Université de Nice-Sophia Antipolis
E.S.I.N.S.A. 1re année

Anonyme

Projet de Java : un jeu de Quizz
Correction

Bon travail **14.75/20**

Travail sérieux. Attention néanmoins à mieux penser l'organisation de vos classes, attributs et méthodes, et à ne pas abuser des champs publics.

Divers détails dans votre `main`, dans `Quizz.java` :

ligne 15 : `static boolean jouer` devrait être défini par `boolean jouer` localement dans le `main` ;

lignes 38 et 43 : `double niveau2 = demanderDouble() ;` suivi immédiatement par `niveau = niveau2 ;`, à remplacer par `niveau = demanderDouble() ; ;`

ligne 50 : le nombre maximal de questions devrait être demandé au `Jeu` :
`System.out.print("Entrez un nombre de questions entre 1 et " + Jeu.TailleTabKestion() +" : ") ; ;`

ligne 53 : `int nbDeQuestionChoisis = (int)demanderDouble() ;` écrire une méthode `int demanderEntier()` qui ne fasse pas de *cast* eût été préférable ;

lignes 66 et 71 : encore une fois, il était plus simple d'écrire `nbDeQuestionChoisis = demanderEntier() ;`.

L'écriture des méthodes `demanderDouble()` et `demanderRejouer()` est une bonne initiative. On pourra néanmoins réécrire `t.compareTo("non")==0` en : `t.equals("non")`, et retourner un booléen dans `demanderRejouer()`. Offrir des choix par défaut dans une interface textuelle aurait aussi été appréciable (si l'utilisateur appuie sur « entrée » directement, alors mettre la difficulté par défaut par exemple).

2.2 Mots clefs pondérés 2

Vous faites un bon usage de mots clefs pondérés. L'idée d'utiliser des pondérations négatives sur certains mots clefs est bonne.

Les problèmes viendraient plutôt d'une organisation brouillonne de vos classes et de vos champs trop souvent publics. Par ailleurs vous ne vérifiez pas que les réponses finales correspondent aux mots clefs (comme vous avez des mots clefs négatifs, il faut seulement vérifier qu'au moins un mot clef positif apparaît dans la réponse officielle), qu'il existe au moins un mot clef positif pour chaque question, et que le nombre de réponses officielles est bien le même que celui des questions. À ce propos, `public static int TailleTabKestion()` s'écrit simplement `return questions.length ;` dans `Jeu.java`.

2.3 Pertinence 2

Le calcul de la pertinence aurait été pu être mieux réparti entre `Reponse` et `MotClef` : la gestion des accents, majuscules, ... se faisant dans `MotClef`, et la pondération finale dans `Reponse`.

2.4 Niveau de difficulté et scores 1

Pas de problème particulier. Il est dommage de ne pas avoir proposé de difficulté par défaut.

3 Fonctionnalités avancées 2.5

3.1 Gestions des majuscules 1

Parfait.

3.2 Gestions des accents 1

Incomplète : d'une part, vous ne considérez que le cas où l'utilisateur ajoute des accents inutiles et pas le cas où il en oublie, d'autre part les accents possibles en français sont bien plus nombreux : àâéèêëîïôùûüçÀÂÉÈÊËÎÏÔÙÛÜÇ au minumum.

3.3 Pondérations négatives 0.5

C'est une bonne idée !

4 Conception 2.5

On estime généralement que 80% à 90% des efforts investis sur un logiciel le sont dans sa maintenance. Dès lors, une bonne conception de départ est indispensable pour permettre d'étendre et d'améliorer le logiciel de départ. Vous allez en faire

vous-même l'expérience puisque le projet du second semestre réutilise votre travail du premier semestre, sur lequel vous devrez greffer de nouvelles fonctionnalités.

4.1 Modularité 1.25

Plusieurs défauts de ce point de vue, surtout la classe `Jeu` qui n'est qu'une réserve de méthodes et de champs `static`, sans même un constructeur !

Les autres classes sont assez bien écrites, mais comme indiqué précédemment pour le calcul de la précédence, il faudrait mieux répartir les tâches entre les classes : par exemple, une question doit pouvoir fournir son intitulé, sa réponse officielle, et un moyen d'évaluer une réponse de l'utilisateur. Ces deux dernières fonctions sont déléguées à la réponse, qui délègue à son tour (au moins partiellement) la dernière au système de mots clefs.

L'utilisation de champs publics à outrance vous dissimule l'effort nécessaire d'organisation.

4.2 Concepts objets 0.25

Et justement, vous abusez des champs `public` et des méthodes `static` ! L'intérêt des classes est de pouvoir construire un objet dynamiquement (et non statiquement) qui va fournir une interface proposant diverses opérations sur son contenu privé (retourner une réponse, calculer une pertinence, ...) *via* des méthodes publiques, contrôlant ainsi exactement ce qui est fait.

Votre code permet par exemple l'accès direct au tableau des questions, que n'importe quel programmeur fou peut changer pour un tableau de questions ne respectant aucune contrainte (par exemple sans mot clef, avec des réponses sans rapport avec la question, ...).

4.3 Utilisation avancée de l'API Java 1

L'utilisation de `String.toLowerCase()` et de `String.split()` est pertinente.

5 Packaging 2.5

La présentation de votre projet est un point important, et il le sera de plus en plus. Il est en particulier recommandé de créer des répertoires séparés `src/` et `doc/` pour le code source et la documentation. Un fichier `LISEZMOI` à la racine présentant un minimum le travail effectué est aussi appréciable. Indiquez de même les noms des intervenants du projet dans un fichier `AUTEURS`. Évitez les noms de fichiers avec des espaces ou des accents qui sont encombrants sur certains systèmes.

Le respect des conventions de code rend le code plus lisible, pour vous, pour votre correcteur, et à l'avenir pour les autres membres de votre équipe de développement. Une bonne documentation devrait permettre à une personne n'ayant pas accès au code source (mais uniquement au *bytecode* des fichiers `.class`) d'utiliser votre travail sans mauvaises surprises (d'où l'intérêt des antécédents, conséquents et exceptions).

5.1 Documentation 2

Bon effort de documentation.

5.2 Clarté du code 0.5

Bon effort sur les commentaires dans le code, mais ce dernier est parfois un peu noyé. Quand vous écrivez des lignes et des lignes de commentaires pour expliquer votre implémentation, posez-vous la question : ne pourriez-vous pas mieux organiser ce code de manière à le rendre plus clair ? (Attention, il s'agit ici de commentaires d'implémentation, pas de `javadoc`) Globalement, le code le mieux maîtrisé dans votre projet est celui qui est le moins commenté, car mieux organisé, et dans ces cas votre documentation suffit.

Par ailleurs, n'hésitez pas à revenir à la ligne à l'intérieur d'une instruction : un bon moyen pour cela est de garder votre fenêtre Emacs à sa taille par défaut (80 colonnes), et d'insérer des retours chariots dans les instructions enroulées.