Travail pratique 1 – Partie 3

Professeurs : Sylvain Labranche et Abdelhabib Yahia

Adaptée par : Dorsaf Haouari

Partie 3 :

Monsieur Barette semble satisfait de votre dernière remise et commence à s'enrichir avec son restaurant. Cependant, il ne vous fait pas totalement confiance et aimerait que vous lui fournissiez un logiciel correctement testé. Vous devez donc écrire des tests unitaires pour votre programme. De plus, les spécifications suivantes s'ajoutent :

- Il peut y avoir des commandes incorrectes. Une commande est incorrecte si :
    - Le nom du client ou le plat n'existe pas (vu précédemment).
    - Le format d'un client, d'un plat ou d'une commande n'est pas respecté.
    - Des chiffres sont erronés (prix du plat, quantité de plats commandés).
- En sortie, on souhaite avoir une facture affichée à l'écran (terminal) **et** éditée dans le fichier (**Facture-du-***date-heure***.txt**).
- Dans ces sorties (**Terminal**, **Facture-du-***date-heure***.txt)**, vous devez afficher toutes les commandes incorrectes (les détails de la commande et la raison de l'erreur) avant les factures. L'ergonomie de l'affichage est laissée à votre discrétion.
- Si un client a une facture de 0$, sa facture n'est pas affichée dans le fichier de sortie.
- En revanche, les taxes sont maintenant appliquées (à raison de TPS 5% et TVQ 10%).

Directives :

1. Ce travail devrait être séparé sur deux packages (**main** : incluant les fonctionnalités de l'application et **test** contenant les cas de test implémentés).
2. Avant de commencer l'implémentation des nouvelles fonctionnalités :
    a. Créer un fichier journal **suivi.txt**, dans lequel vous allez partager les tâches. Vous indiquez le nom du membre suivi par les fonctionnalités qu'il va se charger d'ajouter. Ce fichier doit être mis sur GitHub dès le premier jour (premier commit avec le commentaire : Début de la partie 3 du projet AQL - Tâches).
    b. Créer les spécifications détaillées pour chaque fonctionnalité (chaque coéquipier doit détailler les spécifications des fonctionnalités liées à ses tâches).
        i. Posez des questions à votre client pour clarifier les situations ambiguës.
        ii. Chaque membre doit mettre ses spécifications dans un fichier texte sous format : **Specifications-Prénom-Nom.txt**
        iii. Faire un deuxième/troisième commit avec la mention « Spécifications de Prénom Nom ».
    c. Récupérer les spécifications du coéquipier, et commencer le développement des tests unitaires correspondants aux fonctionnalités du coéquipier (application de la méthode TDD). Si besoin, demandez des clarifications à votre coéquipier. S'il y

a des nouvelles clarifications, elles doivent être mises à jour dans le fichier de spécification correspondant.

d. Une fois que les tests de votre coéquipier sont terminés, Vérifiez s'ils ne présentent pas des anomalies et demandez à votre coéquipier de les corriger s'il y a lieu.
e. Commencer le développement de vos fonctionnalités qui devraient satisfaire les tests mis en place par votre coéquipier.
f. Une fois les tests des nouvelles fonctionnalités terminés, implémentez d'autres tests liés aux précédentes fonctionnalités développées durant la partie 2 du projet.

3. Chaque membre de l'équipe doit au moins créer 6 tests unitaires pour votre programme (nouvelles et anciennes fonctionnalités), en utilisant le Framework JUnit.
4. En utilisant **EclEmma,** assurez-vous d'obtenir une couverture de votre code d'au minimum **75%**. Utilisez la mesure **Line Counters** de EclEmma.
5. Vous devez mettre en place une plate-forme de gestions de bogues pour votre équipe et assigner au moins 1 bogue à chaque membre de l'équipe et le résoudre.
6. Une fois que vous avez réalisé toutes les étapes précédentes, votre programme devrait être fonctionnel. Vous devez faire un commit sur la branche master pour sauvegarder l'état de votre projet à ce niveau. Le commit doit obligatoirement avoir le message suivant : « **version 1 du projet – avant réusinage**».
7. Vous devez faire un **ré-usinage** correct de votre code :
    a. Chacun des membres de l'équipe doit effectuer le **ré-usinage** du code de son coéquipier.
    b. Pour ce faire, vous devriez créer des **pull-request** afin de faire le suivi cohérent des modifications et des corrections appliquées.
    c. Les **pull-request** devraient être validés ou refusés par les coéquipiers qui ont initialement créés ces fonctionnalités.
8. Une fois rendue à cette étape, votre code est ré-usiné et devrait être fonctionnel. Vous devez faire un commit sur la branche master pour sauvegarder l'état de votre projet à ce niveau. Le commit doit obligatoirement avoir le message suivant : « **version 2 du projet – après réusinage**».
9. Vous devez avoir une branche ***master*** ou ***prod*** et une branche ***dev***. En tout temps, le professeur pourra faire un ***pull*** de votre projet et la branche de production **doit** s'exécuter sans erreur et ne pas contenir de bogues. Le développement se fait sur la branche ***dev***.
10. Chacun des membres de l'équipe doit faire au moins 6 commit et un pull-request (dans cette étape, l'échange entre les coéquipiers est très important. Vous devrez faire des pulls à chaque fois avant de commencer à produire des choses qui peuvent être faites par votre coéquipier).
11. **Points Boni :** Celui qui utilise les mocks pour faire les tests unitaires d'au moins 3 méthodes différentes, aura 2 points boni sur le projet et ceci sans dépasser la note maximale.

Procédure de remise :

- Le professeur fera un *pull* du dernier *commit* effectué sur la branche master juste avant la date de remise.
- Vous devez envoyer un courriel à l'adresse suivante cours4b4@gmail.com dans lequel vous enverrez **l'url du dépôt distant** et **l'url de la plateforme de gestion de bogues**, indiquez les **noms de votre équipe** ainsi que vos **identifiants correspondants de Github**.
- Si la plateforme de gestion de bogues que vous utilisez est privé, vous devez me donner accès en m'envoyant une invitation à mon compte mail cours4b4@gmail.com

Barème :

Cette partie compte pour 10% de la session selon le barème suivant :

Tests unitaires	6 points
Fonctionnement du logiciel	2 points
Gestion des bogues et couverture de code	2 points

Ces critères ne sont pas exhaustifs. D'autres erreurs pourraient vous faire perdre des points. Chaque élément des spécifications et des directives sera évalué.