1. ***Modèle Objet du calcul de Taxes***

Nous pouvons partir sur un Modèle Objet permettant à partir des informations de l’énoncé, de définir les Classes suivantes

Un Panier/Commande :

Qui sera une Collection de produits et dont on pourra imprimer une facture contenant le montant total des achats ainsi que le montant des taxes.

Un Produit :

Qui pourrait être défini par son nom (e.g : livre, CD Musical,..) son prix, Son origine (importé ou pas) duquel dépendrons certaines taxes et son type (Livre, Nourriture,..) duquel dépendrons également certaines taxes.

L’Origine des produits :

Permettant de savoir si un produit est importé ou s’il s’agit de produits locaux (les taxes s’appliquant différemment à l’un ou l’autre).

Il est défini par son nom et un taux.

Le Type de Produits :

Permettant de savoir si un produit est classé livre ou Médicament, ou nourriture ou autre.

Locaux (les taxes s’appliquant différemment à l’un ou l’autre).

Il est défini par son nom et un taux.

Afin d’avoir une Cohérence forte sur les classes, Les deux Classes origine des produits et Type de produits permettront de savoir quel taux appliquer pour chaque produit pour calculer le montant des taxes mais ne calculeront pas d’elles même les taxes. Le calcul de taxe sera ainsi délégué à une autre Classe.

La Taxe :

Classe qui porte la logique de calcul des taxes (prix \* taux). Le taux étant fonction du type de taxe (produits importés, tva,…).

Pour éviter une création abusive de Taxes, Types de produits et Origine de produits nous empêcherons leur création à volonté (constructeur privé)

Une classe utilitaire

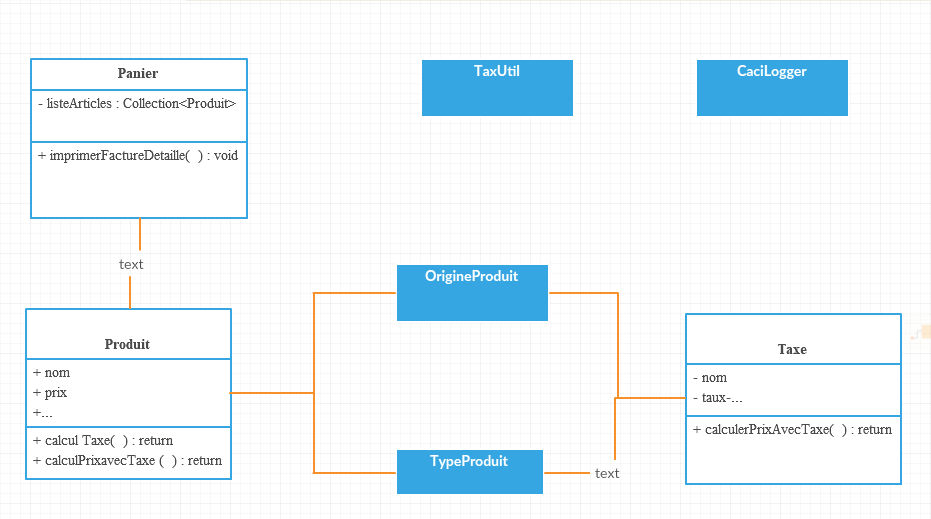
Pour la fonction d’arrondi défini dans l’énoncé.

Une Classe pour la gestion des logs.

Ce modèle permet de donner d’apporter de la flexibilité au programme, facilement extensible et adaptable (minimum de ligne de code) en cas d’ajout d’un nouveau Type de produit de Taxe tout en limitant les possibilités de modifications pouvant entrainer des abus.

Remarque : On aurait pu également partir sur le modèle d’une fabrique abstraite (« Abstract Factor »), permettant de fabriquer une famille de produits /Taxes et le patron  « Stratégy » pour implémenter le comportement de chaque produit (définis via une interface commune).

***Diagramme de classe simplifié :***



1. ***Pour Tester :***

Cloner le repo Git  sur l’url : <https://github.com/tksteph12/cacitaxe.git>

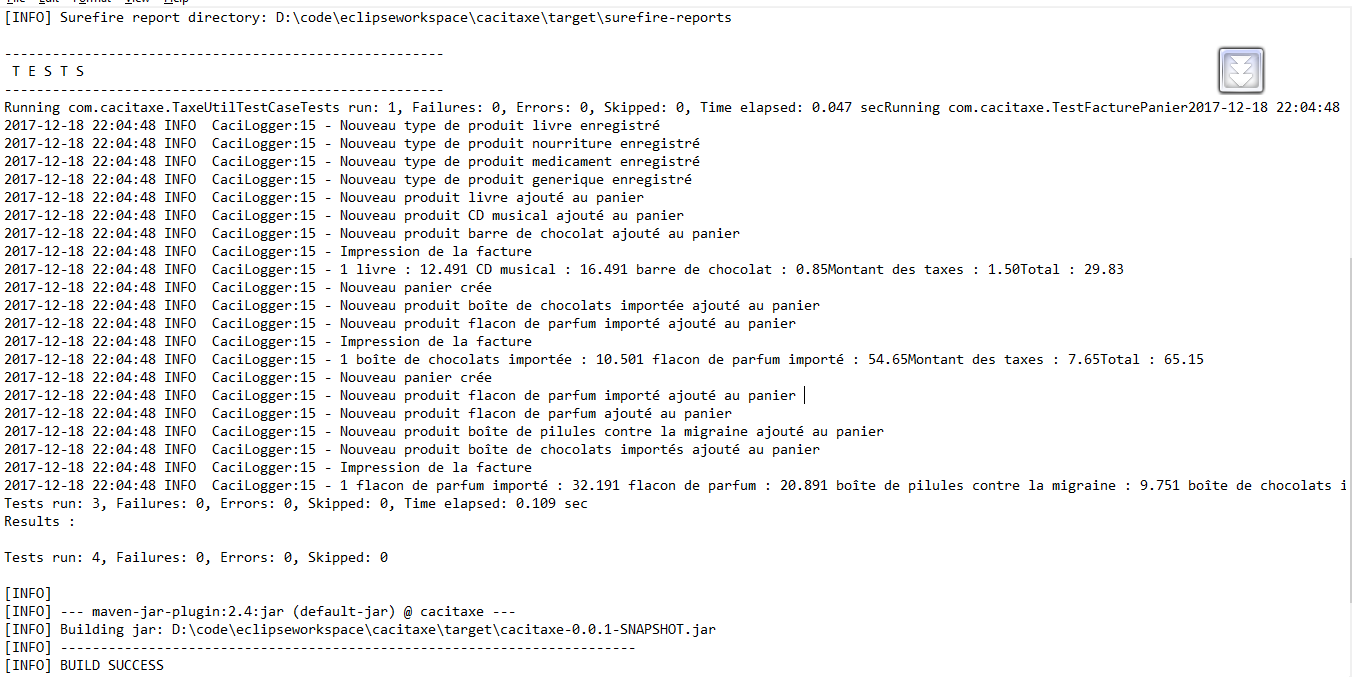
Vérifier que votre poste dispose d’une JVM 1.8,

Vérifier que vous disposez d’une installation de maven.

Naviguer dans le répertoire du projet

Double cliquer sur le fichier build.bat (Windows) ou build.sh (Unix)

Ouvrir le fichier caci-build.log et vérifier que le build s’est terminé correctement (comme ci-contre)



Vous pouvez également ouvrir le fichier caci-kata.log pour avoir le détail des tests d’impression de facture exécutés.

Importer le projet sous Eclipse pour avoir les détails d’implémentations et Tests.