Python
Les classes abstraites et les protocoles

Nicolas Delestre

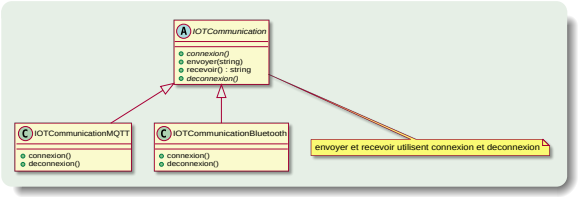
INSA
ROUEN NORMANDIE

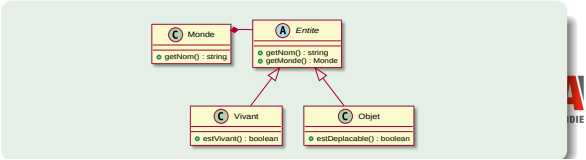
Classe abstraite (CA), pourquoi ?

- Créer une classe dont certaines méthodes sont déclarées mais non définies, mais qui peuvent être utilisées dans des méthodes déclarées et définies
- Créer une classe qui est conceptuelle, qui factorise des attributs ou des méthodes, évitant de les copier/coller dans différentes classes

Conséquences

- Classe qui ne peut pas avoir d'instance (d'où le terme abstrait)
- Classe qui peut avoir des méthodes abstraites : méthodes déclarées mais non définies
- Couplage fort : une classe C est de type CA si elle est sous-classe de CA





Interface (I)

- Pourquoi ?
  - Définir un type avec uniquement des déclarations de méthodes
- Conséquence
  - Il n'est pas possible de définir des méthodes, juste les déclarer
  - Couplage faible : une classe C est de type I si elle implémente I en définissant les méthodes déclarées par I

INSA ROUEN NORMANDIE

Classe abstraite en Python

Comment ?

- Déclarer une classe abstraite :
  - hériter de la classe `ABC` (pour *Abstract Base Class*) proposé par le module `abc`
  - posséder au moins une méthode abstraite (utilisation du décorateur `abstractmethod`). Ce décorateur peut être associé aux décorateurs **classmethod**, **staticmethod** et **property**
- S'assurer que les sous-classes implémentent ces méthodes

Exemple

```python
from abc import ABC, abstractmethod

class Forme(ABC):
    @abstractmethod
    def aire(self) -> float:
        pass

    @abstractmethod
    def perimetre(self) -> float:
        pass
```

```python
class Cercle(Forme):
    def __init__(self, rayon: float):
        self.rayon = rayon

    def aire(self) -> float:
        return 3.14 * self.rayon**2

    def perimetre(self) -> float:
        return 2 * 3.14 * self.rayon
```

Les (`Protocol`)

Constat

- Grâce au *Duck Typing*, le concept d'Interface n'est pas utile en python
- Mais ce n'est qu'à l'exécution que l'on peut se rendre compte des problèmes $\Rightarrow$ besoin de formaliser des dépendances, des attentes

Les protocoles : les interfaces de Python

- Améliorer le typage statique en spécifiant des méthodes requises

Équivalent aux interfaces d'autres langages comme Java

Comment ?

- Déclarer un protocole : sous classe de la classe `Protocol` du module typing
- Être du même type qu'un protocole : juste posséder les méthodes définies par le protocole

Exemple

```python
from typing import Protocol

class PeutVoler(Protocol):
    def voler(self) -> None:
        pass

class Oiseau:
    def voler(self) -> None:
        print("Battre des ailes et voler!")

class Avion:
    def voler(self) -> None:
        print("Planer àtravers les nuages!")

def effectuer_vol(entite: PeutVoler) -> None:
    entite.voler()

oiseau_instance = Oiseau()
avion_instance = Avion()

effectuer_vol(oiseau_instance)
effectuer_vol(avion_instance)
```

INSA ROUEN NORMANDIE

Conclusion

- Les classes `ABC` et `Protocol` offrent des mécanismes puissants pour améliorer la structure et la fiabilité du code en renforçant les contrats
- Les classes abstraites garantissent l'implémentation de méthodes spécifiques dans les sous-classes, renforçant la hiérarchie des classes
- Les protocoles offrent une flexibilité accrue en spécifiant des contrats moins rigides, adaptés à des situations où une hiérarchie stricte n'est pas nécessaire

INSA ROUEN NORMANDIE