Fiche récap : Comprenez les fondements de la POO

Développement

1. Définir des classes avec attributs et méthodes
2. Appliquer l'encapsulation pour sécuriser les données
3. Modéliser les relations entre objets par composition
4. Utiliser l'héritage et le polymorphisme efficacement

Code 💻

python

```python
# Définition d'une classe avec encapsulation
class Player:
    def __init__(self, nom, identifiant):
        self.nom = nom
        self._id_interne = identifiant
    def get_id_interne(self):
        return self._id_interne
    def __repr__(self):
        return f"Player(nom='{self.nom}', id={self._id_interne})"
# Héritage avec redéfinition et super()
class Personne:
    def __init__(self, nom):
        self.nom = nom
class Joueur(Personne):
    def __init__(self, nom, elo):
        super().__init__(nom)
        self.elo = elo
```

Attributs

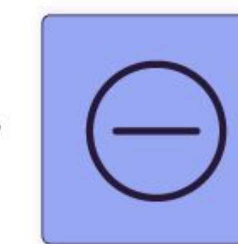
Solde

Titulaire

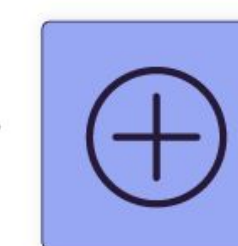
Compte Bancaire

Solde	500€
Titulaire	Dupont

Méthodes



Retirer



Déposer

Définitions 🔎

Objet
Entité logicielle concrète combinant données (attributs) et comportements (méthodes), issue d'une classe.

Classe
Modèle définissant la structure commune d'objets avec leurs attributs et méthodes.

Encapsulation
Principe de protection des données internes d'un objet via des accès contrôlés.

Polymorphisme
Capacité à utiliser une même interface pour des comportements différents selon l'objet.

Héritage
Mécanisme permettant à une classe fille de réutiliser et spécialiser les éléments d'une classe mère.

Bonnes pratiques

- ✅ Définir des classes claires pour chaque entité
- ✅ Utiliser l'encapsulation pour protéger les attributs
- ✅ Appeler super() dans les constructeurs de sous-classes
- ✅ Modéliser les relations fortes avec la composition
- ✅ Créer des mixins pour les rôles multiples
- ✅ Définir repr pour faciliter le débogage
- ✅ Utiliser un Manager pour centraliser la logique globale
- ✅ Préférer la composition à l'héritage quand possible

Erreurs classiques

- ❌ Modifier directement les attributs privés d'un objet
- ❌ Oublier d'appeler le constructeur parent avec super()
- ❌ Créer une instance sans passer par le constructeur
- ❌ Utiliser une méthode d'instance sans objet instancié
- ❌ Coupler trop de responsabilités dans une seule classe
- ❌ Utiliser l'héritage pour des comportements optionnels
- ❌ Redéfinir une méthode sans respecter l'interface
- ❌ Ignorer les cycles de vie des objets en composition