2N6 Programmation 2

# avec python™

Pseudo-code,

héritage de méthodes,
et méthodes de classes

# Pseudocode

# Pseudocode

- L'UML permet de mettre les objets en relation avant de commencer un projet

- Avant de coder, on veut comprendre la logique de ce qu'on va faire.
  - Plus facile de le faire sans se préoccuper de la syntaxe du code
  - D'où l'utilité du pseudo code

# Pseudocode (Kecéça?)

- Le pseudocode est une façon d'écrire du code sans se préoccuper des particularités d'un langage spécifique.
- On écrit ce qui sera fait à chacune des étapes du code dans un langage naturel.
- Permet de planifier facilement ce qu'on veut faire, de le communiquer avec d'autres programmeurs et même avec des non programmeurs.

# Exemple simple de pseudo-code

- On veut calculer la somme de deux nombres:

```
Début
  # Demander à l'utilisateur d'entrer les deux nombres
  Demander "Entrez le premier nombre : " et le stocker dans la variable a
  Demander "Entrez le deuxième nombre : " et le stocker dans la variable b

  # Calculer la somme des deux nombres
  somme = a + b

  # Afficher le résultat
  Afficher "La somme de ", a, " et ", b, " est égale à ", somme

Fin
```

# Exemple simple de pseudo-code

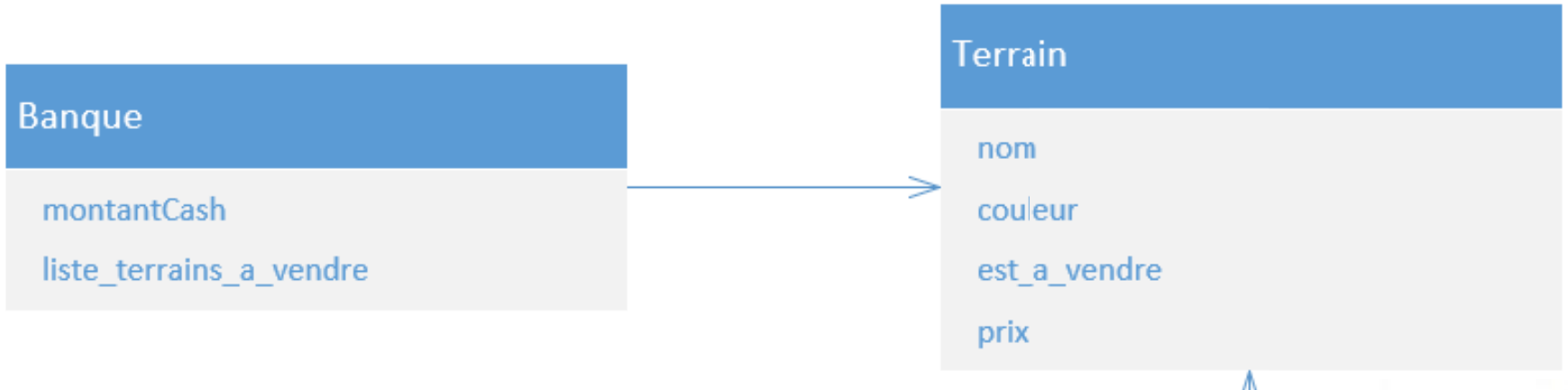
- Dans cet exemple, le pseudocode utilise des instructions simples comme "Demander", "Afficher" et des opérateurs mathématiques comme "+" pour représenter les étapes nécessaires pour calculer la somme de deux nombres.

- Ce pseudocode est très simple, mais il illustre comment le pseudocode peut être utilisé pour décrire un algorithme de manière plus claire et compréhensible que le code réel.

# Exemple de Monopoly

- Supposons qu'on développe un jeu de Monopoly
- On veut développer une fonction qui va permettre au joueur d'acheter des terrains de la banque.
- Avant de s'y lancer. On conceptualise ce que nous allons faire.

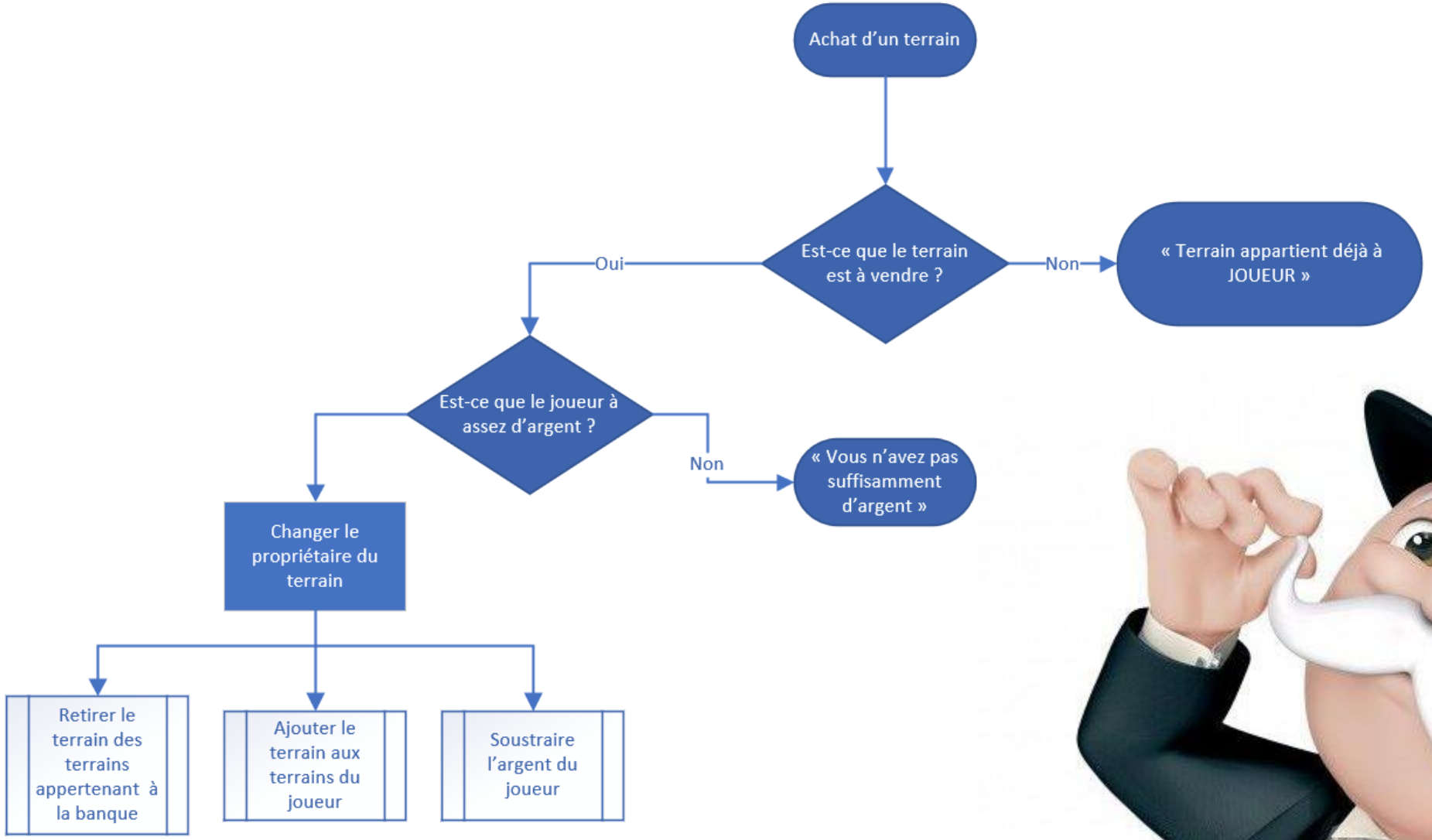
# Exemple de Monopoly - UML




# Exemple de Monopoly

- On veut modéliser la méthode acheter() du Joueur.
- Il faudra en premier vérifier que le terrain que le joueur veut acheter est bel et bien à vendre
  - Si oui :
    - Il faudra aussi vérifier que le joueur a aussi assez de cash pour acheter le terrain
    - Si oui :
      - On diminuera le montant cash du joueur du coût du terrain
      - On ajoutera le terrain en question dans la collection de terrains de ce joueur
      - On enlèvera le terrain de la collection des terrains à vendre de la Banque
      - On indiquera que le terrain n'est plus à vendre.
    - Sinon :
      - On pourra lancer un message comme quoi le joueur n'a pas assez d'argent pour acheter le terrain
  - Sinon :
    - On pourra lancer un message comme quoi le terrain en question n'est pas à vendre.

# Exemple de Monopoly – Diagramme de flux




# Retour sur le TP1

- On va refaire le TP1 mais en utilisant le pseudocode.

# Méthodes de classe

# Héritage des méthodes

```python
class Voiture:
    def __init__(self,marque) -> None:
        self.marque = marque

    def klaxon(self):
        print("honk!")
```

```python
class Pickup(Voiture_moteur):
    def __init__(self, marque, reservoir,puissance):
        super().__init__(marque, reservoir)
        self.puissance = puissance

remorque = Pickup("Ford","60L","1200hp")
remorque.klaxon()
```

```
PROBLÈMES 1   TERMINAL   ···   Python

honk!
```

- La classe Pickup hérite de la méthode klaxon et les objets de la classe Pickup peuvent donc utiliser cette méthode

# Surcharge de méthodes

```python
class Pickup(Voiture_moteur):
    def __init__(self, marque, reservoir,puissance):
        super().__init__(marque, reservoir)
        self.puissance = puissance

    def klaxon(self):
        print("HOOONKKK!")

remorque = Pickup("Ford","60L","1200hp")
remorque.klaxon()
```

```
PROBLÈMES 1  SORTIE  TERMINAL  ···  Python

HOOONKKK!
```

```python
class Voiture_de_luxe(Voiture_moteur):
    def __init__(self, marque, reservoir,prix):
        super().__init__(marque, reservoir)
        self.prix = prix

fancy_car = Voiture_de_luxe("Mercedes","60L","120000")
fancy_car.klaxon()
```

```
PROBLÈMES 1  SORTIE  TERMINAL  ···  Python

honk!
```

# Méthodes de classes

```python
class Employe:
    nb_employes = 0
    base_augmentation = 1.04

    def __init__(self, prenom, nom, salaire):
        self.prenom = prenom
        self.nom = nom
        self.salaire = salaire
        self.courriel = prenom + '.' + nom +'@gmail.com'
        Employe.nb_employes +=1

    def nom_complet(self):
        return '{} {}'.format(self.prenom, self.nom)

    def donner_augmentation(self):
        self.salaire = int(self.salaire * self.base_augmentation)
        # nous utilisons self car l'augmentation de base pourrait varier selon l'employé instancié

    @classmethod
    def from_string(cls, emp_str):
        "Constructeur pour créer un employé à partir d'une chaine séparée avec un '-'"
        prenom, nom, salaire = emp_str.split('-')
        return cls(prenom, nom, salaire)
```

Decorator

Méthode de classe

Fait référence à la classe

# Méthodes de classes

```python
@classmethod
def from_string(cls, emp_str):
    "Constructeur pour créer un employé à partir d'une chaine séparée avec un '-'"
    prenom, nom, salaire = emp_str.split('-')
    return cls(prenom, nom, salaire)
```

- On va rarement instancier tous nos employés à la main. Cette méthode permettrait d'instancier des employés à partir d'une seule ligne de texte qui nous proviendrait de la lecture d'un fichier, csv ou autre.

```python
emp_1 = Employe('Marc','Tremblay',50000)

emp_2 = Employe.from_string("Joanna-Tremblay-52000")
```

# Méthodes de classes

- On fait des méthodes de classe quand la méthode ne fait pas référence aux objets instanciés
- Cette méthode sera la même pour tous les objets instanciés.
- On doit utilisé le decorator **@classmethod** pour identifier que c'est une méthode de classe et pour pouvoir appeler la classe en utilisant **cls**

# Priorité des noms

```python
class Employe:
    id_1 = 1
    id_2 = 1
    id_3 = 1

    def __init__(self,nom,prenom) -> None:
        self.nom = nom
        self.prenom = prenom
        self.id_1 = 2
        self.id_2 = 2

    def retourne_id(self):
        id_1 = 3
        print(id_1)

exemple = Employe("a","b")

exemple.retourne_id()
print(exemple.id_2)
print(exemple.id_3)
```