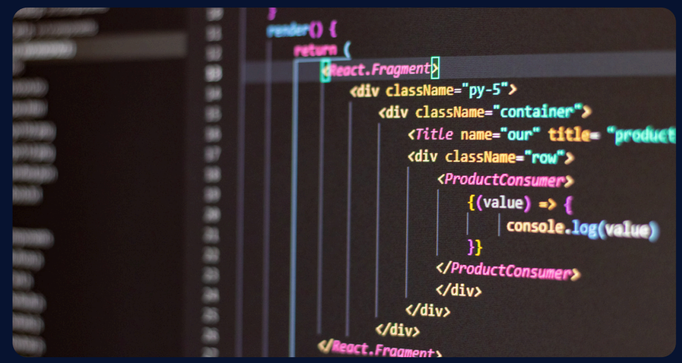


L'HÉRITAGE

Laffet takwa

Qu’est-ce que l’héritage ?

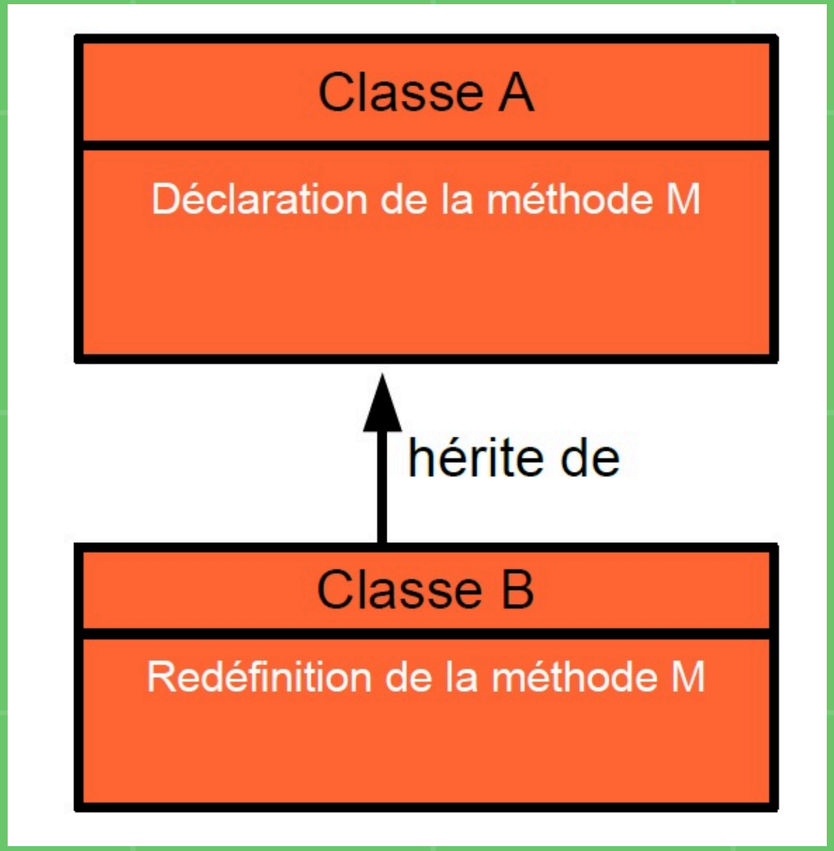
L’héritage est un mécanisme qui permet de créer une nouvelle classe (appelée classe fille) à partir d’une classe existante (appelée classe mère).
➡ La classe fille hérite des attributs et des méthodes de la classe mère, et peut :

- Les utiliser directement,
- Les modifier (surcharge),
- Ou en ajouter de nouvelles.

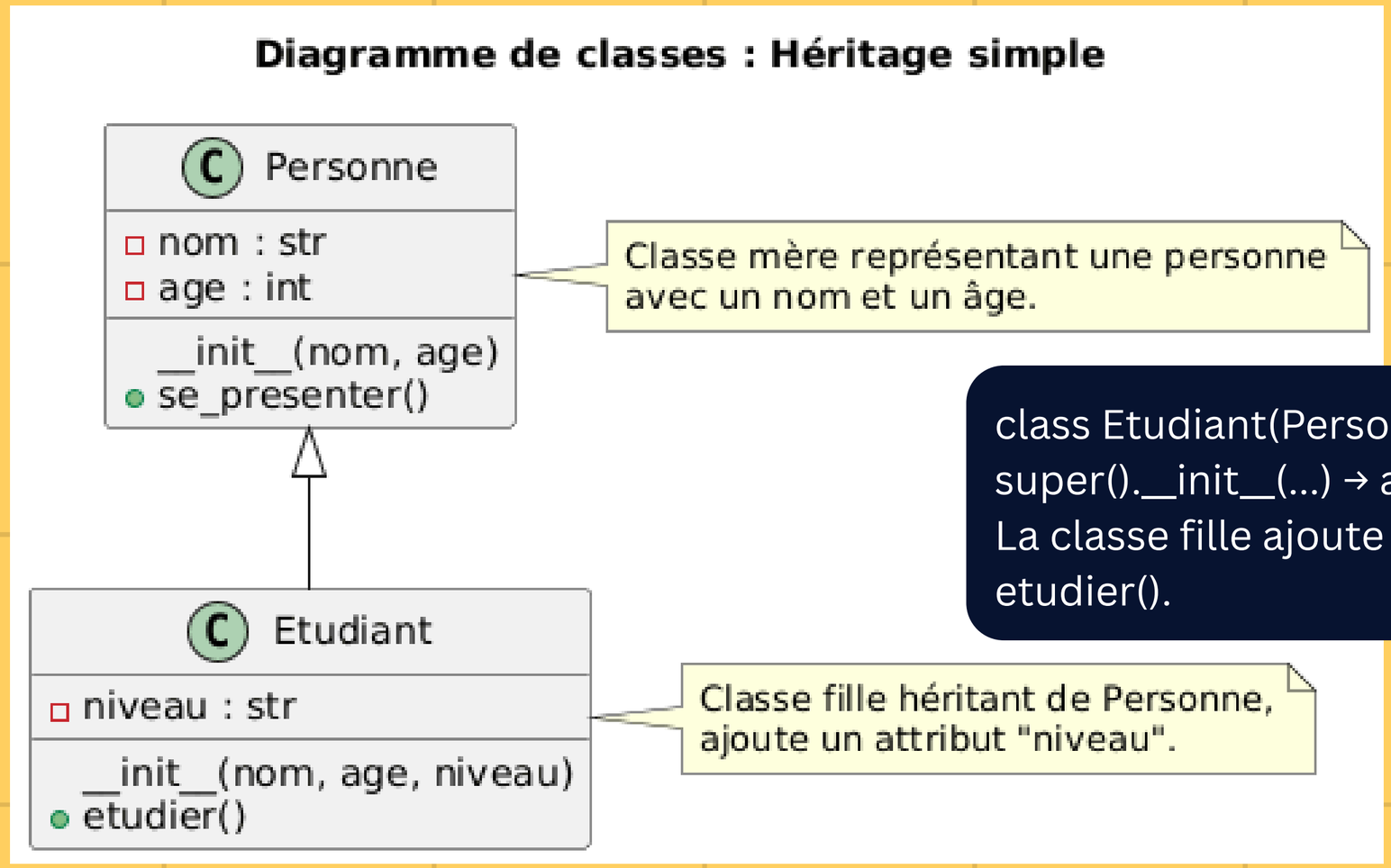
Qu’est-ce que l’héritage ?

L’héritage est un mécanisme qui permet de créer une nouvelle classe (appelée classe fille) à partir d’une classe existante (appelée classe mère).
➡ La classe fille hérite des attributs et des méthodes de la classe mère, et peut :

- Les utiliser directement,
- Les modifier (surcharge),
- Ou en ajouter de nouvelles.



Héritage simple

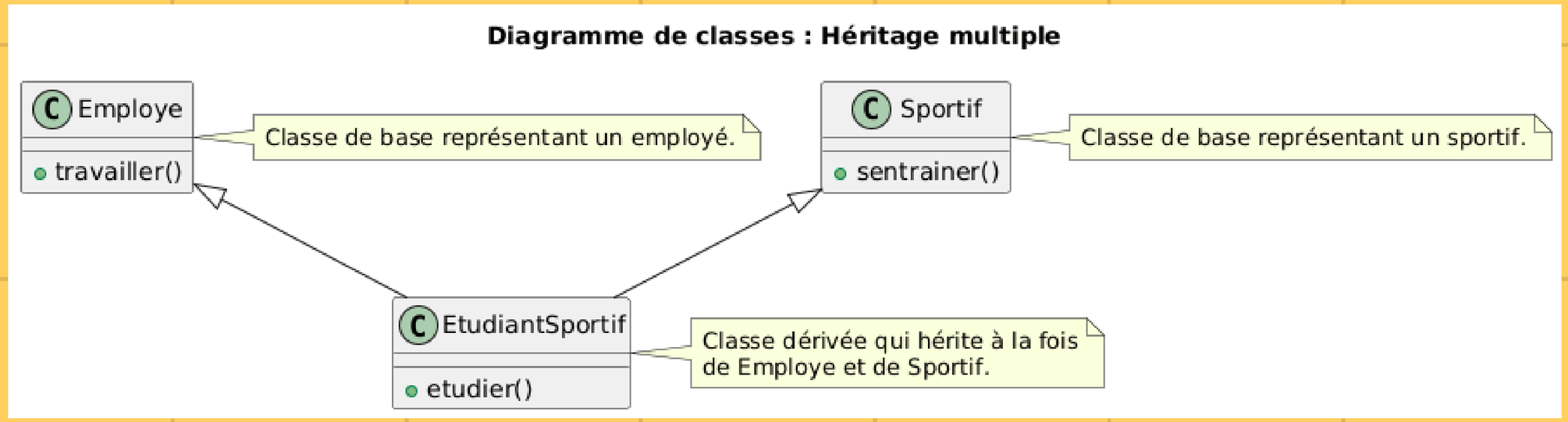


class Etudiant(Personne) → Etudiant hérite de Personne.
super().__init__(...) → appelle le constructeur de la classe mère.
La classe fille ajoute son propre attribut niveau et sa méthode etudier().

Héritage simple

```python
class Personne:
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def __init__(self, nom, age):
        self.nom = nom
        self.age = age

    Windsurf: Refactor | Explain | Generate Docstring | ×
    def se_presenter(self):
        print(f"Je m'appelle {self.nom} et j'ai {self.age} ans.")

# Classe fille
Windsurf: Refactor | Explain
class Etudiant(Personne):
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def __init__(self, nom, age, niveau):
        super().__init__(nom, age)  # Appel du constructeur de la classe mère
        self.niveau = niveau

    Windsurf: Refactor | Explain | Generate Docstring | ×
    def etudier(self):
        print(f"{self.nom} étudie en {self.niveau}.")

# Utilisation
e1 = Etudiant("Sara", 19, "2ème année")
e1.se_presenter()
e1.etudier()
```

Héritage multiple



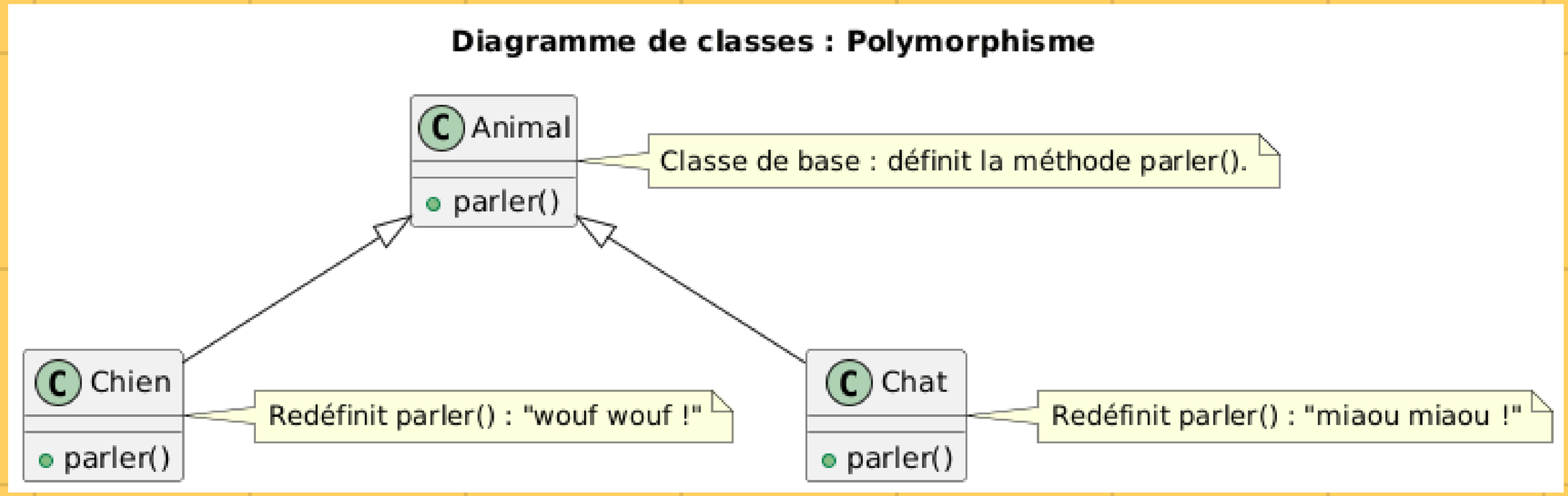
Héritage multiple

Une classe peut hériter de plusieurs classes à la fois.

```python
class Employe:
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def travailler(self):
        print("Je travaille dans une entreprise.")

Windsurf: Refactor | Explain
class Sportif:
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def sentrainer(self):
        print("Je m'entraîne tous les jours.")

Windsurf: Refactor | Explain
class EtudiantSportif(Employe, Sportif):
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def etudier(self):
        print("J'étudie aussi mes cours à l'université.")

# Utilisation
p = EtudiantSportif()
p.travailler()
p.sentrainer()
p.etudier()
```

Le polymorphisme

Le polymorphisme signifie “plusieurs formes”.
En POO, cela veut dire que plusieurs classes différentes peuvent partager le même nom de méthode, mais avec un comportement différent.



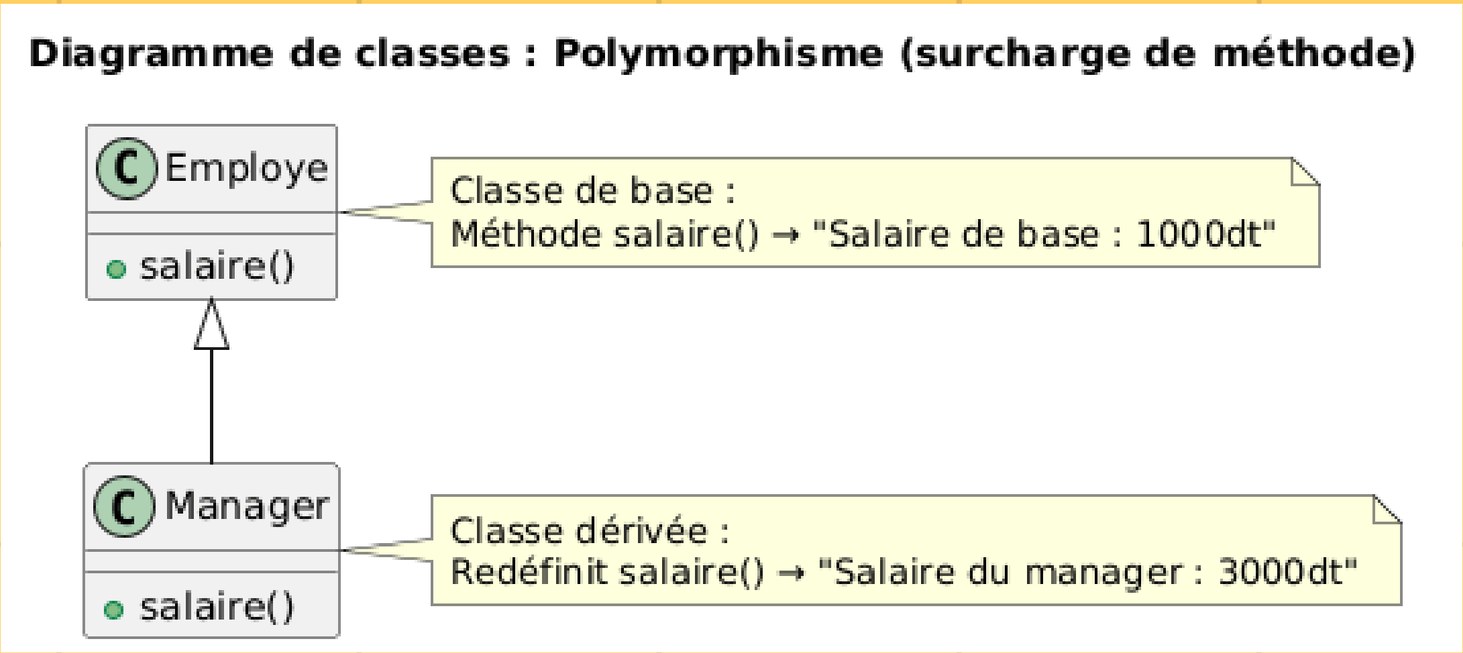
Le polymorphisme

Chaque objet réagit à sa manière à la même méthode parler().
==> C’est le polymorphisme.

```python
class Animal:
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def parler(self):
        print("Un animal fait un bruit.")

Windsurf: Refactor | Explain
class Chien(Animal):
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def parler(self):
        print("wouf wouf !")

Windsurf: Refactor | Explain
class Chat(Animal):
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def parler(self):
        print("miaou miaou !")

# Utilisation
animaux = [Chien(), Chat(), Animal()]

for a in animaux:
    a.parler()
```

Surcharge des méthodes

La surcharge signifie redéfinir une méthode héritée de la classe mère dans la classe fille.

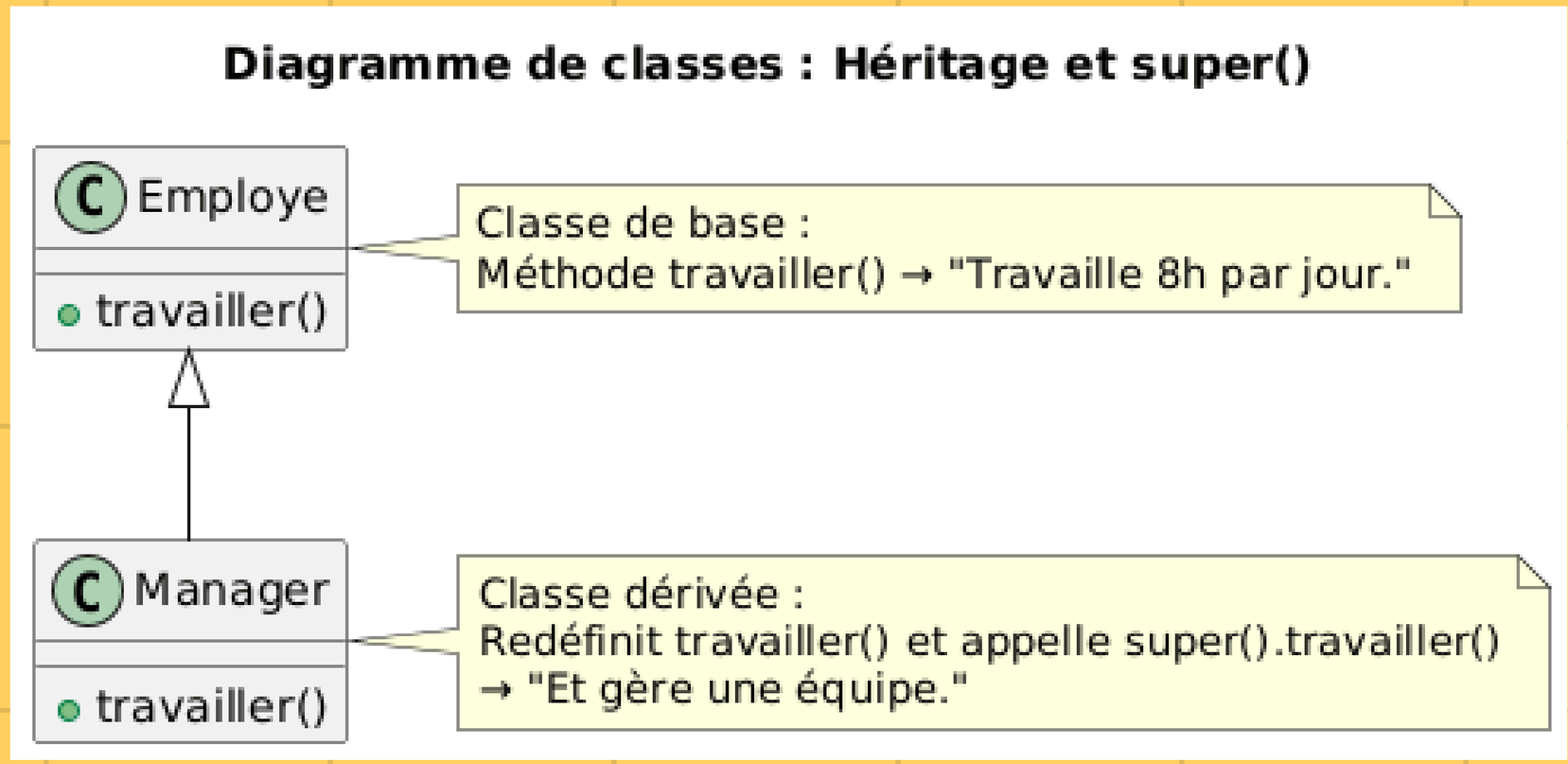


Surcharge des méthodes

Ici, Manager redéfinit (surcharge) la méthode salaire().

```python
class Employe:
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def salaire(self):
        print("Salaire de base : 1000dt")

Windsurf: Refactor | Explain
class Manager(Employe):
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def salaire(self):
        print("Salaire du manager : 3000dt")

# Utilisation
e = Employe()
m = Manager()

e.salaire()
m.salaire()
```

Appeler la méthode mère depuis la fille

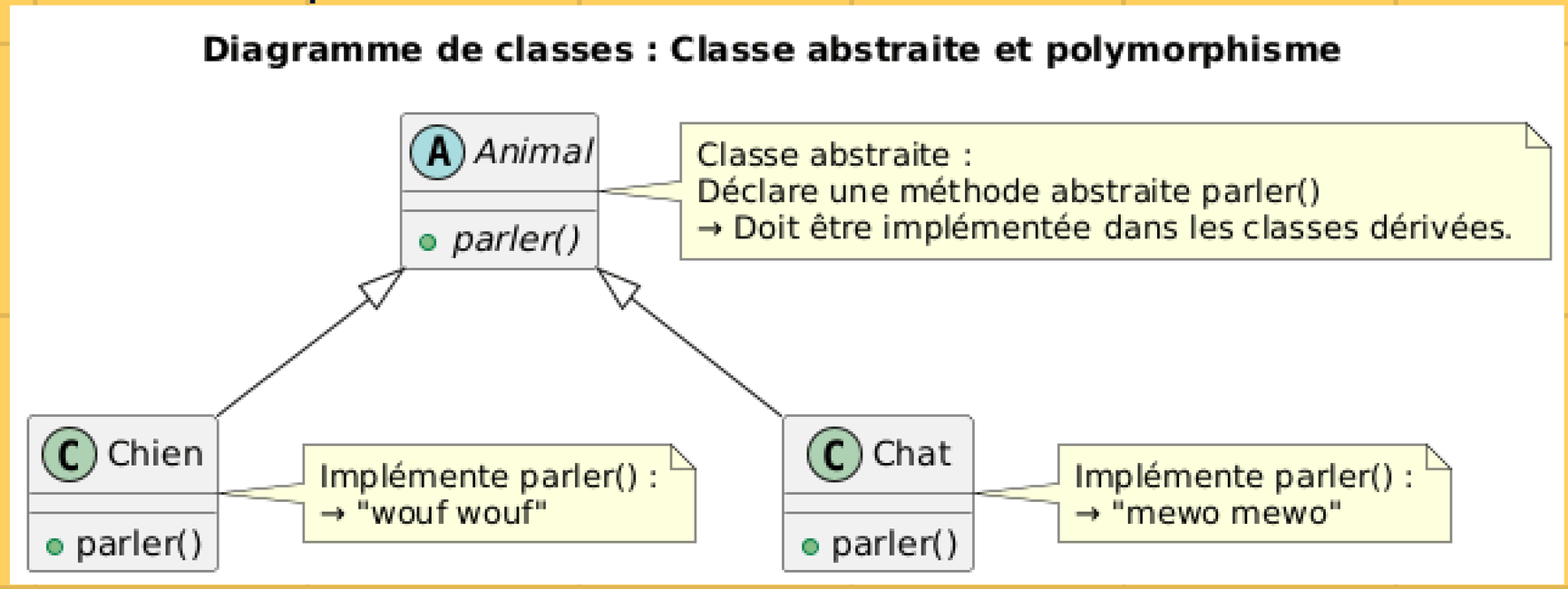


Appeler la méthode mère depuis la fille

```python
class Employe:
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def travailler(self):
        print("Travaille 8h par jour.")

Windsurf: Refactor | Explain
class Manager(Employe):
    Windsurf: Refactor | Explain | Generate Docstring | ×
    def travailler(self):
        super().travailler()  # Appel méthode de la classe mère
        print("Et gère une équipe.")

m = Manager()
m.travailler()
```

Polymorphisme et classes abstraites

e polymorphisme est souvent utilisé avec des classes abstraites (via le module abc), quand on veut forcer les classes filles à implémenter une méthode.



Polymorphisme et classes abstraites

Ici :
ABC = classe abstraite.
@abstractmethod oblige les classes filles à définir la méthode parler().

```python
from abc import ABC, abstractmethod

Windsurf: Refactor | Explain
class Animal(ABC):
    Windsurf: Refactor | Explain | Generate Docstrin
    @abstractmethod
    def parler(self):
        pass

Windsurf: Refactor | Explain
class Chien(Animal):
    Windsurf: Refactor | Explain | Generate Docstrin
    def parler(self):
        print("wouf wouf ")

Windsurf: Refactor | Explain
class Chat(Animal):
    Windsurf: Refactor | Explain | Generate Docstrin
    def parler(self):
        print("mewo mewo")

animaux = [Chien(), Chat()]
for a in animaux:
    a.parler()
```

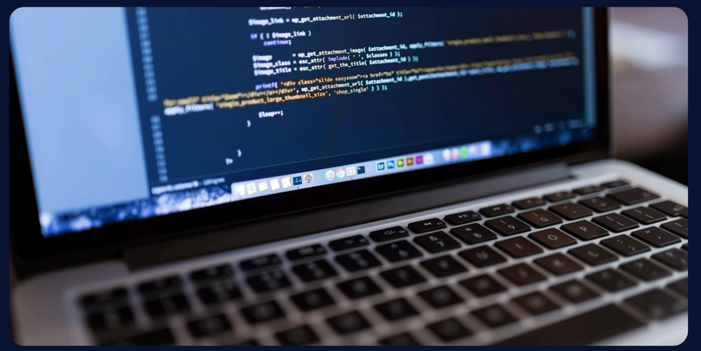
Résumé général

Concept	Description	Exemple
Héritage simple	Une classe hérite d’une seule autre	class Fille(Mere):
Héritage multiple	Une classe hérite de plusieurs classes	class C(A, B):
super()	Appelle une méthode de la classe mère	super().__init__()
Polymorphisme	Même méthode, comportement différent	a.parler()
Surcharge	Redéfinir une méthode héritée	def salaire(self): ...

Exercices

crée une hiérarchie de classes :

1. Classe mère : Employe
    - attributs : nom, salaire_base
    - méthode : afficher_salaire()
2. Classe fille : Manager
    - attribut supplémentaire : prime
    - surcharge la méthode afficher_salaire() pour afficher le total
3. Classe fille : Developpeur
    - attribut supplémentaire : langage
    - ajoute une méthode coder()



Thank You