OPENCLASSROOMS

Corrigé Activité P1C4

Le corrigé ci-dessous illustre l'utilisation du constructeur (__init__) pour Pour ce corrigé, nous considérons que la classe Player (avec son nom et _id_interne encapsulé) existe déjà, comme défini dans le chapitre précédent.

Python

```
# Hypothèse : la classe Player existe, avec son __init__(nom, id_interne)
# et sa méthode get_id_interne()
class Match:
    # Match contient (référence/Agrégation) deux Players
    def __init__(self, joueur_blanc, joueur_noir):
        # Utilisation des instances de Player comme attributs
        self.joueur_blanc = joueur_blanc
        self.joueur_noir = joueur_noir
        self.resultat = "En cours"
    def __repr__(self):
        return f"Match(Blancs: {self.joueur_blanc.nom}, Noirs: {self.joueur_noir.nom}, Résultat: {self.resultat})"
class Round:
    # Round possède (Composition) des Matchs
    def __init__(self, numero: int):
```

```
        self.numero = numero
        # Initialisation de la collection vide pour la Composition
        self.listeDesMatchs = []
    def ajouter_match(self, match: Match):
        # Ajout d'une instance de Match à la collection
        self.listeDesMatchs.append(match)
        print(f"Match ajouté à la Ronde {self.numero}.")
    def __repr__(self):
        return f"Round(Numéro: {self.numero}, Nb Matchs:
{len(self.listeDesMatchs)})"
class Tournoi:
    # Tournoi possède (Composition) des Rounds
    # Tournoi agrège des Players (qui lui sont passés)
    def __init__(self, nom: str, lieu: str, participants: list):
        self.nom = nom
        self.lieu = lieu
        # Agrégation : référence à des objets Players qui existent ailleurs
        self.participants = participants
        # Composition : la liste des Rondes est créée et dépend de l'existence
du Tournoi
        self.rondes = []
        print(f"Tournoi '{self.nom}' créé avec {len(participants)} joueurs.")
    def ajouter_ronde(self, ronde: Round):
```

```
        # Ajout d'une instance de Round à la collection (Composition)
        self.rondes.append(ronde)
        print(f"Ronde {ronde.numero} ajoutée au Tournoi {self.nom}.")
    def __repr__(self):
        return f"Tournoi('{self.nom}', Nb Rondes: {len(self.rondes)})"
# ---------------- Démonstration ----------------
# 1. Préparation des entités de base (Players)
# Ces instances sont indépendantes (Agrégation)
joueur_a = Player("Alice", 1) # Assurez-vous d'avoir la classe Player du C3
joueur_b = Player("Bob", 2)
# 2. Création de l'entité principale
grand_tournoi = Tournoi("Grand Open", "Paris", [joueur_a, joueur_b])
# 3. Création des composants (Composition)
ronde1 = Round(1)
match1 = Match(joueur_blanc=joueur_a, joueur_noir=joueur_b)
# 4. Construction de la hiérarchie
ronde1.ajouter_match(match1)
grand_tournoi.ajouter_ronde(ronde1)
# Vérification des liens (la Ronde est bien dans le Tournoi, le Match est dans
la Ronde)
# Et la structure est facilement inspectable grâce à __repr__
print("\nStructure du tournoi :")
print(grand_tournoi)
```

```
print(grand_tournoi.rondes)

print(grand_tournoi.rondes.listeDesMatchs)

# Accès aux joueurs agrégés via le Match (le joueur existe toujours même si le match est annulé)

print(f"ID du joueur blanc (via match): {grand_tournoi.rondes.listeDesMatchs.joueur_blanc.get_id_interne()}")
```

Ce corrigé illustre comment, en utilisant des listes d'instances (listeDesMatchs et rondes) comme attributs au sein des classes supérieures, nous implémentons la **Composition**. Si grand_tournoi était détruit, il n'y aurait plus de raison d'avoir ronde1 ni match1 (relation forte), même si les joueurs agrégés (joueur_a, joueur_b) persistent (relation souple).