Programování 1 (NPRG030)

Cvičení 7: Dědičnost a polymorfismus

Shrnutí přednášky: Dědičnost

```python
from abc import ABC, abstractmethod

class Zvire(ABC):
  def __init__(self, jmeno: str, druh: str):
    self._jmeno = jmeno # chráněný atribut (angl. protected)
    self._druh = druh

  @abstractmethod
  def zvuk(self) -> str:
    pass

  def pozdrav(self) -> str:
    return f"Jsem {self._jmeno}, zvíře druhu {self._druh}"

class Pes(Zvire):
  def __init__(self, jmeno: str, druh: str, rasa: str):
    super().__init__(jmeno, druh) # volání konstruktoru předka
    self.__rasa = rasa            # privátní atribut

  def zvuk(self) -> str:
    return "Haf haf"

  def pozdrav(self) -> str:
    return f"Jsem {self.__jmeno}, pes rasy {self.__rasa}"
```

pokus o vytvoření instance abstraktní třídy způsobí TypeError

deklarace abstraktní metody

při tvorbě instance třídy se konstruktory inicializují v pořadí od předků k potomkům

přepsání metody předka

Cvičení 7.3: Geometrické tvary II

- Rozšiřte program Geometrické Tvary z minulého cvičení

- Vytvořte *třídu*, která *reprezentuje Trojúhelník*, a obsahuje atributy:
  - `strana_a` (datový typ float)
  - `strana_b` (datový typ float)
  - `strana_c` (datový typ float)
- Třída Trojúhelník bude umožňovat *výpočet obvodu*, *obsahu* a *ověření trojúhelníkové nerovnosti*

- Dále vytvořte *společného předka* pro všechny geometrické tvary `GeometrickyTvar` a *deklarujte společné metody*
  - Nezapomeňte společného předka přiřadit každému potomkovi

https://cs.wikipedia.org/wiki/Heronův_vzorec

Cvičení 7.4: Nalezení největšího a nejmenšího tvaru

- Rozšiřte program ze cvičení 7.3 tak, že umožníte *nalezení geometrického tvaru s největším a nejmenším obsahem*
  - Tvary *ukládejte do vhodné kolekce* a výběr kolekce zdůvodněte
    - Můžete využít `list`, `tuple`, `set`, `dict` nebo vlastní kolekce

- Vhodně *využijte polymorfismu* a společného rozhraní (předka) `GeometrickyTvar`

- Na standardní výstup *vypište rozměry spolu s obsahem* nejprve u největšího a následně u nejmenšího tvaru

Příklad 7.5: Poskládání automobilu z dílů

- Napište program v jazyce Python, který umožní *sestavit různé typy automobilů* z různých dílů
  - Každé auto bude mít určenou *značku* a *typ*
  - Pro reprezentaci automobilu, *motoru* a *kol* použijte vhodné třídy.
    - Každý motor bude reprezentován třídou `Motor`, která obsahuje metodu `start` pro spuštění motoru
    - Každá sada kol bude reprezentována třídou `Kola`, která obsahuje metodu `otacet_kola` pro otáčení kol
    - Třída `Auto` bude obsahovat metodu `rozjet_se`, která pustí motor, otáčí kola a uvede auto do pohybu
- Použijte *kompozici namísto dědičnosti* a vhodně zvolte datové typy atributů

Cvičení 7.6: Kompozice vs dědičnost

- Navrhněte *příklad*, kdy byste různé objekty téhož konceptu řešili pomocí *kompozice*
  - Vytvořte třídy s vhodnými atributy a využijte kompozici k propojení tříd

- Navrhněte *další příklad*, kdy byste naopak použili *dědičnost*
  - Vytvořte nadřazenou třídu (předka) a odvozené třídy pro různé varianty (potomky)
  - Použijte dědičnost ke sdílení společných vlastností a metod

- V obou případech *své rozhodnutí zdůvodněte*

Jak psát kód: Polymorfismus

- Využijte polymorfismus k jednotnému zpracování různých typů a zvýšení flexibility kódu

```python
# Bez použití polymorfismu
def calculate_area(shape):
  if isinstance(shape, Circle):
    return pi * shape.radius ** 2
  elif isinstance(shape, Rectangle):
    return shape.width * shape.height
  elif ...
```

```python
# S polymorfismem
class Shape:
  def calculate_area(self):
    pass

class Circle(Shape):
  def calculate_area(self):
    return pi * self.__radius ** 2

class Rectangle(Shape):
  def calculate_area(self):
    return self.__width * self.__height
```

Jak psát kód: Upřednostněte kompozici před dědičností

- Používejte *kompozici k vytváření složitých objektů* místo toho, abyste se spoléhali na dědičnost tříd
  - Tj. nevytvářejte příliš mnoho typů, které se liší maličkostmi

```python
# Dědičnost
class Animal:
  def make_sound(self):
    pass

class Dog(Animal):
  def make_sound(self):
    return "Woof"
```

```python
# Kompozice
class Animal:
  def __init__(self, sound):
    self.__sound = sound

  def make_sound(self):
    return self.__sound

dog = Animal("Woof")
```

Reference

Dědičnost (angl. inheritance)

- https://www.w3schools.com/python/python_inheritance.asp
- https://naucse.python.cz/course/pyladies/beginners/inheritance/

Polymorfismus

- https://www.w3schools.com/python/python_polymorphism.asp