Programmation Objet

2 – La programmation orientée objet

VCOD 2022/2023

Maxime Raynal

Plan du jour

- Le paradigme de la POO
- Les concepts fondamentaux de la POO
- Exemples sur machines

Le paradigme de la POO

- Qu'est-ce qu'un paradigme de programmation ?
  - Permet de décrire et classer les langages de programmation en fonction de plusieurs critères (fonctionnalités, exécution, données …)

- Pourquoi différents paradigmes ?
  - Il n'existe pas, en programmation, de langage magique qui puisse tout faire.
  - On a donc une diversité d'outils pour résoudre différents problèmes

Le paradigme de la POO

- Le paradigme de la POO:
  - Permet au développeur de créer des briques logicielles, appelées *objets*.
  - Un objet représente un concept, et possède une structure interne

- Histoire:
  - Simula (1966) regroupe données et procédures.
  - Simula I (1972) formalise les concepts d'objet et de classe.
  - Smalltalk (1972) : généralisation de la notion d'objet.
  - C++ (1983) : Extension du C s'inspirant de Simula.

Le paradigme de la POO

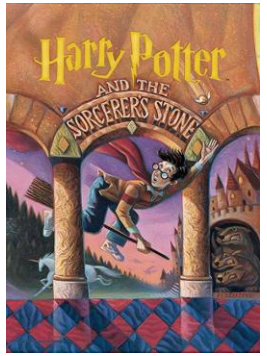
- Pourquoi la POO ?
  - Facilite la conception et la maintenance de gros logiciels.
  - Au fil du temps, les logiciels ont gagné en complexité.

Le petit prince
~ 2000 lignes

UNIX

Unix V1 (1977)
~4,500 lignes de code

Hubble space telescope (1990)
~50,000 lignes de code



La saga Harry Potter
~100,000 lignes

Linux™

Linux v1.0 (1994)
~175,000 lignes de code

Le paradigme de la POO

- Pourquoi la POO ?
  - Facilite la conception et la maintenance de gros logiciels.
  - Au fil du temps, les logiciels ont gagné en complexité.

Linux
v1.0 (1994):~175,000 lignes
v5.19 (2022):~ 27,000,000 lignes

Windows 10
~ 50,000,000 lignes

debian

Debian sid
~ 1,700,000,000 lignes

Google
~ 2,000,000,000 lignes

Le paradigme de la POO

- Pourquoi la POO ?
  - Facilite la conception et la maintenance de gros logiciels.
  - Au fil du temps, les logiciels ont gagné en complexité.

- La POO permet:
  - de faciliter la maintenance;
  - de faciliter le développement collaboratif;
  - de faciliter l'adaptation aux changements;
  - de faciliter la réutilisabilité des briques logicielles.

Le paradigme de la POO

- Avantages de la POO:
  - **Modularité**: Les objets sont faciles à manipuler.
  - **Abstraction**: Les objets représentent des concepts du monde réel.
  - **Réutilisabilité**: Les objets sont facilement réutilisables et extensibles.
  - **Encapsulation**: Les objets peuvent gérer le niveau de visibilité de leurs composantes.

Les concepts fondamentaux de la POO

- **Les concepts que nous allons voir ce semestre**
  - **Objet**
  - **Classe**
  - **Instance**
  - **Attribut**
  - **Méthode**

  } aujourd'hui

  - Héritage
  - Encapsulation
  - Polymorphisme
  - Composition
  - ….

Les concepts fondamentaux de la POO

- **Objet:**
  - Un objet représente un **concept**
  - Il possède trois composantes:
    - Son identité: généralement une ***référence*** à l'objet, ou son adresse mémoire.
    - Son état: une collection de variable appelées ***attributs*** contenant des informations sur l'objet.
    - Son comportement: une collection de fonctions appelées ***méthodes***.

Les concepts fondamentaux de la POO

- **Classe:**
  - Une **classe** est un modèle à partir duquel on fabrique des objets.
  - On peut voir une classe comme un moule, ou une maquette.
  - La classe est un modèle de la structure (attributs et méthodes) des objets associés à cette classe.

- **Instance:**
  - Une **instance d'une classe** est un objet qui "appartient" à cette classe.
  - Le mot vient de l'anglais *instance.*

Les concepts fondamentaux de la POO

Par exemple:

- Prenons le concept de *vélo*.
  - On peut parler de la classe *Vélo.*
  - Chaque vélo garé sur le campus est un objet, une instance de *Vélo.*
  - Les vélos possèdent des points communs, sont constitués des mêmes composants. On peut les garer, les utiliser, les réparer … ; ils sont composés d'un cadre, d'une paire de roues, ...

Les concepts fondamentaux de la POO

- **Exemple:** Créons une classe pour les points dans le plan.

```
1 class Point:
2     def __init__(self, x=0, y=0):
3         self.x = x
4         self.y = y
5
6     def move(self, dx, dy):
7         self.x += dx
8         self.y += dy
```

Le mot-clé *class* définit une nouvelle classe appelée *Point.*

- Pour différencier les noms de classe, on les note en CamelCase.
- Le niveau d'indentation sous *class* représente la portée (*the scope*) de la définition de la classe.

Les concepts fondamentaux de la POO

- **Exemple:** Créons une classe pour les points dans le plan.

```python
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x += dx
        self.y += dy
```

Chaque classe possède obligatoirement un constructeur: *__init__*

- Ce nom (commence et termine par __)est un nom réservé en python.
- Le constructeur est une méthode qui sert à initialiser les attributs de l'objet.
- Le premier argument de toutes les méthodes d'une classe est *self*, qui est une référence à l'objet lui-même (*this* dans d'autres langages).

Les concepts fondamentaux de la POO

- **Exemple:** Créons une classe pour les points dans le plan.

```python
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x += dx
        self.y += dy


p1 = Point()
p2 = Point(1, 4)
p3 = Point(x=2, y=6)
```

P1, p2 et *p3* sont objets, ce sont des instances de la classe Point.

Les concepts fondamentaux de la POO

- **Exemple:** Créons une classe pour les points dans le plan.

```python
class Point:
    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def move(self, dx, dy):
        self.x += dx
        self.y += dy


p1 = Point()
p2 = Point(1, 4)
p3 = Point(x=2, y=6)

print(type(p1))  # <class 'Point'>
print(p2.x, p2.y)  # 1, 4
p2.move(dx=2, dy=4)
print(p2.x, p2.y)  # 3, 8
print(isinstance(p1, Point))  # True
```

On peut accéder aux attributs et aux méthodes d'un objet via l'opérateur '.'

On peut utiliser les fonctions *type* pour renvoyer le type d'un objet, et *isinstance* pour vérifier qu'un objet a le type (càd la classe) attendu.

Les concepts fondamentaux de la POO

Prenons de suite de bonnes habitudes

- On code en anglais
- Respect des conventions d'écriture (snake_case pour les variables et fonctions, CamelCase pour les classes)
- On donne les types des arguments et du retour d'une fonction ou d'une méthode
- On met des commentaires dans son code

```python
class Point:
    """This class represents a 2D point in the plan"""
    def __init__(self, x: float = 0, y: float = 0):
        self.x = x
        self.y = y

    def move(self, dx: float = 0, dy: float = 0) -> None:
        """Shifts the point of dx and of dy"""
        self.x += dx
        self.y += dy
```

Les concepts fondamentaux de la POO

Les attributs d'un objet: il existe deux types d'attributs

- **Les attributs d'instance:**
  - Sont définis dans le constructeur (bonnes pratiques) ou dans les autres méthodes (mauvaises pratiques).
  - Chaque instance possède ses propres attributs d'instance.
  - On y accède via le nom de l'objet.
- **Les attributs de classe:**
  - Sont définis dans le scope de la classe
  - Les attributs de classe sont partagés par toutes les instances de la classe.
  - **On y accède via la nom de la classe.**

Les concepts fondamentaux de la POO

Les attributs d'un objet: il existe deux types d'attributs

```python
class Point2D:
    """This class represents a 2D point in the plan"""
    DIMENSIONS = 2

    def __init__(self, x: float = 0, y: float = 0):
        self.x = x
        self.y = y

    def move(self, dx: float = 0, dy: float = 0) -> None:
        """Shifts the point of dx and of dy"""
        self.x += dx
        self.y += dy


p1 = Point2D(3, 7)
p2 = Point2D(1, 4)

print(p1.x, p1.y)  # 3 7
print(p2.x, p2.y)  # 1 4
print(Point2D.DIMENSIONS)  # 2
```

- *x* et *y* sont des attributs d'instance.
- *DIMENSIONS* est un attribut de classe.