Opjektum Orientált Programozás

LÁTHATÓSÁGI MÓDOSÍTÓK
(encapsulation)

Készítette: Vastag Atila

2020

A láthatósági módosítok az objektum-orientált programozás (OOP) egyik alapvető fogalma. Leírja az adatok és az adatokon változtatásokat előidéző függvények elrejtésének gondolatát osztályon belül. Ezzel korlátozni lehet a változók és függvények közvetlen elérését, és megakadályozhatja az adatok véletlenszerű módosítását. A véletlen változások elkerülése érdekében az objektum változóit csak az objektum függvényeivel lehet megváltoztatni. Az ilyen típusú változókat és függvényeket **private vátozó/függvény**-nek nevezzük.

```python
class Employee:
    def __init__(self, name: string, salary:float):
        self.name: string = name
        self._salary: float = salary # protected attribute


class Teacher(Employee):
    def __init__(self, name: string, sal: float, hours: float):
        Employee.__init__(self, name, sal)
        self.__hours: float = hours # private attribute


kirian: Teacher = Teacher("Kiran",10000, 40)

print(kirian.name)
print(kirian._sal)
print(kirian.__hours) #     HIBA: AttributeError:
                      #    ‘Teacher' object has no attribute '__hours'
```

A **python** *három* módosítót ismer, melyek alkalmazhatóak adattagokra és függvényekre is:

- **public:** az osztályon kívül és belül teljes mértékben hozzáférhető, ez az alapértelmezett érték is egyben
- **private:**
    - csakis a tartalmazó osztályon belül látható,
    - a változó értékét csak az osztályon belül lehet módosítani vagy kiolvasni,
    - a leszármazott osztályokban nem láthatóak
- **protected:**
    - csakis a tartalmazó és a leszármazott osztályon **(öröklődött objektum)** belül látható
    - a változó értékét csak az osztályon és az öröklő osztályon belül lehet módosítani vagy kiolvasni
    - a leszármazott osztályokban láthatóak

get() és **set()** metódusok alkalmazása az adattagok manipulánsára:

```
# person.py
class Person:
    def __init__(self, name: str, age: int = 0):
        self.name: str = name
        self.__age: int = age

    def display(self):
        print(self.name)
        print(self.__age)

    # visszaadja az __age adattag értékét az osztály példányán
    def getAge(self) -> int:
        return self.__age

    # beállítja az __age adattag értékét az osztály példányán
    def  setAge(self, age: int) -> None:
        self.__age = age
```

get() és **set()** metódusok alkalmazása az adattagok manipulánsára:

```
# main.py
person = Person('John', 40)

# adattagok kiírása a display() függvény segítségével
person.display()

# az age adattag megváltoztatása set() függvénnyel
person.setAge(45)
# az age adattag kiolvasasa get() függvénnyel
print(f"{person.name}'s age : {person.getAge()}")

# output
John
40
John's age : 45
```

get() és **set()** metódusok alkalmazása **Python** módra az adattagok manipulánsára:

```
#celsius.py

class Celsius:
    def __init__(self, temperature: float = 0):
        self.temperature = temperature

    def toFahrenheit(self) -> float:
        return (self.temperature * 1.8) + 32

    @property
    def temperature(self) -> float:
        print("Getting value...")
        return self._temperature

    @temperature.setter
    def temperature(self, value: float) -> None:
        print("Setting value...")
        if value < -273.15:
            raise ValueError("Temperature below -273 is not possible")
        self._temperature = value
```

get() és **set()** metódusok alkalmazása **Python** módra az adattagok manipulánsára:

```
from celsius import *

#main.py

human = Celsius(37)
print(human.temperature)
print(human.toFahrenheit())

human.temperature = 38
print(human.temperature)
print(human.toFahrenheit())
```

```
#main.py

Setting value...
Getting value...
37
Getting value...
98.60000000000001
Setting value...
Getting value...
38
Getting value...
100.4
```

Vegyük észre, hogy a **temperature** adattag értékének olvasása és írása nem az adattagon történik direkt módon, hanem indirekt módon a **get()** és **set()** függvényeken keresztül.