P21 Guidelines

CM0 Introduction au Module

- Compiler régulièrement et tester régulièrement le code.
- Avoir un environement de développement correctement configuré (avec compilation et exécution rapide, debugger, tests, LSP, ...).
- Indenter le code.
- Le nom des variables et des fonctions commencent par une minuscule et respecte le camelCase
- Ne pas écrire `==true` ni `==false`.
- Une fonction ne doit effectuer qu'une seule tâche (et devrait se limiter à au plus une vingtaine de lignes)
- Ne pas faire d'affichage dans une fonction qui gère la logique de l'application.
- Vous pouvez utiliser les boucles de type "for each"
- Vous pouvez écrire une fonction contenant plusieurs return (c'est même conseillé lorsque cela simplifie le code !)

CM1 Les Classes

- Le nom des classes commence par une majuscule et respecte le camelCase.
- Le nom des attributs commence par `m_` (ou un autre préfixe clair indiquant un attribut)
- Les attributs d'une classe ne doivent pas être publiques (règle d'encapsulation forte)
- Définir la méthode `toString()` pour chaque classe.
- Chaque classe ne doit avoir qu'une seule responsabilité.

CM2.1 Agrégations, Association, UML

- Ne pas appeler des accesseurs (*getters*) sur des accesseurs, définir une méthode supplémentaire si besoin.
- Éviter les cycles.

CM2.2 Packages

- Le nom d'un package commence par une minuscule et respecte le CamelCase.

CM3.1 Interfaces et polymorphisme

- Spécifier en commentaire/documentation le contrat de chaque interface.
- L'interface doit rester minimale.
- Ne pas implémenter de méthodes `default` dans une interface.
- Ne pas utiliser `instanceof` ni `getClass()` en dehors de la méthode `equals(Object obj)`

CM3.2 Tests Unitaires

- Chaque test ne doit tester qu'une seule chose à la fois.
- Le résultat d'un test doit être prédictible.

CM4.1 Héritage

- S'assurer que chaque méthode de la classe mère peut être définie de façon cohérente dans la classe fille.
- Ajouter l'annotation `@Override` aux méthodes redéfinies.

CM5.1 Références

- Ne jamais utiliser l'opérateur `==` sur des `String` (utiliser la méthode `equals(Object obj)`)
- Quand cela est possible, utiliser les types primitifs plutôt que les types `Wrapper` (un type primitif ne peut pas être `null`)
- Ne pas utiliser `null` pour indiquer un conteneur vide, préférer utiliser `ArrayList<Double>()` plutôt que `null`.
- Lorsqu'une méthode peut dans certains cas ne pas renvoyer de résultat renvoyer `Optional<T>` plutôt que de renvoyer la référence `null`.