2N6 Programmation 2

avec python™

Méthodes statiques

Méthodes privées et abstraites

Classes abstraites

Sommaire des différentes méthodes

- **Méthode** :
  - **d'instance** : appartient à une instance d'une classe.
  - **de classe** : appartient à la classe elle-même et non à chaque instance.
  - **statique** : Méthode qui est contenue dans la classe, mais fonctionne indépendamment. Elle ne fait pas de référence et n'affectent pas la classe ou une instance.

  - **privée** : Méthodes qui ne sont pas appelées hors de la classe.
  - **public** : Méthodes qui peuvent être appelées hors de la classe

  - **abstraite** : Méthodes qui DOIVENT être redéfinies dans les sous-classes.

Résumer différentes méthodes (EXEMPLE)

```python
class Employe:
    liste_employe = []
    next_ID = 1000
    def __init__(self,nom,prenom):
        pass

    def retourner_nom_complet(self):
        return f"{self.prenom} {self.nom}"

    @classmethod
    def afficher_liste_employe(cls):
        print(json.dumps(cls.liste_employe, indent = 4))

    @staticmethod
    def info_retraite():
        return "Il faut avoir 65 et plus ou avoir 35 ans d'ancienneté pour se qualifier."
```

méthode d'instance

méthode de classe

méthode statique

Méthodes statiques

- Méthodes qui appartiennent à la classe, mais qui ne font pas référence à une instance ou bien à la classe elle-même.
- On utilise le décorateur "@staticmethod"

```
    @staticmethod
    def info_retraite():
        return "Il faut avoir 65 et plus ou avoir 35 ans d'ancienneté pour se qualifier."
```

- La méthode info_retraite() est une méthode statique. Elle appartient à la classe Employe mais n'utilise pas d'attributs ou méthodes de la classe ou de l'instance.

Méthodes de classe vs méthodes statique

```python
class Employe:
    taux = 1.09
    def __init__(self) -> None:
        pass

    def changer_taux_1(nvx_taux):
        Employe.taux = nvx_taux

    def changer_taux_2(Employe,nvx_taux):
        Employe.taux = nvx_taux

    @classmethod
    def changer_taux_3(cls,nvx_taux):
        cls.taux = nvx_taux

    @staticmethod
    def changer_taux_4(nvx_taux):
        Employe.taux = nvx_taux

```

- Toutes ces méthodes donnent le même résultat.

MAIS

- Une seule est conforme aux standards de programmation

- On DOIT respecter les standards pour que notre code soit lisible par d'autres programmeurs et par nous-mêmes

Méthodes privées

retour

- On indique une méthode privée avec un "__" (double "underscore")
- Méthodes qui ne sont pas utilisées hors de la classe.
  - Seule la classe peut appeler ces méthodes.
  - Appeler une méthode privée hors de la classe génère une erreur.

```
28      def __methode_prive():
29          print("Cette méthode est privée.")
30
31  Employe.__methode_prive()
32
```

PROBLÈMES SORTIE CONSOLE DE DÉBOGAGE TERMINAL .NET INTERACTIVE JUPYTER

```
    Employe.__methode_prive()
AttributeError: type object 'Employe' has no attribute '  methode_prive'
```

Méthodes privées

retour

- Tous les types de méthodes peuvent être privés.

```
    def __methode_instance_prive(self):
        pass
    @classmethod
    def __methode_classe_prive(cls):
        pass
    @staticmethod
    def __methode_static_prive():
        pass
```

Méthodes et classes abstraites

- Supposons que nous avons une classe employée avec des sous-classes Programmeur et Vendeur

  MAIS

- On ne veut pas avoir d'instances de la classe Employe.
  Parce que tous nos employés ont un rôle ou un poste.

- On transforme Employe en une classe abstraite. Une classe abstraite est une classe qui ne peut pas être instanciée, mais à partir de laquelle on peut créer des sous-classes.

Méthodes abstraites

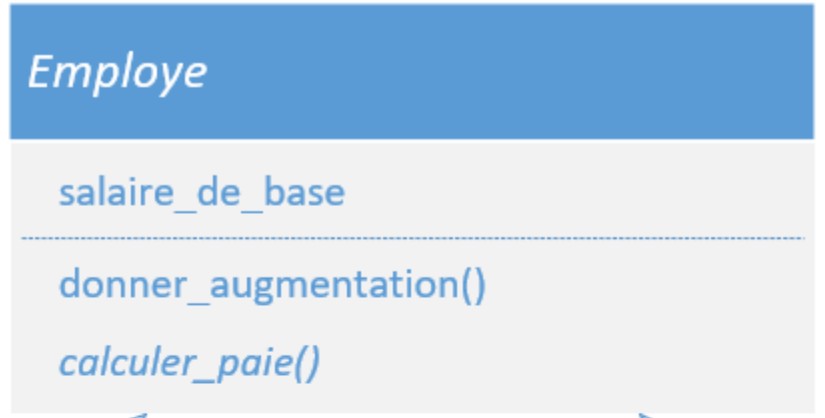
- Cette classe abstraite a des méthodes dont au moins une qui sera abstraite.

- Lorsqu'on fait une sous-classe à partir de cette classe, on devra redéfinir les méthodes qui étaient abstraites dans la classe parent.

Méthodes et classes abstraites (UML)

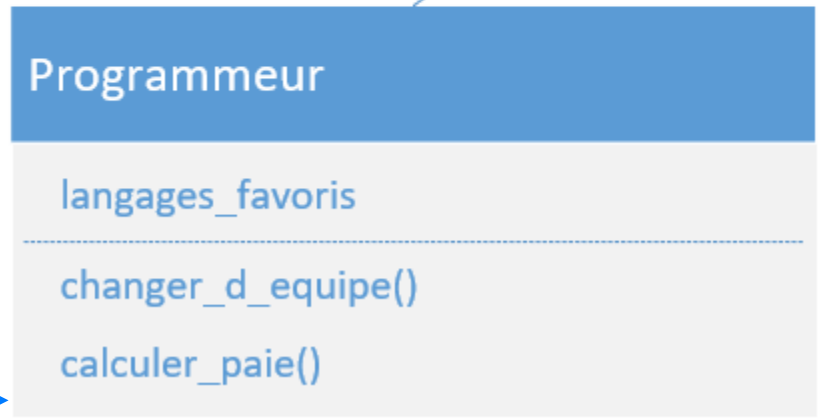
La classe *Employe* et la méthode *calculer_paie()* sont en italique.

Indique qu'elles sont abstraites



La méthode calculer_paie() réapparait. Cette fois elle n'est pas en italique.

Indique que la méthode a été redéfinie.



Création de méthodes et classes abstraites

- Nécessite l'utilisation du module abc (Abstract Base Classe)
- Permets de créer une classe abstraite simplement en dérivant cette classe de la classe ABC
- Permets de créer des méthodes abstraites à l'aide du décorateur @abstractmethod

```python
from abc import ABC, abstractmethod


class Employe(ABC):
    liste_employe = []
    next_ID = 1000
    def __init__(self,nom,prenom):
        pass

    @abstractmethod
    def test_abs(self):
        pass


class Programmeur(Employe):
    def __init__(self, nom, prenom,language_favori) :
        pass

    def test_abs(self):
        print("vla")

```

Utilisation de classes abstraites

- Employe hérite de ABC
  - ➔ permet d'utiliser le décorateur @abstractmethod

- La présence d'une méthode abstraite fait que la classe ne peut **pas** être instancié.

```python
from abc import ABC, abstractmethod


class Employe(ABC):
    liste_employe = []
    next_ID = 1000
    def __init__(self,nom,prenom):
        pass

    @abstractmethod
    def test_abs(self):
        pass


class Programmeur(Employe):
    def __init__(self, nom, prenom,language_favori) :
        pass

    def test_abs(self):
        print("vla")

```

```
23  employe1 = Employe("Anna","Tremblay")
24
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\pierre-paul.gallant\OneDrive - Cégep Édouard-Montpetit\Session 24H\2N6R-Pr
N6R-Prog2/R19/prep/employe.py"
Traceback (most recent call last):
  File "c:\Users\pierre-paul.gallant\OneDrive - Cégep Édouard-Montpetit\Session 24H\2N
    employe1 = Employe("Anna","Tremblay")
TypeError: Can't instantiate abstract class Employe with abstract method test_abs
```

Utilisation de classes abstraites

- Programmeur hérite de Employe
- La méthode test_abs **DOIT** être redéfinis
- On peut maintenant instancié la classe Programmeur

```python
from abc import ABC, abstractmethod


class Employe(ABC):
    liste_employe = []
    next_ID = 1000
    def __init__(self,nom,prenom):
        pass

    @abstractmethod
    def test_abs(self):
        pass


class Programmeur(Employe):
    def __init__(self, nom, prenom,language_favori) :
        pass

    def test_abs(self):
        print("vla")

```

```python
employe1 = Programmeur("Anna","Tremblay","python")
employe1.test_abs()
print( isinstance(employe1, Employe) )
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\pierre-paul.gallant\OneDrive - Cégep Édouard-Montpetit
N6R-Prog2/R19/prep/employe.py"
vla
True
```