fundamentalen Säulen:

- **Klassenkapselung (encapsulation)**
- **Vererbung (inheritance)**
- **Polymorphismus (polymorphism)**

Im täglichen Leben fasst man eine Pizza als ein **Objekt** eines bestimmten Lebensmitteltyps auf. Darum ist es natürlich, virtuelle Pizzas ebenfalls als Objekte zu modellieren. Für jede Pizzasorte definierst du eine **Klasse**, in der **Eigenschaften und Verhalten** der zu erzeugenden Objekte festgelegt sind. Wir sagen, dass die Klasse einen **Datentyp** definiert und dass eine bestimmte Pizza ein Objekt oder eine **Instanz** mit diesem Datentyp ist. Eigenschaften werden durch **Instanzvariablen** (oder Attribute) beschrieben und das Verhalten durch **Methoden** (oder Memberfunktionen) festgelegt. Diese können nur unter Bezug auf die Klasse verwendet werden. Zudem kann der Zugriff von Aussen eingeschränkt sein. Man sagt deswegen, dass Eigenschaften und Verhalten der Objekte in ihrer Klasse **gekapselt** sind.

Deine virtuellen Pizzas haben einen gemeinsamen Grundaufbau, aber unterschiedliche Zutaten. Der Grundaufbau jeder Pizza wird in der Klasse *Pizza* beschrieben und besteht aus dem Teig (dough), Tomatenstücken und Mozzarellakäse. Allerdings wird eine solche Pizza nie hergestellt, sondern nur Pizzas, die weitere Zutaten (ingredients) enthalten. Für die tatsächlich hergestellten Pizzasorten definierst du Klassen, die du aus der Klasse *Pizza* **ableitest**: Damit brauchst du die gemeinsamen Eigenschaften und Verhalten und einmal zu programmieren und vermeidest damit die gefährliche Codeduplikation.

Typ, Klassenname	**Zutaten**
Margarita	Oregano
Prosciutto	Schinken
Vegetariana	Gemüse
Napoli	Schinken, Pilze, Oliven

Wie du siehst, ist die Pizza *Napoli* eigentlich eine Pizza *Prosciutto* mit Pilzen und Oliven als zusätzlichen Zutaten. Darum leitest du die Klasse *Napoli* nicht von Pizza, sondern von *Prosciutto* ab.

Den Klassen gibt du die **Fähigkeiten,** sich grafisch darzustellen (statt dass sie in der Küche zubereitet werden). Du verwendest dazu der Einfachheit halber die

Turtlegrafik, könntest aber auch eine andere Grafikbibliothek einsetzen, beispielsweise *GPanel (*in Source-Download enthalten). Jedes Pizza-Objekt soll auch als **Eigenschaft** ihren Namen und ihren Verkaufspreis kennen, der beim Erstellen der Pizza zum Tagespreis in Schweizerfranken angegeben wird. Die Eigenschaften nennt man auch **Instanzvariablen** und sie werden beim Erzeugen der Pizza im **Konstruktor** *__init__()* definiert und initialisiert. Die Fähigkeiten modellierst du mit **Methoden**. Beispielsweise fügt *addIngredients()* die Zutaten hinzu.

Da es nie ein Objekt der Basisklasse *Pizza* geben darf, definierst du sie als eine sogenannte **abstrakte Klasse** und die Methode *drawIngredients()* **als abstrakte Methode**, die durch diese spezielle Auszeichnung in jeder aus Pizza abgeleiteten Klassen definiert werden **muss**. Dies erreichst du in Python, indem du im Kopf von *Pizza* die Zeile

```
__metaclass__ =ABCMeta
```

einfügst und vor der leeren Methode *addIngredients()* die Annotation

```
@abstractmethod
```

setzt. Achte darauf, dass du in jeder Methode einen zusätzlichen ersten Parameter verwenden musst, den man normalerweise *self* nennt (aber der auch irgendeinen anderen Namen haben kann). Mit diesem Parameter kannst du auf die aktuelle Instanz zugreifen, um beispielsweise innerhalb der Klasse Methoden aufzurufen und Instanzvariablen zu definieren, zu lesen und zuzuweisen. Methoden und Variablen *ohne self* gehören zum *globalen Namensraum*. Als *global* bezeichnet, können Variablen sogar auch innerhalb der Klasse definiert und zugewiesen werden, was aber schlechter Programmierstil ist und nach Möglichkeit vermieden werden sollte.

```
# pizza.py

from abc import *
from gturtle import *
from math import *

class Pizza():
    __metaclass__ = ABCMeta

    def makeDough(self):
        setPenColor(makeColor(255, 218, 148))
        setPos(0, 0)
        dot(350)

    def addTomato(self):
        setPenColor(makeColor(200, 50, 0))
        dot(300)

    def addMozzarella(self):
        setPenColor(makeColor(243, 243, 210))
        for i in range(6):
            setPos(100 * sin(i * pi/3), 100 * cos(i * pi/3))
            dot(40)

    def showPrice(self):
        setPenColor("blue")
        setPos(-300, -250)
```

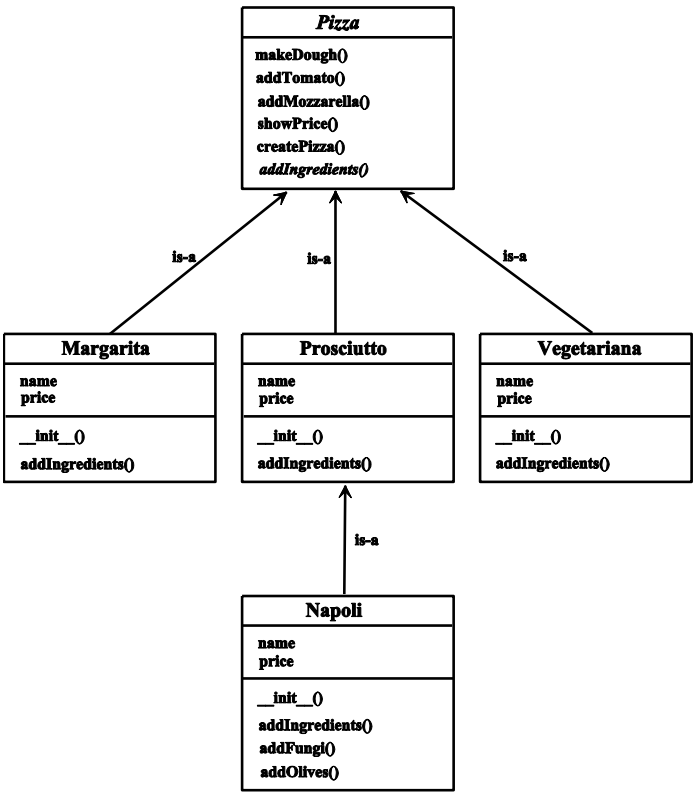
```
        label("Please pay  " + self.price + " Fr.")

    def createPizza(self):
        self.drawDough()
        self.drawTomato()
        self.drawMozzarella()
        self.drawIngredients()
        self.showPrice()
        setTitle(self.name)

    @abstractmethod
    def addIngredients(self):
        pass
```

Du kannst kontrollieren, dass es tatsächlich nicht möglich ist, eine Instanz von *Pizza* zu erzeugen. Fügst du nämlich am Ende von *Pizza.py* die Zeile *p = Pizza()* hinzu, so erhältst du bei der Ausführung die Fehlermeldung

TypeError: Can't instantiate abstract class Pizza with abstract methods addIngredients

Die von der Basisklasse *Pizza* abgeleiteten Klassen **erben** alle ihre Methoden und müssen die Methode *addIngredients()* **überschreiben**. Damit wird garantiert, dass jede Pizzasorte ihre Zutaten auch tatsächlich hinzufügt. Bei der Erzeugung der verschiedenen Pizzasorten initialisiert der Konstruktor das Attribut (die Instanzvariable) *price*.

In einem **Klassendiagramm** wird die Klassenkonstruktion übersichtlich dargestellt (dabei sind abstrakte Elemente kursiv).



Es gehört zum guten Programmierstil, für jede einzelne Klassen eine separate Datei mit dem Klassennamen in Kleinschreibung zu verwenden. Da sich alle Dateien im gleichen Verzeichnis befinden, können sie problemlos importiert werden. Damit ergibt sich ein sauber entkoppeltes und übersichtliches Programmpaket.

```
# margarita.py

from gturtle import *
import random
from pizza import Pizza

class Margarita(Pizza):
    def __init__(self, price):
        self.name = "Pizza Margarita"
        self.price = price

    def addIngredients(self):
        setPenColor(makeColor(23, 67, 0))
        for i in range(50):
            setPos(random.random() * 200 - 100,
                   random.random() * 200 - 100)
            dot(10)
```

```
# prosciutto.py

from gturtle import *
from pizza import Pizza

class Prosciutto(Pizza):
    def __init__(self, price):
        self.name = "Pizza Prosciutto"
        self.price = price

    def addIngredients(self):
        setPenColor(makeColor(230, 87, 84))
        setPos(0, 0)
        dot(155)
```

```
# vegetariana.py

from gturtle import *
from pizza import Pizza

class Vegetariana(Pizza):
    def __init__(self, price):
        self.name = "Pizza Vegetariana"
        self.price = price

    def addIngredients(self):
        setPenColor(makeColor(65, 116, 65))
        setPos(0, 0)
        dot(150)
```

```
# napoli.py
```

```
from gturtle import *
from math import *
from prosciutto import Prosciutto

class Napoli(Prosciutto):
    def __init__(self, price):
        self.name = "Pizza Napoli"
        self.price = price

    def addIngredients(self):
        Prosciutto.drawIngredients(self)
        self.addFungi()
        self.addOlives()

    def addFungi(self):
        setPenColor(makeColor(190, 190, 190))
        setPos(0, 0)
        dot(80)

    def addOlives(self):
        setPenColor("black")
        for i in range(1, 5):
            setPos(60 * sin(i * pi/2), 60 * cos(i * pi/2))
            dot(25)
```

Schliesslich schreibst du in *pizzaiolo.py* das Applikationsprogramm. Dort frägt sozusagen der Pizzabäcker in einem Eingabedialog nach der gewünschten Pizzasorte und stellt dann die Pizza her.

```
# pizzaiolo.py

from gturtle import *
from margarita import Margarita
from prosciutto import Prosciutto
from vegetariana import Vegetariana
from napoli import Napoli

choice = inputString("Margarita, Prosciutto, Vegetariana or Napoli?")
myPizza = None
if choice == "Margarita":
    myPizza = Margarita("15.50")
elif choice == "Prosciutto":
    myPizza = Prosciutto("19.50")
elif choice == "Vegetariana":
    myPizza = Vegetariana("18.00")
elif choice == "Napoli":
    myPizza = Napoli("22.50")
else:
    msgDlg("Sorry, no pizza of this kind!")

if myPizza != None:
    makeTurtle()
    hideTurtle()
    myPizza.createPizza()
```

Es ist interessant, dass beim Erstellen des Programms *pizzaiolo.py* der Datentyp von *myPizza* nicht genau bekannt ist. Man weiss lediglich, dass es sich um einen Typ mit der Basisklasse *Pizza* handelt. Damit bleibt vorerst offen, welche der Methoden

drawIngredients() beim Aufruf von *myPizza.createPizza()* tatsächlich aufgerufen wird. Erst nachdem der Anwender nach Start des Programms seine Pizzawahl getroffen hat, wird die entsprechende Methode ausgewählt. Da erst zu Laufzeit die Variable *myPizza* an den spezifischen Pizzatyp gebunden wird, um die Methode dieses Typs aufzurufen, spricht von einer **dynamischen Typbindung** oder von **Polymorphismus** (vom griechischen "Vielgestaltigkeit").

Download aller Sourcen (für Turtle und GPanel)