Programování 1 (NPRG030)

Cvičení 6: Třídy a objekty

Shrnutí přednášky: Třídy a objekty

```python
class Osoba:
  def __init__(self, jmeno: str, vek: int):
    """ Konstruktor třídy s parametry jmeno, vek """
    self.__jmeno = jmeno     # privátní atribut __jmeno
    self.__vek = vek         # privátní atribut __vek

  def get_jmeno(self) -> str:
    """ Tzv. getter, který umožňuje přístup k privátnímu atributu """
    return self.__jmeno

  @classmethod
  def porovnej_vek(cls, osoba1: 'Osoba', osoba2: 'Osoba') -> str:
    """ třídní metoda porovnávající věk dvou osob """
    if osoba1.__vek > osoba2.__vek:
      return f"{osoba1.__jmeno} je starší než {osoba2.__jmeno}"
    elif osoba1.__vek < osoba2.__vek:
      return f"{osoba2.__jmeno} je starší než {osoba1.__jmeno}"
    else:
      return f"{osoba1.__jmeno} a {osoba2.__jmeno} mají stejný věk"

  def __str__(self) -> str:
    """ Vrací textovou reprezentaci objektu třídy Osoba """
    return f"{self.__jmeno}, {self.__vek} let"
```

přímý přístup k privátnímu atributu způsobí chybu AttributeError

volání třídní metody přes Osoba.porovnej_vek(osoba1, osoba2)

Příklad 6.3: Geometrické tvary

- Vytvořte *třídu* v jazyce Python, která reprezentuje *Obdélník*, a druhou třídu, která reprezentuje *Kruh*
  - Třída `Obdelnik` bude mít následující atributy:
    - `strana_a` (datový typ float)
    - `strana_b` (datový typ float)
  - Třída `Kruh` bude mít následující atributy:
    - `polomer` (datový typ float)
  - Obě třídy budou umožňovat *výpočet obvodu* a *obsahu*

Cvičení 6.4: Spojový seznam

- Vytvořte jednosměrný *spojový seznam* v jazyce Python
  - Vnitřní uzel reprezentuje jako třídu `Uzel`, která má dva atributy
    - `data` (datový typ int), který nese data
    - `dalsi` (datový typ Uzel), tj. ukazatel na další uzel
  - Spojový seznam reprezentujte jako třídu `SpojovySeznam` a implementujte následující metody:
    - `je_prazdny -> bool`, která zjistí, zda je spojový seznam prázdný
    - `pridat_na_zacatek(data: int)`, která přidá na začátek seznamu uzel s danými daty
    - `pridat_na_konec(data: int)`, která přidá na konec seznamu uzel s danou hodnotou
    - `najit(data: int) -> bool`, která zjistí, zda je zadaná hodnota uložena v seznamu
    - `smazat(data: int)`, která odstraní první výskyt uzlu s danou hodnotou ze seznamu
    - `__str__`, která na standardní výstup vytiskne obsah spojového seznamu

Cvičení 6.5: Iterátor

- Vytvořte *iterátor pro* jednosměrný *spojový seznam* z příkladu 6.4
  - Iterátor je objekt, který umožňuje postupně procházet prvky seznamu

Cvičení 6.6: Generátor

- Vytvořte *generátor* pro jednosměrný *spojový seznam* z příkladu 6.4
  - Generátory jsou funkce, které umožňují postupně generovat hodnoty
  - Můžeme vytvořit generátor pro vrácení všech prvků spojového seznamu

Cvičení 6.7: Iterable

- Vytvořte *iterovatelný objekt ze spojového seznamu* z příkladu 6.5 nebo 6.6
  - Když máme implementovaný iterátor nebo generátor, můžeme třídu implementující spojový seznam označit jako iterable tím, že přidáme metodu `__iter__`, která vrátí iterátor nebo generátor
  - Tímto způsobem můžeme použít spojový seznam jako iterable v různých kontextech, například v cyklech `for`
    - Vyzkoušejte iteraci spojovým seznamem pomocí Vámi implementovaného iterable

Jak psát kód: Modularita a zapouzdření

- Rozdělte kód na *modulární komponenty*, které jsou *volně provázané* a poskytují dobře definovaná *rozhraní*
  - Známe *funkce* a *třídy*, později se naučíme pracovat s *moduly*

```
1 # Bez zapouzdření
2 class BankAccount:
3   def __init__(self, balance):
4     self.balance = balance
```

```
1 # Zapouzdření
2 class BankAccount:
3   def __init__(self, initial_balance):
4     self.__balance = initial_balance
5
6   def get_balance(self):
7     return self.__balance
```

Jak psát kód: Výčtové hodnoty

- Pro reprezentaci množiny souvisejících konstantních hodnot použijte výčty (enum)

```
1 # Použití výčtu
2 from enum import Enum
3
4 class Color(Enum):
5   RED = 1
6   GREEN = 2
7   BLUE = 3
```

Shrnutí: Jak psát kód: Zásady SOLID

- *Single Responsibility*
  - Každý objekt nebo třída by měla mít pouze jednu důležitou úlohu, aby byl kód snadno udržovatelný
- *Open-Closed*
  - Třídy by měly být otevřeny pro rozšíření novými funkcemi, ale zavřeny pro změny v existujícím kódu
- *Liskov Substitution*
  - Když třída dědí od jiné třídy, měla by být schopna být nahrazena instancí svého nadřazeného typu bez změn ve funkčnosti programu
- *Interface Segregation*
  - Místo jednoho obrovského rozhraní by měla být vytvářena menší, která jsou specifická pro konkrétní použití, aby se zabránilo nutnosti implementovat zbytečné metody
- *Dependency Inversion*
  - Programování by mělo záviset na abstrakcích a ne na konkrétních implementacích, což umožňuje flexibilitu a snadnou změnu závislostí

```python
# Uplatnění principů SOLID ve třídě
class Shape(ABC):
  @abstractmethod
  def area(self):
    pass

class Circle(Shape):
  def area(self):
    return __PI * self.__radius ** 2
```

Jak psát kód: Volné provázání (angl. loose coupling)

- Navrhujte moduly a třídy s *nízkou vzájemnou závislostí*, abyste zlepšili udržovatelnost a flexibilitu

```
1 # Tight coupling
2 class A:
3   def do_something(self, b_instance):
4     b_instance.perform_action()
```

```
1 # Loose coupling
2
3 class A:
4   def do_something(self, action_performer):
5     action_performer.perform_action()
```

Jak psát kód: Zbytečně nepřetěžujte operátory

- Přetěžujte operátory *pouze tehdy*, *pokud to zvyšuje přehlednost kódu*, nikoliv jen kvůli možnosti přetěžovat operátory

```
1 # Zbytečné přetěžování operátoru
2 class Vector:
3   def __add__(self, other):
4     pass
```

```
1 # Upřednostněte explicitní metody
2 class Vector:
3   def add(self, other):
4     pass
```

Jak psát kód: Déméteřin zákon

- Dodržujte Déméteřin zákon (*princip minimální znalosti*)
  - Minimalizujte přímé interakce mezi objekty a komunikujte pouze s nejbližšími přáteli

```
# Porušení Déméteřina zákonu
customer = order.customer
customer_name = customer.name
```

```
# Následování Déméteřina zákonu
customer_name = order.get_customer_name()
```

Jak psát kód: Dekorátory vlastností

- Využijte *dekorátory vlastností* k vytvoření metod *getter* a *setter*

```
 1 class Circle:
 2   def __init__(self, radius):
 3     self.__radius = radius
 4
 5   @property
 6   def radius(self):
 7     return self.__radius
 8
 9   @radius.setter
10   def radius(self, value):
11     if value < 0:
12       raise ValueError("Radius cannot be negative")
13     self.__radius = value
```

Reference

Třídy a objekty

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

Iterátory, generátory, iterable

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