UNIVERSITÉ TOULOUSE III PAUL SABATIER

M1 Info DL, IHM, iLord
Développement Collaboratif, Qualité (EMINL1H1)
Examen Session 1 2018-2019 19/11/2018

Durée : 2h00 (1h50 de composition + 10min pour le rendu git).
Tout document est autorisé.
Il est strictement interdit de communiquer de manière directe ou indirecte, de quelque manière que ce soit (messagerie instantanée, e-mail, SMS, réseaux sociaux, communication orale, etc.) avec une personne autre que les surveillant(e)s.
Toute violation de cet interdit entraînera une sanction lourde et est susceptible de poursuite pour fraude à examen.
Vous pouvez travailler sur vos propres machines. Vous êtes alors responsable de tout problème technique survenant pendant la séance (panne matériel, panne wifi, panne système, etc.). Votre machine doit alors être configurée avec IntelliJ Ultimate Edition, Maven, Git, et dotée d'un JDK 1.8.
Suite à certains problèmes, pour les machines de l'UPS, il faudra lancer IntelliJ sous Linux et pas sous Windows.
La sauvegarde de votre projet dans le sous-dossier *~/Documents/* n'est pas nécessaire, et est même déconseillée (pour éviter certains warnings).
Le barème ici est juste indicatif.

---

1. Création et initialisation du projet Maven "question-score" (2 points)

- Créez sur votre poste de travail un projet Maven ayant les caractéristiques suivantes :
    - groupId : deqo
    - artifactId : question-score
      (vous choisirez également "question-score" comme nom du projet IntelliJ)
- Votre projet utilisera JUnit en version 4.12 pour la création des tests unitaires. Modifiez le fichier pom.xml de votre projet pour exprimer la dépendance à JUnit 4.12.
- Votre projet utilisera les outils d'analyse statique de code Checkstyle et Findbugs, ainsi que Cobertura pour mesurer la couverture du code par les tests sur votre projet. Modifiez le fichier pom.xml de votre projet pour pouvoir générer les rapports Checkstyle, Findbugs, et Cobertura. Vous pouvez également activer le plugin Maven Jxr pour vous faciliter la consultation des rapports.
- Créez un fichier texte "README.md" à la racine du projet en y écrivant votre prénom et votre nom.

Votre travail consiste à reprendre un projet existant visant le calcul du score obtenu par un étudiant à une question à choix multiple ou à choix exclusif.

- Récupérez le dossier "src" fourni sur Moodle puis remplacez le dossier "src" de votre projet par le dossier "src" récupéré.
- Étudiez le code source de votre projet. Vérifiez qu'il compile normalement. Vous pouvez lancez différentes commandes Maven pour vérifier que votre configuration fonctionne comme attendue (génération des rapports OK, etc.).

2. Supervision du projet "question-score" par Git (1 point)

- Faites en sorte que votre projet soit supervisé par Git. On s'attachera à vérifier que le dossier ".git" soit bien à la racine du dossier "question-score".
- Faites en sorte que les dossiers "target", ".idea" et les fichiers "*.iml" soient ignorés par Git.
- Effectuez un premier commit permettant d'ajouter votre projet sur la branche master de votre dépôt Git local.
- Cliquez sur le lien fourni sur Moodle et suivez les instructions pour créer automatiquement un **dépôt privé** pour votre TP au sein de l'organisation GitHub DCLL-MDL.
- Après avoir localement configuré le remote correspondant à votre dépot privé sur Github, poussez votre branche master sur votre dépôt distant.

Les questions suivantes doivent être traitées dans l'ordre. **Si vous manquez de temps pour traiter toutes les questions, il est indispensable de consacrer les 10 dernières minutes de l'épreuve à la question 8 pour effectuer le rendu de votre travail**.

3. Amélioration du respect des conventions de codage (3 points)

- Créez une nouvelle branche intitulée "conventions". Basculez sur cette nouvelle branche.
- Modifiez le code des classes de votre projet de telle sorte que :
    - le nombre d'erreurs Checkstyle soit diminué de 25% (**exemple**: **si** initialement il y a 100 erreurs, il faut tomber à 75 erreurs);
    - les nombres de bugs et d'erreurs Findbugs soient de 0.
- Commitez vos modifications.
- Fusionnez vos modifications sur la branche "master".
- N'envoyez pas encore les modification vers le dépôt distant, ceci est à faire à la question 8.

---

4. Couverture du code par les tests de la classe QuestionAChoixExclusif (3 points)

- Créez une nouvelle branche (et basculez dessus) intitulée "testsQuestionAChoixExclusif".
- Créez la classe de test unitaire permettant de tester la classe *QuestionAChoixExclusif*.
- Écrivez les tests unitaires permettant de couvrir le code de la classe *QuestionAChoixExclusif* à 100% (pour cette question, aucun changement dans la classe à tester n'est autorisé).
- Commitez vos modifications.
- Fusionnez vos modifications sur la branche "master".

5. Couverture du code par les tests de la classe QuestionAChoixMultiple (3 points)

- Créez une nouvelle branche intitulée "testsQuestionAChoixMultiple".
- Créez la classe de test unitaire permettant de tester la classe *QuestionAChoixMultiple*.
- Écrivez les tests unitaires permettant de couvrir le code de la classe *QuestionAChoixMultiple* à 100% (pour cette question, aucun changement dans la classe à tester n'est autorisé).
- Commitez vos modifications.
- Fusionnez vos modifications sur la branche "master".

6. Couverture du code par les tests de la classe ScoreCalculateur (4 points)

- Créez une nouvelle branche intitulée "testsScoreCalculateur".
- Créez la classe de test unitaire permettant de tester la classe *ScoreCalculateur*.
- Écrivez les tests unitaires permettant de couvrir le code de la classe *ScoreCalculateur* à 100% en utilisant les contraintes et les cas de tests suivants (aucun changement dans la classe à tester n'est autorisé):
  - pour chaque cas de test, l'instance de *ScoreCalculateur* utilisée est testée sur une question de type *QuestionAChoixMultiple* initialisée de la manière suivante : questionAChoixMultiple = new QuestionAChoixMultiple("q1",new ArrayList<Integer>(Arrays.asList(2,3,5)));
  - quand on calcule le score pour une liste de réponses contenant les valeurs 1 et 4, on obtient 0 comme résultat ;
  - quand on calcule le score pour une liste de réponses contenant les valeurs 2 et 3, on obtient 2*100/3 à 0,01 près comme résultat[1] ;
  - quand on calcule le score pour une liste de réponses contenant les valeurs 2, 3 et 5, on obtient 100 à 0,01 près comme résultat.
- Commitez vos modifications. Fusionnez vos modifications sur la branche "master".

7. Correction de l'application (4 points)

- Créez une nouvelle branche intitulée "corrections".
- Complétez les tests unitaires de la classe *ScoreCalculateur* en ajoutant les cas de test suivants :
  - quand on calcule le score pour une liste de réponses contenant les valeurs 1,2,3,4,5 on obtient 0 à 0,01 près comme résultat (voir note de bas de page) ;
  - quand on calcule le score pour une liste de réponses contenant les valeurs 1,2 et 3 on obtient 16.66 à 0,01 près comme résultat ;
- Après avoir constaté que ces tests ne passaient pas, modifiez le code de l'application et des tests (si nécessaire) afin que **tous les tests passent**. (Conseil : assurez-vous que vos modifications ne créent pas de nouvelles erreurs Checkstyle.)
- Commitez vos modifications.
- Fusionnez vos modifications sur la branche "master".

8. Rendu du travail sur Github

À la fin de l'épreuve, effectuez les actions suivantes (dans un terminal git-bash) :

- **créez un tag "last" sur votre dernier commit effectué** en utilisant la commande suivante :
  `git tag -a last -m "Last commit"`
- **poussez ce tag** en utilisant la commande: `git push origin last`
- **poussez toutes les branches créées lors des questions précédentes** (si vous avez traité toutes les questions, cela concerne donc master, conventions, testsQuestionAChoixExclusif, testsQuestionAChoixMultiple, testsScoreCalculateur, corrections).

---

[1] pour tester une égalité à 0,01 près: https://junit.org/junit4/javadoc/latest/org/junit/Assert.html#assertEquals(double,%20double,%20double) avec delta=0.01