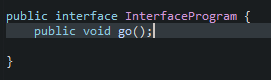
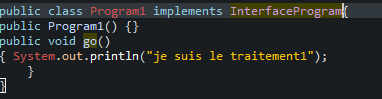
**Corrigé TD1 Design Patterns**

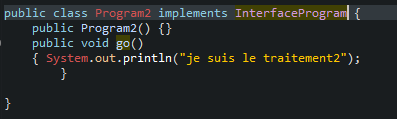
**Factory, AbstractFactory et Singleton**

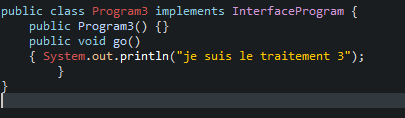
**Exercice n°1:**

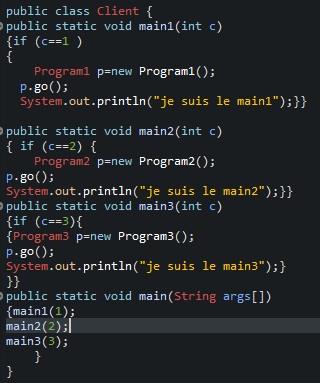
Si on applique pas le pattern Factory le code source de notre projet serait le suivant:

****





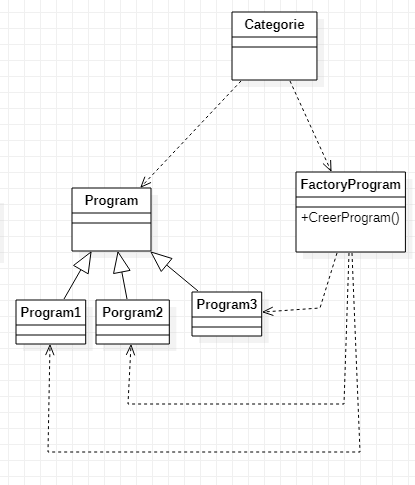




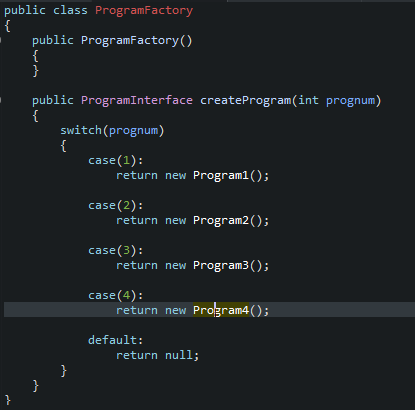
Comme vous le remarquez il ya une duplication du code d'une part et un couplage fort entre le client et l'autre partie du programme. au fait si on tente d'ajouter une autre classe program4 le client devrait être obligé de faire d'autres tests et de remettre en question son code de même s'il envisage de supprimer une classe program => contradiction avec les principes du génie logiciel et des critères de qualité d'un logiciel

**Solution => Application du pattern Factory**

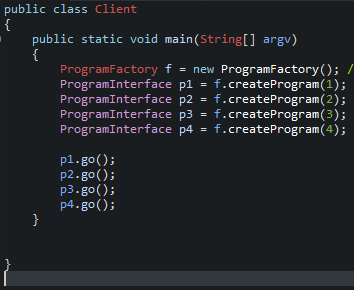
**2) Diagramme de classes:**

****

**un classe ProgramFactory qui se charge de la création des instances**

****

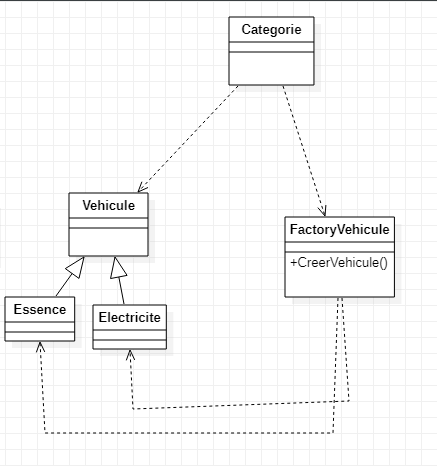
le code de la classe client est le suivant:



Donc c'est seulement la classe ProgramFatory qui va subir des modifications si on envisage de modifier ou de mettre à jour les constructeurs des classes program(i) => le client est à l'abri de toute modification en plus l'extension de l'application est plus facile

**Exercice n°2:**

1) Si on adopte le design pattern Factory, le diagramme de classes de la solution sera le suivant:



2)On envisage d'ajouter une autre famille de produits : Diesel

a) si on adopte la solution Factory dans ce cas là l'ajout d'une nouvelle famille de produits par exemple Diesel ceci va impliquer l'ajout d'une structure conditionnelle dans la classe Factory(si essence elif electrite elif diesel etc) ce qui va compliquer son code.

Cependant si on odopte le pattern AbstractFactory l'ajout de nouvelle famille implique l'ajout d'une sous classe concreteFactory qui hérite de l'interface AbstarctFactory aucune modification de la classe cliente.

b) Diagramme de classes

