

Checklist itération 1 (officielle)

Préalable

“ Tous ces éléments sont à observer pour que le travail soit évaluable

- ☒ Une archive .zip est déposée sur Moodle.
- ☒ L'archive comporte un projet eclipse mentionnant le nom et le prénom (une infraction de nommage tolérée sur les 3 itérations)
- ☒ Le projet respecte la structure demandée (présence des répertoires src et tests, fichiers sources dans les bons paquetages. -0,25 sur la note finale par fichier mal placé)
- ☒ Les versions de Java, Junit et Eclipse sont conformes à celles demandées (une infraction tolérée sur les 3 itérations)
- ☒ Eclipse ne mentionne pas de problème de compilation (sauf problèmes de liaison au JDK, à JUnit 5)
- ☒ AI-1: Créer une partie
- ☒ AI-2: Voir une partie
- ☒ AI-3 : Revenir au menu principal

Acquis 1 : Programmer des énoncés de conception en Java selon les principes de la POO

“ Seuil de réussite (tous sont à valider pour obtenir l'acquis)

- ☒ [Prog. Procédurale] PMD relève au plus 2 méthodes trop longues et/ou trop complexes
- ☒ [Cohésion] PMD relève au plus 2 classes comportant 5 attributs ou plus.
- ☒ [Cohésion] PMD relève au plus 1 classe divine ou une classe de données

- ✓ [Encapsulation] PMD relève au plus 2 attributs d'accès autres que private
- ✓ [Encapsulation] Au moins 2 méthodes ou constructeurs respectent le principe de copies défensives pour les collections.
- ✓ [Masquage] Les classes métiers respectent la loi de Déméter (2 infractions tolérées)
- ✓ [RDD] La plupart des types du domaine correspondent aux candidats identifiés pendant l'étape de conception.
- ✓ [Polymorphisme] Les paramètres des constructeurs des superviseurs sont des types abstraits (interfaces ou classe abstraite)
- ✓ [Polymorphisme] PMD ne lève pas d'alerte LooseCoupling
- ✓ [Architecture] Les tests ArchUnit ne révèlent pas de problème d'architecture

Acquis 2 : Valider les comportements des objets programmés par des tests unitaires

“ Seuil de réussite (tous sont à valider pour obtenir l'acquis)

- ✓ Chaque classe du domaine est validée par une classe de test. Les classes de test sont définies dans le même package (mais dans le répertoire tests).
- ✓ L'ensemble des classes de test du domaine couvre au moins 90% des classes du domaine.
- ✓ Tous les tests unitaires des classes du domaine réussissent. (1 test en échec toléré)

Acquis 3 : Documenter son code

“ Seuil de réussite (tous sont à valider pour obtenir l'acquis)

- ✓ Les types et les méthodes publics sont documentés par de la javadoc inspirée par les énoncés (cf. PMD, 5 fautes tolérées)
- ✓ Les pré ou postconditions demandées figurent dans la javadoc, dans l'en-tête de la méthode `PlayGameSuperviser.onEnter`, et les réponses concernent les 3 classes `Player`,

CaseMap et Case et sont correctes.

☑ La CTT demandée est correctement (justifiées, avec le pire des cas si il existe, avec signification des variables N, M, etc) évaluée dans la Javadoc de la classe TreasureQuestGameFactory

Acquis 4 : Justifier le choix d'une structure de données particulière

“ Seuil de réussite (tous sont à valider pour obtenir l'acquis)

☑ Les choix de type de collection sont correctement documentés dans le code pour les cases du terrain de jeu, dans la javadoc de la classe CaseMap.

☑ Les choix d'implémentation sont correctement documentés dans le code pour les cases du terrain de jeu, dans la javadoc de la classe CaseMap.

Degré de maîtrise

“ Vérifiés SI tous les critères de seuil de réussite ont été observés pour affiner la cote

☑ PMD ne relève aucun problème de méthode trop complexe et/ou trop longue

☑ Plus de 2 méthodes ou constructeurs appliquent les principes de copies défensives sur les collections

☑ Au moins 2 constructeurs ou méthode de fabrique valident leurs paramètres.

☑ PMD ne relève aucun problème de type data class ou god object

☑ Les méthodes ne faisant pas partie du diagramme de classe ou des diagrammes de collaboration sont private (voire protected)

☑ Le packaging domain est couvert à 100% par les classes de tests du même package.

☑ La documentation de plusieurs méthodes fait état de précondition et de postcondition correctes

☑ Les collections sont utilisées sans réinventer la roue (pas de reprogrammation de méthodes existantes, ou de parcours séquentiel quand l'accès direct est prévu par exemple)

☐ Un plan de test exhaustif valide le code correspondant à la CTT demandée

Revision #8

Created 29 April 2023 12:46:00 by SnowCode

Updated 9 May 2023 09:06:23 by SnowCode