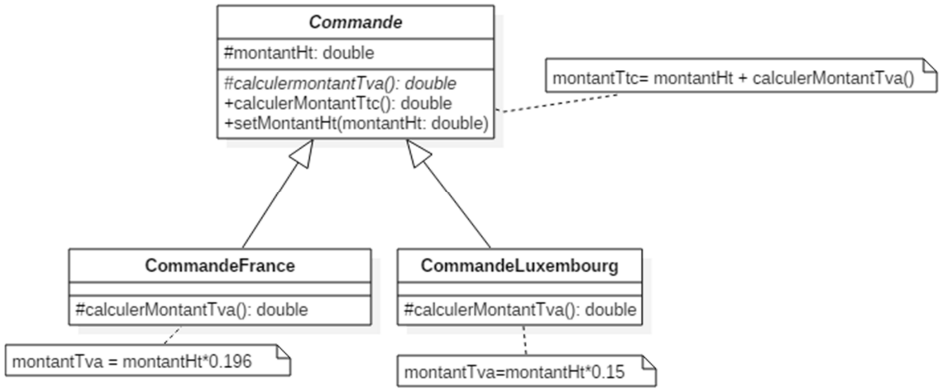
M2104 - TD : Diagramme de classes, Génération de code & Tests (Partie n°1)

Exercice 1 : Une histoire de TVA …

Au sein d'un système de vente en ligne de véhicules, vous allez devoir gérer des commandes issues de clients en France et au Luxembourg.
La différence entre les commandes en France et le Luxembourg concerne notamment le calcul de la TVA : si en France, le taux de TVA est toujours de **19,6 %**, il est variable au Luxembourg (12 % pour la partie des prestations, **15 %** pour le matériel : nous considérons que nos commandes ne contiennent que du matériel).
Le calcul de la TVA demande donc deux opérations de calcul distinctes en fonction du pays.

Chaque commande connait son montant hors taxes et doit, entre autres, être capable de calculer le montant TTC. Le calcul du montant TTC varie d'un pays à l'autre puisque rappelons que le montant TTC est égal au montant hors taxes + au montant de la TVA.

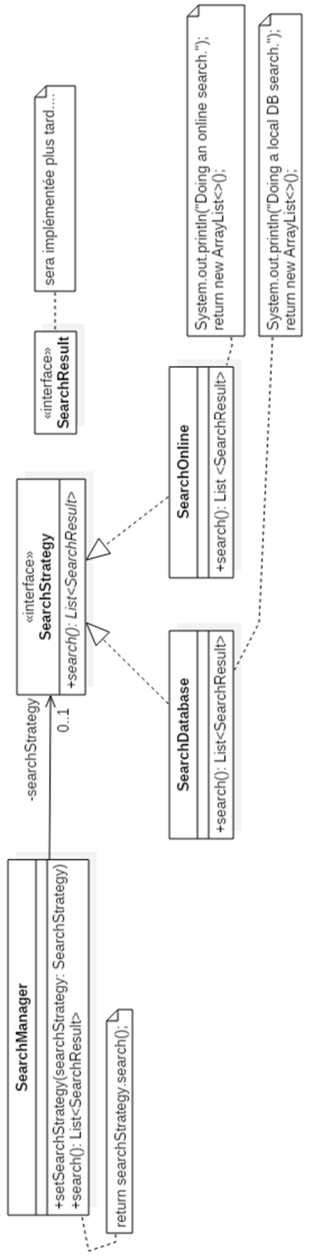
Le diagramme de classes suivant permet de modéliser le contexte précédent en tenant compte de bonnes pratiques de conception.



1. Ecrire, sur une feuille, le code java qui devrait être généré à partir de ce diagramme.

2. Ecrire une classe **`TestCommande`** qui contiendra deux méthodes de test :
   → La première permettra de vérifier qu'une commande passée en France avec un montant Hors taxes donné calcule bien le bon montant TTC.
   → La seconde permettra de vérifier qu'une commande passée au Luxembourg avec un montant donné calcule bien le bon montant TTC.

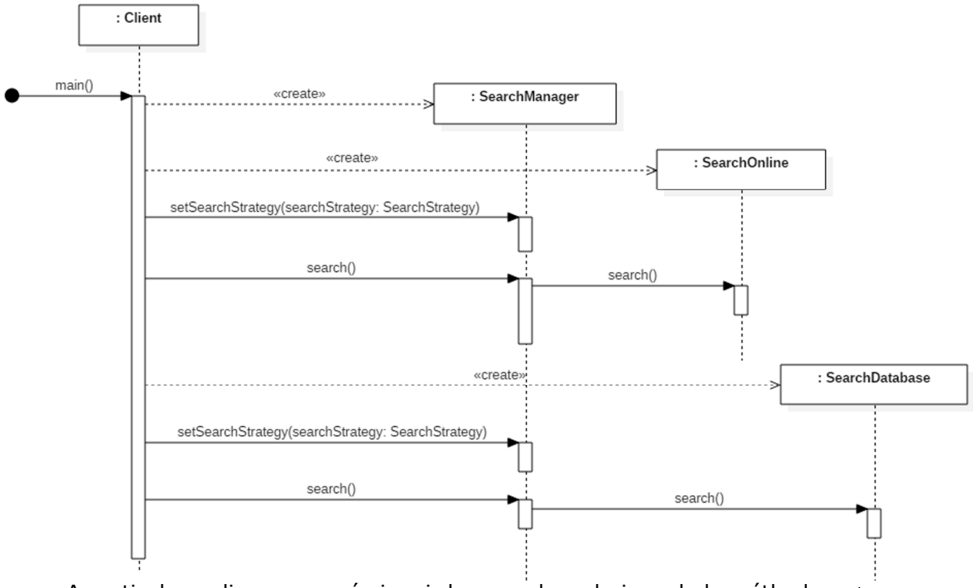
Exercice 2 : Quelle stratégie pour vos recherches ?



1. Notez, sur le diagramme précédent, la relation appropriée (**EST-UN** ou **A-UN** ou **IMPLÉMENTE**) sur chaque flèche du diagramme de classes.

2. Ecrire, sur une feuille, le code java qui devrait être généré à partir de ce diagramme.

3. Le code précédent est testé dans la méthode `main` d'une classe `Client`.
   Le diagramme de séquences suivant modélise les interactions (messages échangés entre les objets) qui apparaissent lors de l'exécution de ce `main`.



A partir de ce diagramme, écrire ci-dessous, le code java de la méthode `main`.

```
public class Client {

    public static void main(String[] args) {









    }
}
```


Exercice 3 : Et si on cancanait un peu …

Veuillez utiliser une feuille séparée pour répondre à cette exercice.
Une fois l'exercice terminé, vous remettrez la feuille à votre enseignant de TD

Vous allez participer au développement d'un jeu de simulation de mare aux canards.
Le jeu doit pouvoir afficher toutes sortes de canards. Les canards peuvent `nager` et `cancaner` (c-a-d émettre des sons).
Chaque sous-type de canard (`Colvert`, `Mandarin`, `Canard en plastique`, `Leurre`, …) est chargé de s'`afficher` correctement, lui seul connait la manière dont il apparaît à l'écran (dans un premier temps l'affichage se limitera à une phrase du style "Je suis un vrai colvert" , "Je suis un vrai mandarin" , …)
Tous les canards peuvent `nager` (et oui, tous les canards flottent, même les leurres!)…
Les canards doivent aussi pouvoir `voler`… mais attention, pas tous : les canards en plastique ne doivent pas voler, cela ferait désordre de voir des canards en plastique voler dans tout l'écran 😉
Les canards n'émettent pas tous le même son : un *vrai* canard (comme le colvert ou le mandarin) émet un cancanement (*can-can*), alors qu'un canard en plastique émet un couienement (*coin-coin*)… ah oui, et un leurre n'émettra aucun son (ce sera un canard muet 😉) .
Et pour finir, saviez-vous que durant l'hiver, le canard perd ses plumes de vol (rémiges) et ne peut donc pas voler pour un certain temps ?
Du coup, ce serait bien de faire en sorte qu'un canard ait un comportement de vol qui lui est propre et qui pourra être modifier dynamiquement au cours du jeu c-a-d n'importe quand durant le cycle de vie de cet objet.

Proposer un diagramme de classes pour modéliser ce système