INF 4067 - UML ET DESIGN PATTERNS

Devoir 1 - Implémentation des patterns de construction

Author
CHEDJOUN KENGUEP Dave 20U2757
Département Informatique - Master 1 - Genie Logiciel
Octobre 2023

Diagrammes UML et résultats

Il nous a été donné comme devoir d'implémenter les designs patterns plus précisement ceux de structuration. Pour se faire, nous avons pour chaque implémentation, deux versions deux codes ainsi que deux versions de diagrammes. Nous vous les présenterons ci dessous en vous donnant, une brève description, les liens vers les dépots gitHub, des screenshots des résultats et les images avec les modèles UML.

1 Factory Pattern

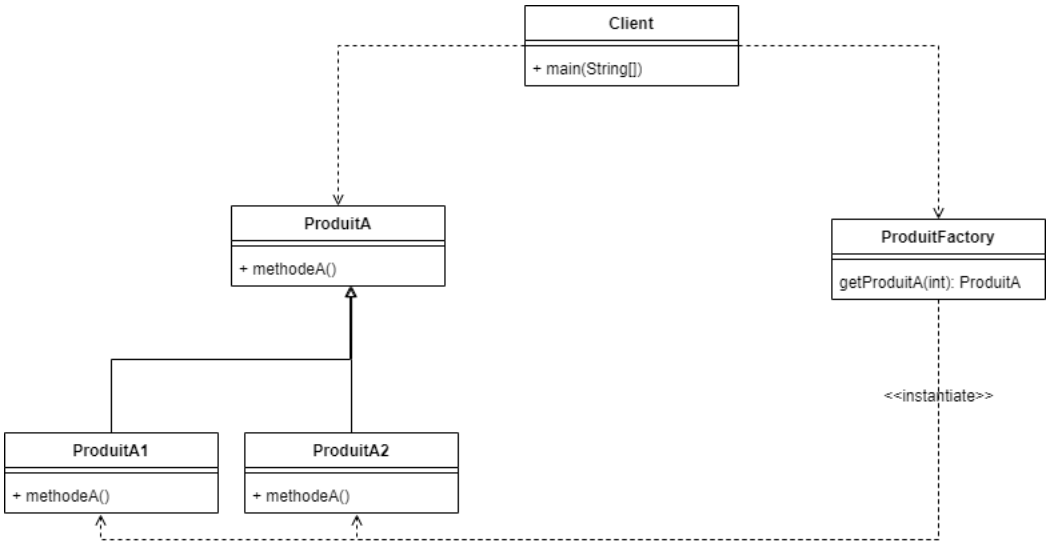
Pour se faire, ici nous avons implementé les deux méthodes présentées en cours à savoir, l'utilisation d'une fabrique abstraite, et d'une methode de fabrique.

1.1 Implementation avec la factory Method

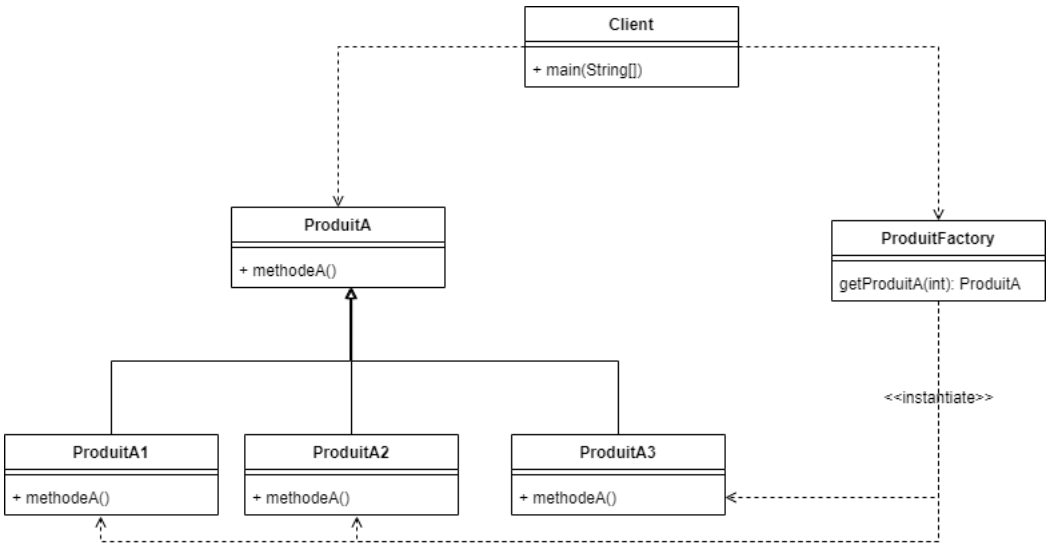
1.1.1 Diagrammes UML

En se basant sur la structure génerale que nous offre cette méthode, et le client pouvant avoir 3 types de produits à fabriquer nous avons obtenu le diagramme suivant:

- Pour l'implémentation avec deux produits A1 et A2 :



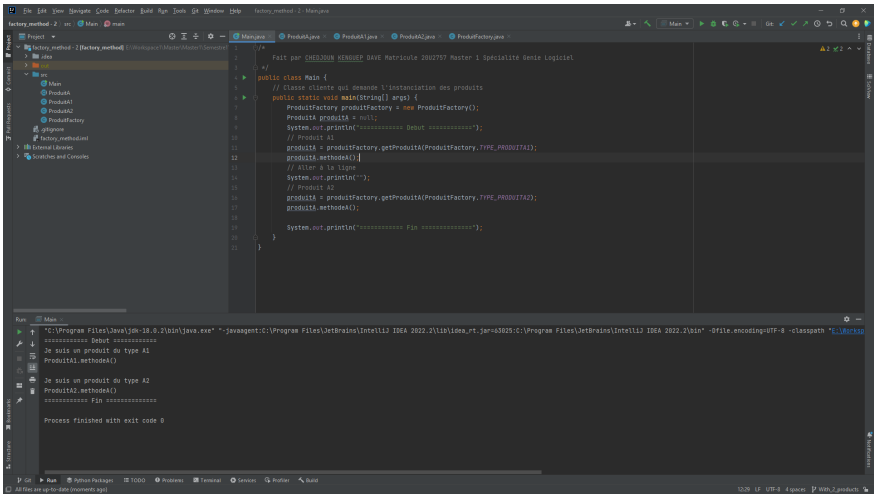
- Pour l'implémentation avec trois produits A1, A2 et A3 :



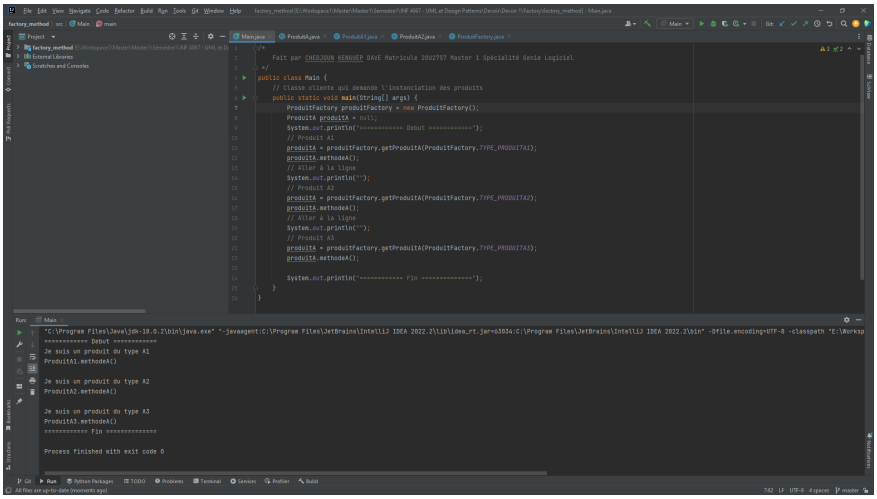
1.1.2 Codes et Résultats

Le code ayant permi l'implémentation se trouve dans le dossier code, avec le nom factory_method. Le code ayant été mis sur gitHub, le lien pour y accéder est le suivant : `https://github.com/DavePhil/factory_method.git`. On retrouvera deux branches: la branche **master** pour l'implementation avec 3 produits et la branche **With_2_products** pour l'implémentation avec 2 produits. Les resultats sont les suivants:

- Pour l'implémentation avec deux produits A1 et A2 :



- Pour l'implémentation avec trois produits A1, A2 et A3 :

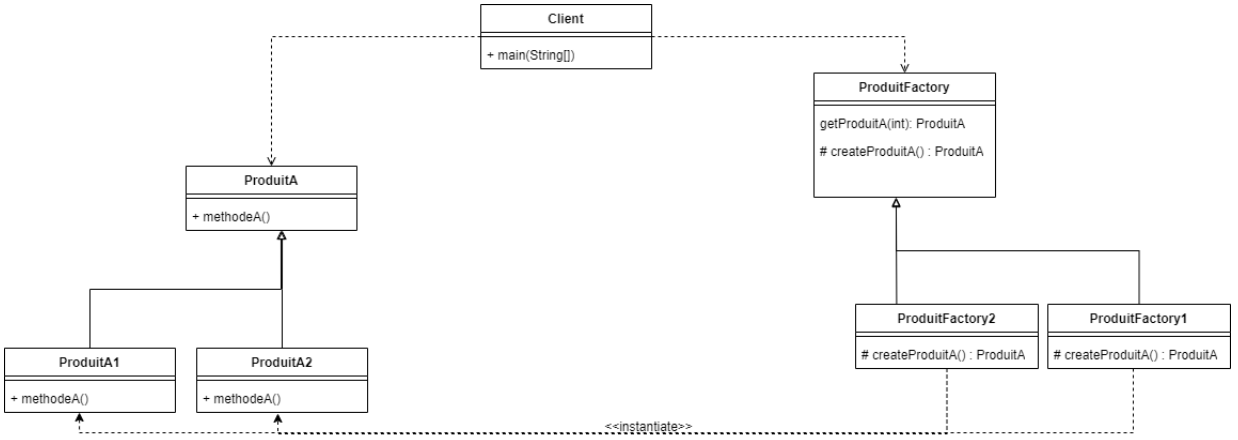


1.2 Implementation avec la factory

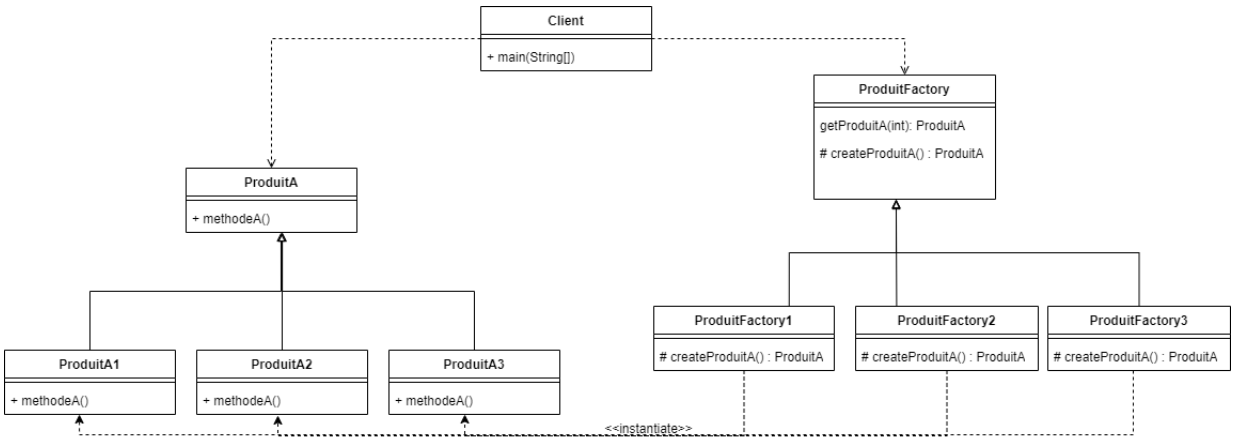
1.2.1 Diagrammes UML

En se basant sur la structure génerale que nous offre la methode factory, et le client pouvant avoir 3 ou 2 types de produits à fabriquer nous avons obtenu le diagramme suivant:

- Pour l'implémentation avec deux produits A1 et A2 :



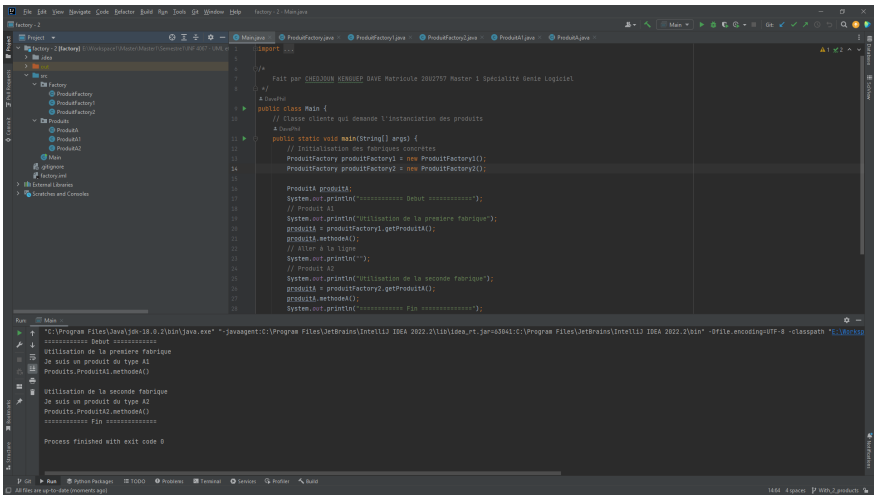
- Pour l'implémentation avec trois produits A1, A2 et A3 :



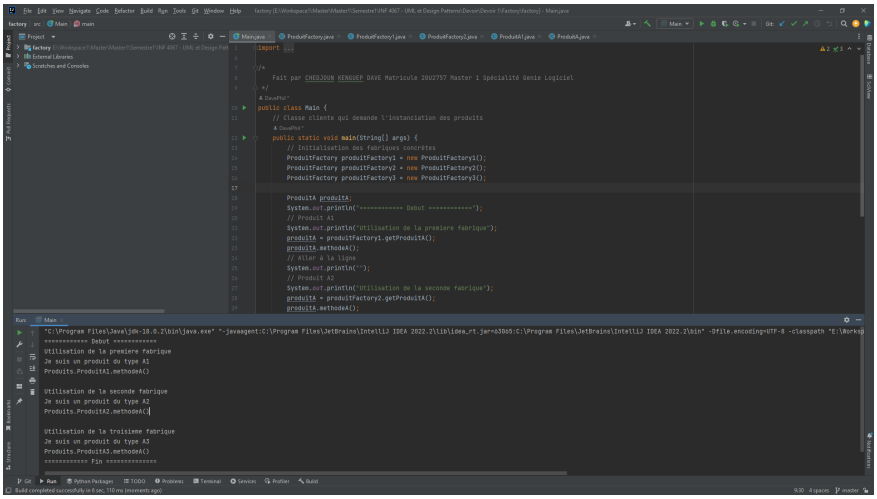
1.2.2 Codes et Résultats

Le code ayant permi l'implémentation se trouve dans le dossier code, avec le nom factory. Le code ayant été mis sur gitHub, le lien pour y accéder est le suivant : `https://github.com/DavePhil/factory.git`. On retrouvera deux branches: la branche **master** pour l'implementation avec 3 produits et la branche **With_2_products** pour l'implémentation avec 2 produits. Les resultats sont les suivants:

- Pour l'implémentation avec deux produits A1 et A2 :



- Pour l'implémentation avec trois produits A1, A2 et A3 :



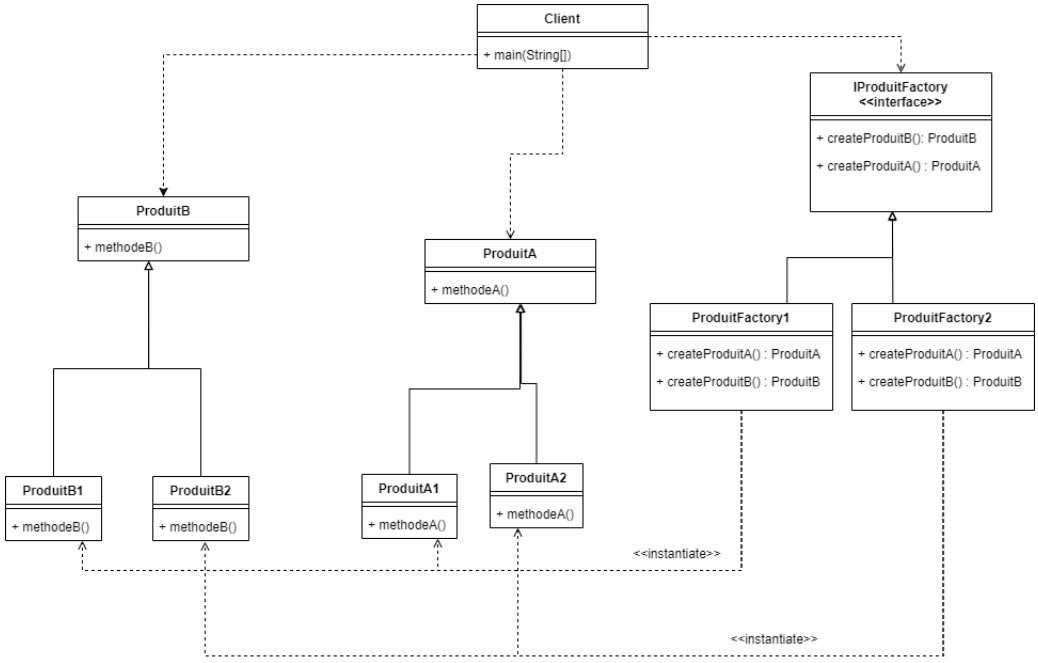
2 Abstract Factory Pattern

2.1 Implementation de l' abstract factory

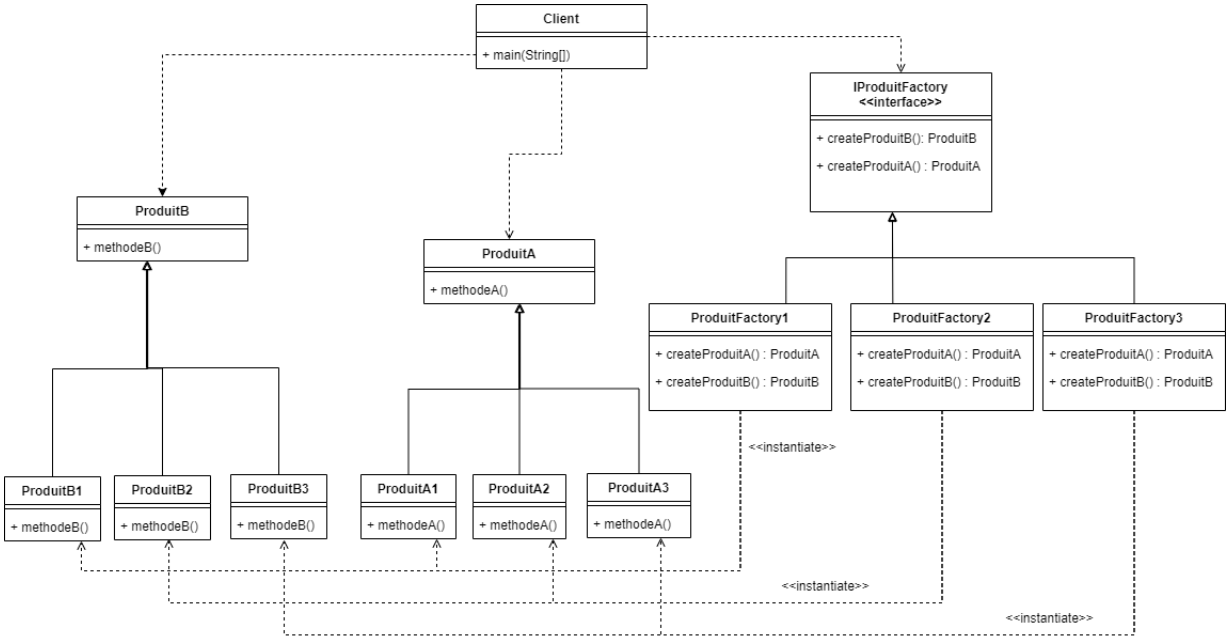
2.1.1 Diagrammes UML

En se basant sur la structure génerale que nous offre la methode, et le client pouvant avoir 3 ou 2 familles de produits à fabriquer nous avons obtenu le diagramme suivant:

- Pour l'implémentation avec deux produits A1 et A2 :



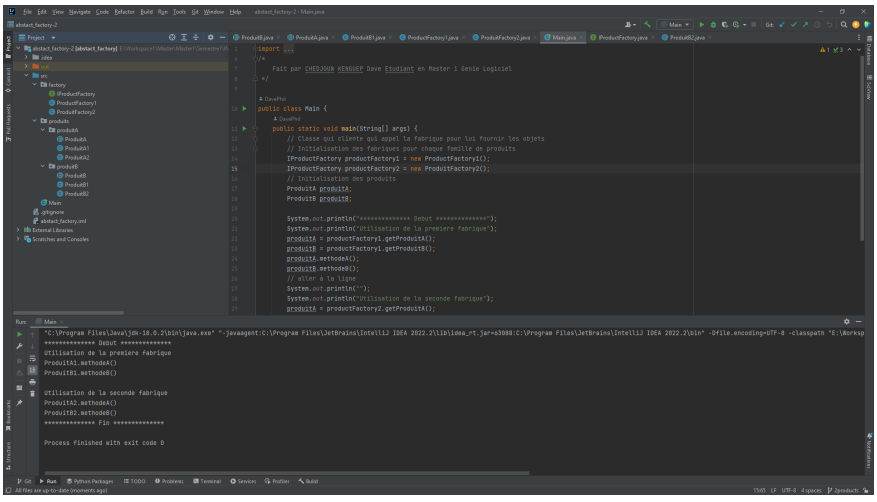
- Pour l'implémentation avec trois produits A1, A2 et A3 :



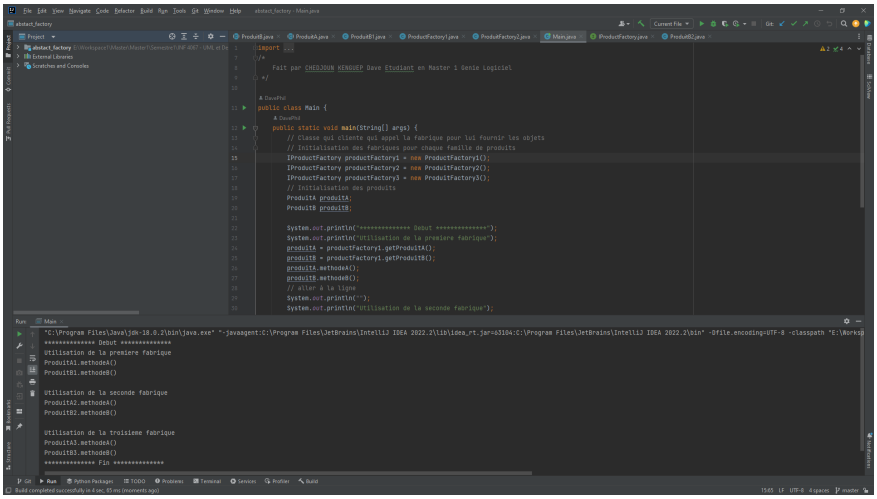
2.1.2 Codes et Résultats

Le code ayant permi l'implémentation se trouve dans le dossier code, avec le nom abstact_factory. Le code ayant été mis sur gitHub, le lien pour y accéder est le suivant : `https://github.com/DavePhil/abstact_factory.git`. On retrouvera deux branches: la branche **master** pour l'implementation avec 3 produits et la branche **2products** pour l'implémentation avec 2 produits. Les resultats sont les suivants:

- Pour l'implémentation avec deux produits A1 et A2 :



- Pour l'implémentation avec trois produits A1, A2 et A3 :



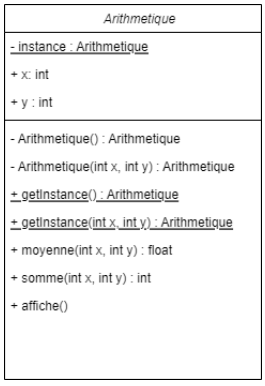
3 Singleton Pattern

3.1 Implementation du singleton

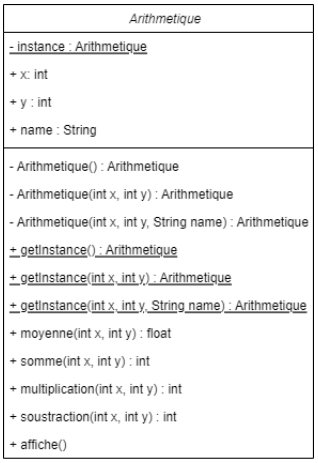
3.1.1 Diagrammes UML

En se basant sur la structure génerale que nous offre la methode, et le client pouvant deux attributs (x et y) ou alors 3 attributs (x, y, name); ainsi que deux constructeurs avec paramètres et sans paramètres et enfin des fonctions de recupération des instances fonction des constructeurs nous avons obtenu les diagrammes suivants:

- Pour l'implémentation avec les attributs x et y :



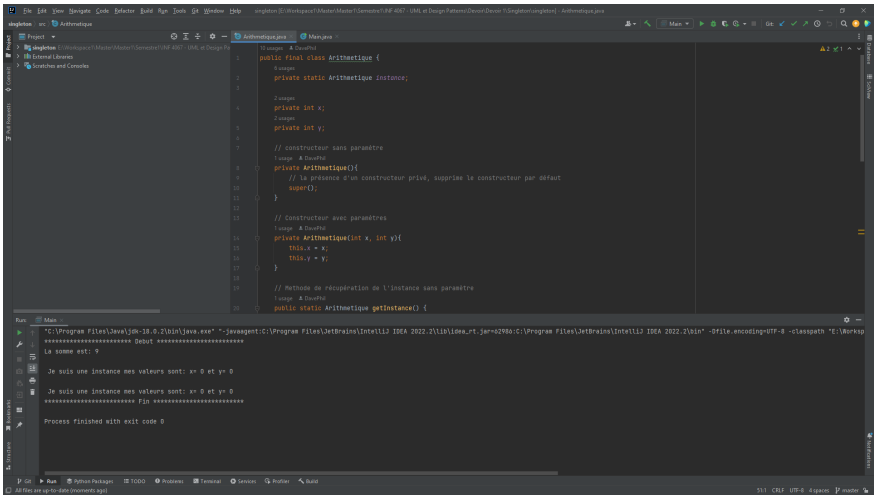
- Pour l'implémentation avec les attributs x, y et name:



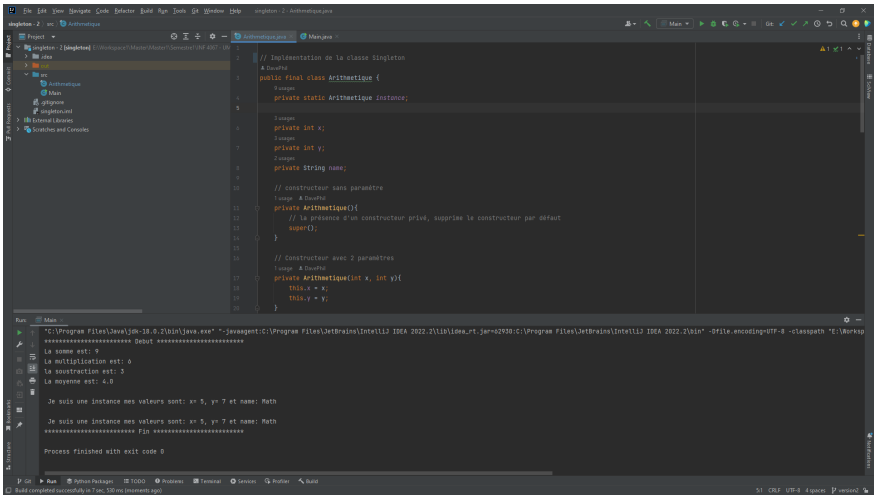
3.1.2 Codes et Résultats

Le code ayant permi l'implémentation se trouve dans le dossier code, avec le nom singleton. Le code ayant été mis sur gitHub, le lien pour y accéder est le suivant : `https://github.com/DavePhil/singleton.git`. On retrouvera deux branches: la branche **master** pour l'implementation avec 2 attributs et la branche **version2** pour l'implémentation avec 3 attributs. Les resultats sont les suivants:

- Pour l'implémentation avec les attributs x et y :



- Pour l'implémentation avec les attributs x, y et name :



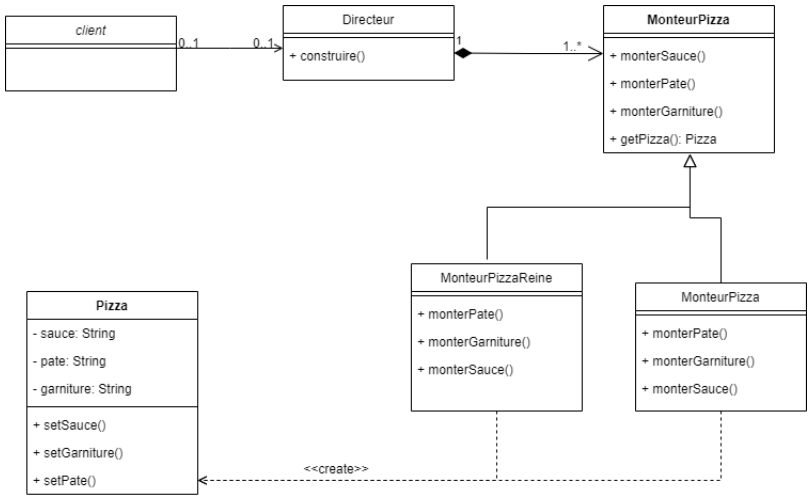
4 Builder Pattern

4.1 Implementation du builder

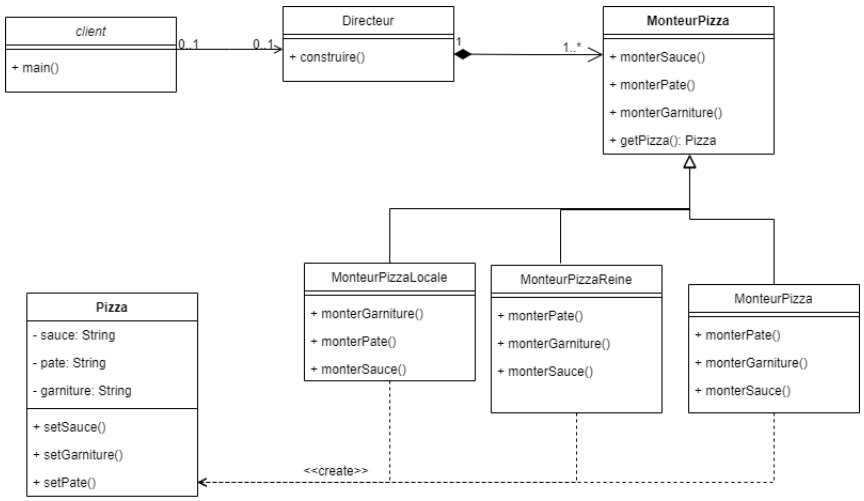
4.1.1 Diagrammes UML

En se basant sur la structure génerale que nous offre la methode, nous avons obtenu le diagramme suivant pour l'exercice des pizza du cours:

- Pour l'implémentation avec deux pizza:



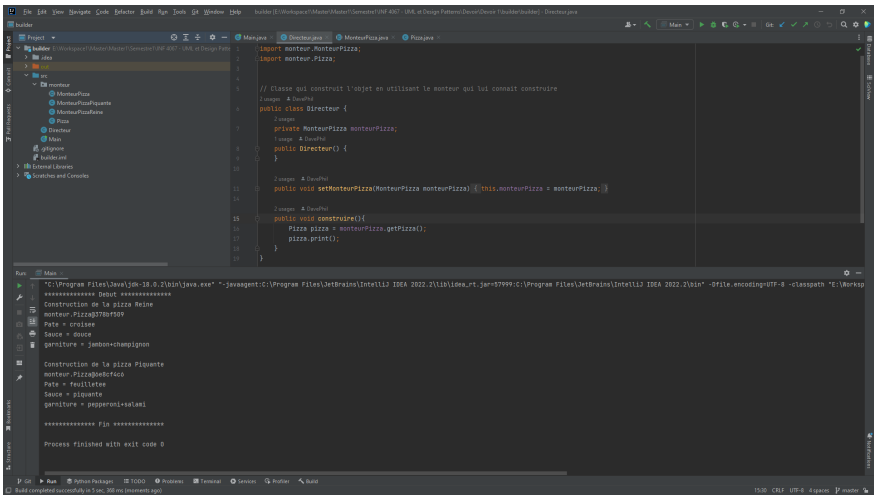
- Pour l'implémentation avec trois pizza:



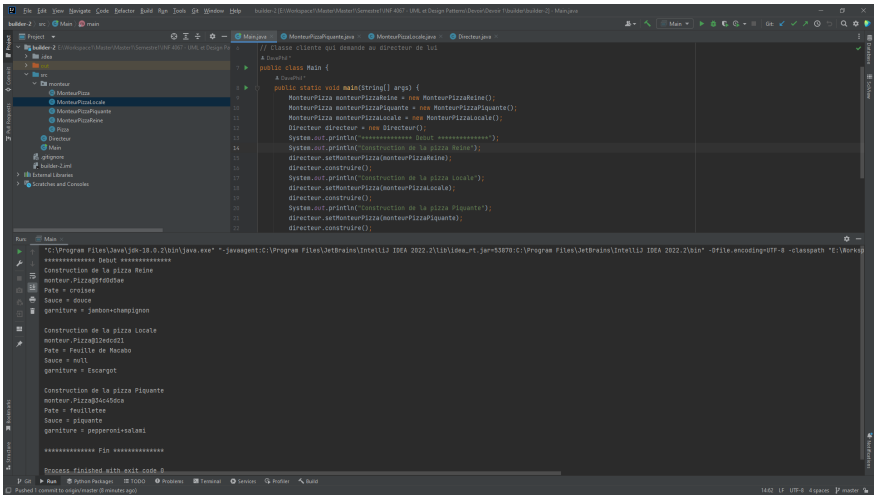
4.1.2 Codes et Résultats

Le code ayant permi l'implémentation se trouve dans le dossier code, avec le nom builder. Le code ayant été mis sur gitHub, le lien pour y accéder est le suivant : `https://github.com/DavePhil/builder.git`. On retrouvera deux branches: la branche **master** pour l'implementation avec 3 produits et la branche **2products** pour l'implémentation avec 2 produits. Les resultats sont les suivants:

- Pour l'implémentation avec deux pizza :



- Pour l'implémentation avec trois pizza :


// Classe cliente qui demande au directeur de loi
public class Main {
public static void main(String[] args) {
MonteurPizza monteurPizzaReine = new MonteurPizzaReine();
MonteurPizza monteurPizzaPiquante = new MonteurPizzaPiquante();
MonteurPizza monteurPizzaLocale = new MonteurPizzaLocale();
Directeur directeur = new Directeur();
System.out.println("************ Debut ************");
System.out.println("Construction de la pizza Reine");
directeur.setMonteurPizza(monteurPizzaReine);
directeur.construire();
System.out.println("Construction de la pizza Locale");
directeur.setMonteurPizza(monteurPizzaLocale);
directeur.construire();
System.out.println("Construction de la pizza Piquante");
directeur.setMonteurPizza(monteurPizzaPiquante);
directeur.construire();
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2\lib\idea_rt.jar=53870:C:\Program Files\JetBrains\IntelliJ IDEA 2022.2\bin" -Dfile.encoding=UTF-8 -classpath "E:\Works
************ Debut ************
Construction de la pizza Reine
monteur.Pizza@5fd0d5ae
Pate = croisee
Sauce = douce
garniture = jambon+champignon
Construction de la pizza Locale
monteur.Pizza@12edcd21
Pate = Feuille de Macabo
Sauce = null
garniture = Escargot
Construction de la pizza Piquante
monteur.Pizza@51cc5dca
Pate = feuilletee
Sauce = piquante
garniture = pepperoni+salami
************ Fin ************
Process finished with exit code 0