

**Commandes des sapins**

**Phase finale**

**Description du projet**

Projet TDD – Groupe 6

[2I4020] 2I - Architecture et qualité logicielle

2018-2019

Arthur VENON

Mohamad MROUE

Julien STILMANT

Yassine AKHARAZE

**Introduction**

Dans le cadre du cours d’architecture et qualité logiciel, nous devons concevoir une application résolvant une problématique que nous avons choisi. Le but de ce projet est de concevoir cette application en utilisant le “Test-Driven Development” (TDD) qui consiste à penser d’abord au test que nous allons soumettre à notre code avant d’écrire le code en lui-même. Cela permet d’avoir une idée concrète sur la manière dont nous allons programmer notre application.

De plus, nous allons soumettre notre code a des règles de codage que nous aurons établie avant l’écriture de celui-ci.

Enfin, nous allons soumettre notre code à des critères de qualités qui devront être obtenus en fin de développement.

**Description de la problématique**

La période de Noël arrive à grand pas et la vente de sapin de noël ne va pas tarder à exploser. Nous avons eu comme idée de créer une application pour un magasin qui faciliterait le choix, la décoration et la commande de sapin afin que chacun puisse trouver son sapin idéal et passer de merveilleuse fête de Noël.

**Description de l’application**

Concrètement, le projet consiste à fournir à un magasin un outil de gestion complet qui lui permettra de gérer les commandes des sapins, de générer les factures des clients ainsi que réaliser la gestion des stocks et des clients.

L’application est contrôlée par le vendeur qui choisit la configuration des sapins à commander avec l’aide client.

Le client aura le choix entre plusieurs types de sapins : Naturel, artificiel mais en bois ou artificiel en plastique. Il pourra aussi s’il le veut choisir la décoration de celui-ci.

Enfin, quand le client est satisfait de son choix, l’application va lui générer sa facture et fournir au vendeur la commande. Ensuite en fonction de la disponibilité et du temps de commande, le vendeur donnera une date de livraison au client.

**Outils utilisés**

Afin de réaliser notre application, nous avons décidé d’utiliser un design pattern de type décorateur afin de modéliser les sapins et leurs décorations.

En ce qui concerne les conventions de codage, nous avons décidé de nous baser sur les conventions de Google Java Style. Nous avons déjà choisi la majorité des règles que nous allons suivre mais nous devons encore débattre sur quelques-unes.

Nous avons choisi comme critères de qualité, un code bien documenté afin de faciliter la compréhension ainsi qu’une rapidité d’exécution élevée.

Enfin, en ce qui concerne la base de données, celle-ci sera réalisée à partir d’un fichier texte.

**Documentation User**

1. Exécution : Pour lancer le programme, faites un build/run sur intelliJ ou Netbeans

Si vous lancez le programme en tant que projet maven le plugin mockito est déjà présent, si vous lancez le code source comme un simple projet java il vous faudra rajouter le .jar de Mockito dans les ressources du projet. Pour maven les commandes à utiliser sont les mêmes que notre job jenkins

1. Entrez les coordonnées du votre client (nom, prénom, ...)
2. Choisissez le type d’arbre que le client voudrait
3. Choisissez le type de commande : standard avec des décorations déjà choisies ou personnalisée
4. Choisissez les différentes décorations que le client veut sur son sapin
5. Tapez la commande ‘q’ lorsque la commande est terminée
6. Un fichier texte représentant la facture s’enregistre

**Règle de codage**

Pour la règle de codage on utilise directement les conventions google à l’aide de google\_checks.xml car c’est un standard de l’industrie et cela permet d’utiliser des plugins d’automatisations de formatage pour avoir un développement plus rapide.

**Résultat métrique**

Le test de performance est réalisé à l’aide d’un timeout de 20 millisecondes dans la classe JUnit PerformanceTest. Résultat : réussite

Résultat JAVADOC : La javadoc est en HTML dans targer/site/apidocs/christmastree/ La javadoc semble est présente pour chaque classe et méthode n’étant pas triviales comme les setters, getters et toString.

Le code n’a plus de warning et d’erreur de checkstyle Google

**Résultat duplication**

Checkstyle affiche 0 duplication et PMD ne donne pas de warning ou d’erreur

Coverage :

Grâce à Mockito on peut simuler les user inputs. Les tests des classes Trees, de Decorator et des méthodes de clientMain sont testées.

**Sondage**

https://www.survio.com/survey/d/Q9D5B9K9U5U8Q1B9C

Tous les résultats sont disponibles dans target/ :

-site/apidocs

-corbertura

-pmd.xml

-checkstyle-result.xml

-surefire-reports

-et autres (.jar, ...)

**Ce qu’il reste à faire**

Dans le code nous n’avons pas eu le temps de faire d’interface graphique et nous avons privilégier le respect des règles et des critères de qualité.

PMD et checkstyle nous disent qu’il n’y a pas de duplication de code en observant nous même le code on a l’impression d’en observer (notamment dans les constructeurs des décorateurs).

Certains if et switch ne sont pas testé car ils sont dans le main .

La description de nos java doc se contente d’une ligne de commentaire car la majorité des classe et méthodes sont suffisamment simples.

Nous n’avons pas mis beaucoup de commentaire hors javadoc dans notre code.

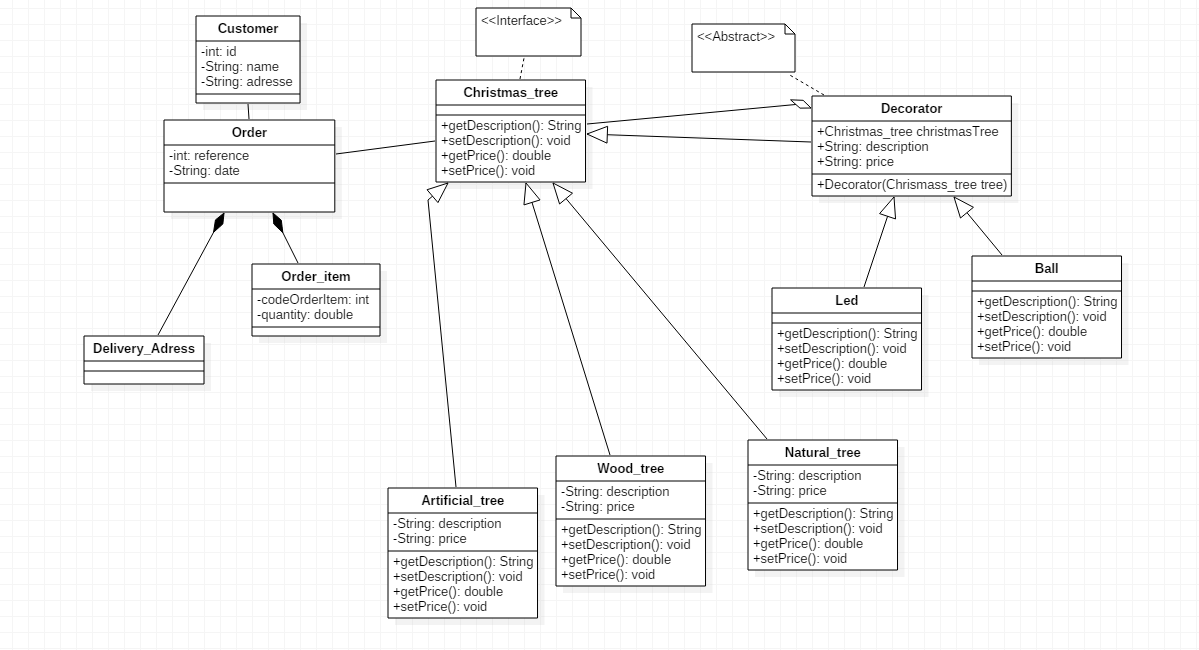
Sur certains PC ou IDE il y a un accès refusé quand on veut créer le fichier de facture, de ce fait on n'a pas vraiment pu tester la classe Bill et la création de facture. Des tests qu’on a pu faire de la classe Bill elle fonctionne mais a des problèmes pour mettre plusieurs décorateurs dans le fichier

**Conclusion**

En conclusion, nous avons réalisé une bonne partie de ce que nous voulions faire (règle de codage, performance, javadoc) hormis une partie des test unitaires, dù à du retard on a pas pu faire tester le sondage a beaucoup de gens.

**Annexe**

**Diagramme de classe : premier jet :**



**Diagramme final**Une image contenant texte, carte

Description générée avec un niveau de confiance très élevé