

Classe Statut

```
class Statut(OeuvreArt):
    def __init__(self, titre, annee_creation, createur, statut, valeur):
        super().__init__(titre, annee_creation, createur)
        self.__statut = statut
        self.__valeur = valeur

    def changer_statut(self, nouveau_statut):
        """
        Change le statut de l'œuvre d'art.

        Args:
            nouveau_statut (str): Le nouveau statut à attribuer à l'œuvre.
        """
        self.__statut = nouveau_statut

# Getters et setters pour les attributs statut et valeur...

def __str__(self):
    return f"{super().__str__()} - Statut : {self.__statut}, Valeur : {self.__valeur}"

# Exemple d'utilisation de la classe Statut
createur_statut = Createur("Artiste Statut", "N/A")
statut_oeuvre = Statut("Œuvre avec Statut", 2000, createur_statut, "Rare", 5000)

# Changer le statut de l'œuvre
statut_oeuvre.changer_statut("Commun")

# Afficher les informations actualisées
print(statut_oeuvre)

class Statut(OeuvreArt):
    def __init__(self, titre, annee_creation, createur, statut, valeur):
        super().__init__(titre, annee_creation, createur)
        self.__statut = statut
        self.__valeur = valeur

# Méthode imaginaire
def une_methode_imaginaire(self):
    # Implémentez ici votre méthode imaginaire
    pass

# Getters et setters pour les attributs statut et valeur
def get_statut(self):
    return self.__statut

def set_statut(self, statut):
    self.__statut = statut
```

```

def get_valeur(self):
    return self.__valeur

def set_valeur(self, valeur):
    self.__valeur = valeur

# Redéfinition de la méthode __str__
def __str__(self):
    return f"{super().__str__()} - Statut : {self.__statut}, Valeur : {self.__valeur}"

# Exemple d'utilisation de la classe Statut
createur_statut = Createur("Artiste Statut", "N/A")
statut_oeuvre = Statut("Œuvre avec Statut", 2000, createur_statut, "Rare", 5000)
print(statut_oeuvre)

```

Uml

Classe Musée

```

class Musée:
    def __init__(self, nom, adresse):
        self.__nom = nom
        self.__adresse = adresse
        self.__collection = []

    def get_nom(self):
        return self.__nom

    def set_nom(self, nom):
        self.__nom = nom

    def get_adresse(self):
        return self.__adresse

    def set_adresse(self, adresse):
        self.__adresse = adresse

    @property
    def collection(self):
        return self.__collection

    def ajouter_oeuvre(self, oeuvre):
        self.__collection.append(oeuvre)

    def une_methode_imaginaire(self):
        # Vous pouvez implémenter ici une méthode imaginaire

```

```

pass

def __str__(self):
    oeuvres_str = "\n".join(str(oeuvre) for oeuvre in self.__collection)
    return f"Musée {self.__nom}, Adresse : {self.__adresse}:\n{oeuvres_str}"

```

2eme cas

```

class Musée:
    def __init__(self, nom, adresse):
        self.__nom = nom
        self.__adresse = adresse
        self.__collection = [] # Liste pour stocker des objets OeuvreArt

    # Les autres méthodes de la classe Musée restent inchangées...

    def ajouter_oeuvre(self, oeuvre):
        self.__collection.append(oeuvre)

    def get_collection(self):
        return self.__collection

    def set_collection(self, collection):
        self.__collection = collection

```

Uml agrégation a OeuvrArt

```

+-----+
| Musée |
+-----+
| - __nom: str |
| - __adresse: str |
| - __collection: [] |
+-----+
| + get_nom(): str |
| + set_nom(nom: str) |
| + get_adresse(): str |
| + set_adresse(adresse: str) |
| + collection: [] |
| + ajouter_oeuvre(oeuvre: OeuvreArt) |
| + une_methode_imaginaire() |
| + __str__(): str |
+-----+

```

Classe oeuvre d'art

```
class OeuvreArt:
    def __init__(self, titre, annee_creation, createur):
        self.__titre = titre
        self.__annee_creation = annee_creation
        self.__createur = createur

    @property
    def titre(self):
        return self.__titre

    @titre.setter
    def titre(self, titre):
        self.__titre = titre

    @property
    def annee_creation(self):
        return self.__annee_creation

    @annee_creation.setter
    def annee_creation(self, annee_creation):
        self.__annee_creation = annee_creation

    @property
    def createur(self):
        return self.__createur

    @createur.setter
    def createur(self, createur):
        self.__createur = createur

    def une_methode(self):
        # Vous pouvez implémenter ici une méthode imaginaire
        pass

    def __str__(self):
        return f"{self.__titre} ({self.__annee_creation}) - Créateur : {self.__createur.nom}"
```

UML

```
+-----+
|  OeuvreArt  |
+-----+
| - __titre: str    |
| - __annee_creation: int|
| - __createur: Createur|
```

```

+-----+
| + titre: str      |
| + annee_creation: int |
| + createur: Createur |
| + une_methode()    |
| + __str__()        |
+-----+

```

UML

User

Avec la hiérarchie agrégation et héritage

ChatGPT

D'accord, voici une description textuelle de la hiérarchie des classes avec des indications sur les relations d'agrégation et d'héritage pour vous aider à créer des schémas UML plus détaillés :

Classe OeuvreArt :

Attributs : titre, annee_creation

Méthodes : __init__, __str__

Relation : A une association avec la classe Createur (association)

Classe Musée :

Attributs : collection (une liste d'objets OeuvreArt)

Méthodes : ajouter_oeuvre, __init__, __str__

Relation : Possède plusieurs OeuvreArt (agrégation)

Classe Createur :

Attributs : nom, nationalite

Méthodes : __init__

Classe Statue (hérite de OeuvreArt) :

Attributs supplémentaires : materiau

Méthodes : __init__

Relation : Hérite de la classe OeuvreArt (héritage)

Classe Tableau (hérite de OeuvreArt) :

Attributs supplémentaires : technique

Méthodes : __init__

Relation : Hérite de la classe OeuvreArt (héritage)

Voici comment vous pourriez représenter ces relations dans vos schémas UML :

Pour l'association entre OeuvreArt et Createur, vous pouvez utiliser une flèche simple avec la mention "A une" ou "Possède" pour indiquer l'association.

Pour l'agrégation entre Musée et OeuvreArt, utilisez un losange creux à côté de la classe Musée avec une flèche pointant vers OeuvreArt pour indiquer que Musée possède plusieurs OeuvreArt.

Pour l'héritage entre Statue et Tableau par rapport à OeuvreArt, utilisez une flèche avec une ligne en pointillé pour indiquer l'héritage de OeuvreArt vers Statue et Tableau.

Ces notations visuelles vous aideront à créer des schémas UML conformes à la hiérarchie d'agrégation et d'héritage que vous avez décrite dans votre projet.