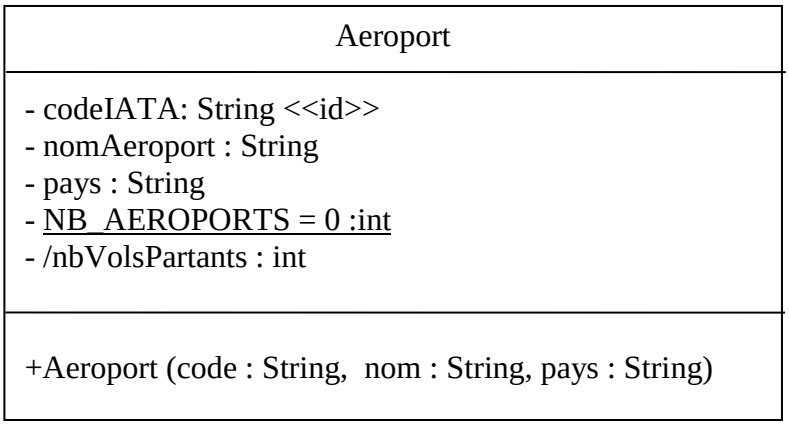
TD5 : Diagramme de classes dans le cadre d'une analyse détaillée

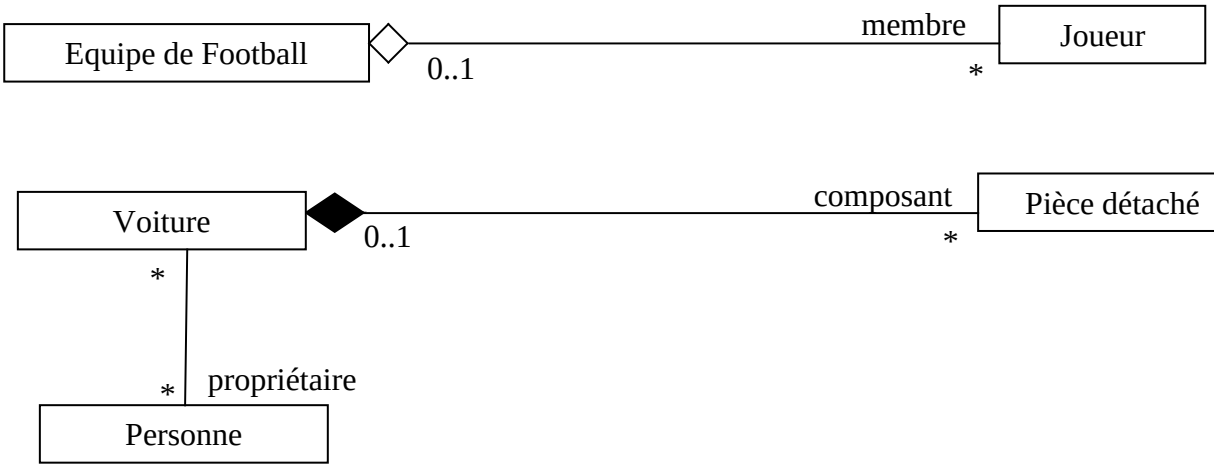
Diagramme de classes – éléments supplémentaires

1. Indiquez sur ce diagramme de classes : les visibilités, les types des attributs, les méthodes avec leur signature, les attributs calculés, les attributs de classe.



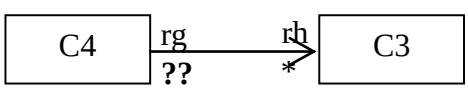
Agrégation et composition

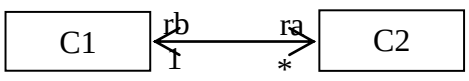
1. Indiquez la différence entre les 3 éléments suivants : une association, une association « agrégation », une association « composition ».
2. Précisez le type d'association dans les diagrammes de classes suivants.

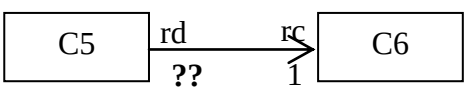


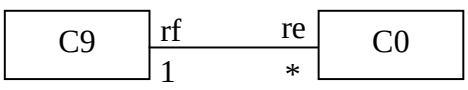
La navigabilité dans le diagramme de classes

1. Expliquez, avec le maximum de détails, les différents diagrammes de Classes : donnez la signification de ra, rb, rc, rd, re et rf.
2. Ecrivez le code des classes (sans le corps des méthodes et des constructeurs).



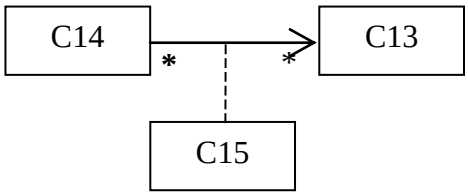


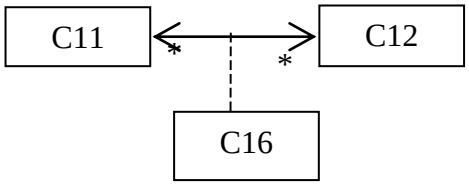




Navigabilité dans le cas des classes associatives

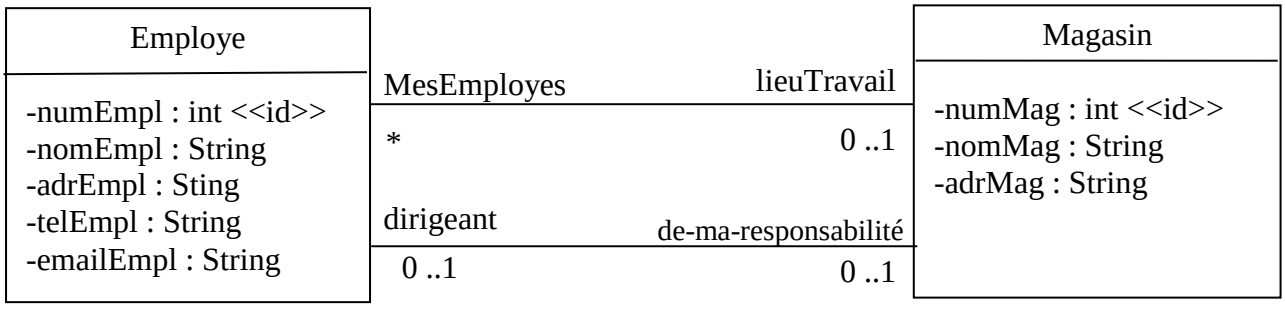
1. Ecrivez le code des classes (sans le corps des méthodes et des constructeurs).



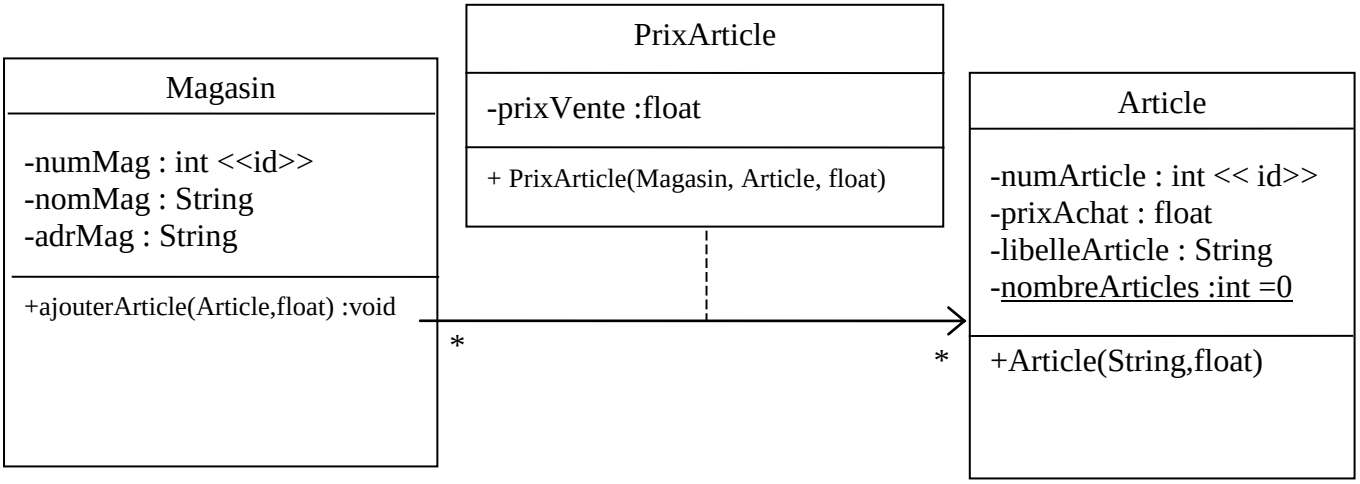


Squelette du code Java

1. Ecrivez le code la classe Employé et celui de la Classe Magasin (sans le corps des méthodes et des constructeurs).



2. Complétez le code de la classe Magasin. Ecrivez le code de la classe Article (sans le corps des méthodes et des constructeurs). Ecrivez le code de la classe « prixArticle».



3. Ecrivez le code du constructeur `Article`.

4. Ecrivez le code du constructeur `PrixArticle (m : Magasin, a : Article, p : float)` et le code de la méthode `ajouterArticle(a : Article, p : float)` de la classe `Magasin`.

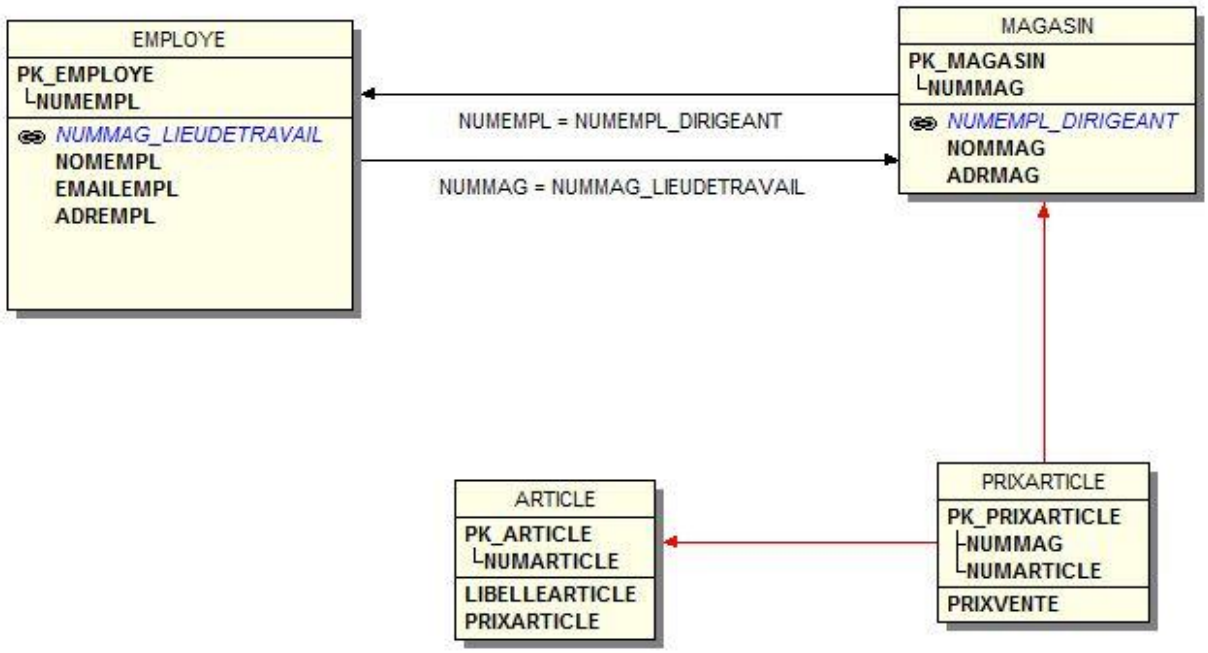
5. Votre code permet-il facilement de trouver le magasin ayant le meilleur prix pour un article donné ? Par exemple : quel est le magasin proposant le meilleur prix pour l'article de numéro 341 ?

6. Modifiez votre code et votre diagramme de classes pour faciliter ce calcul
   a. Ecrivez le code de la méthode `ajouterMagasin()` de la classe `Article`.
   b. Modifier le code du constructeur `PrixArticle(m :Magasin, a : Article, p : float)`.
7. Pour illustrer les inconvénients de cette solution, écrivez le code de la méthode `retirerArticle(art : Article)` de la classe `Magasin`. Cette méthode retire l'article `art` de la vente.
8. Complétez le diagramme de classes avec les méthodes ou constructeurs manquants.

Autre Diagramme

Quel est le nom du diagramme suivant ?

a. A quoi correspond la flèche entre PRIXARTICLE et MAGASIN ?
b. A quoi correspond la flèche entre PRIXARTICLE et ARTICLE ?
c. A quoi correspond la flèche d'étiquette « NUMMAG = NUMMAG_LIEUDETRAVAIL »
d. A quoi correspond la flèche d'étiquette « NUMEMP = NUMEMP_DIRIGEANT »

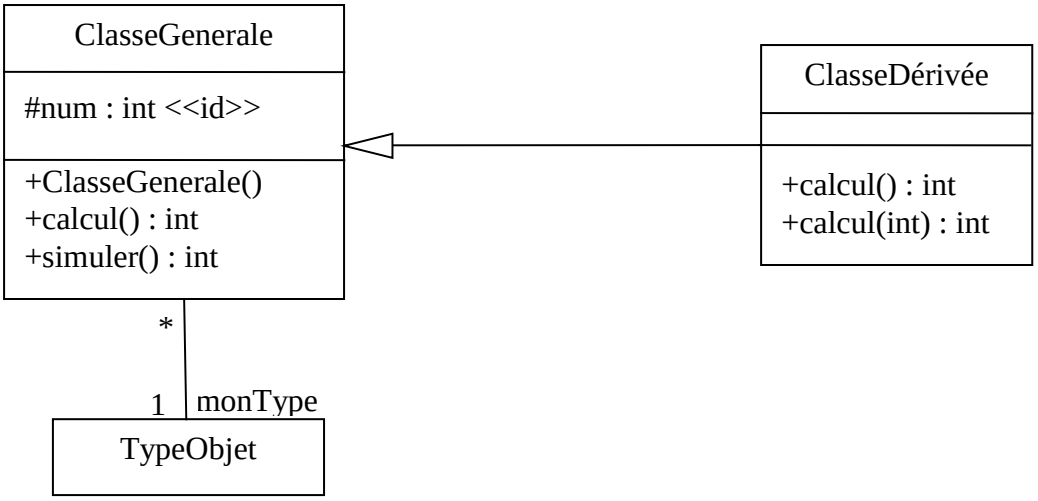


Généralisation/Spécialisation : héritage

La généralisation décrit une relation entre une classe générale (classe de base ou classe parent) et une classe spécialisée (sous-classe, classe dérivée ou classe enfant).
Un objet de la classe spécialisée peut être utilisé partout où un objet de la classe de base est autorisé.
Cette relation de généralisation se traduit par le concept d'héritage. On parle également de relation d'héritage.
Les propriétés principales de l'héritage sont :

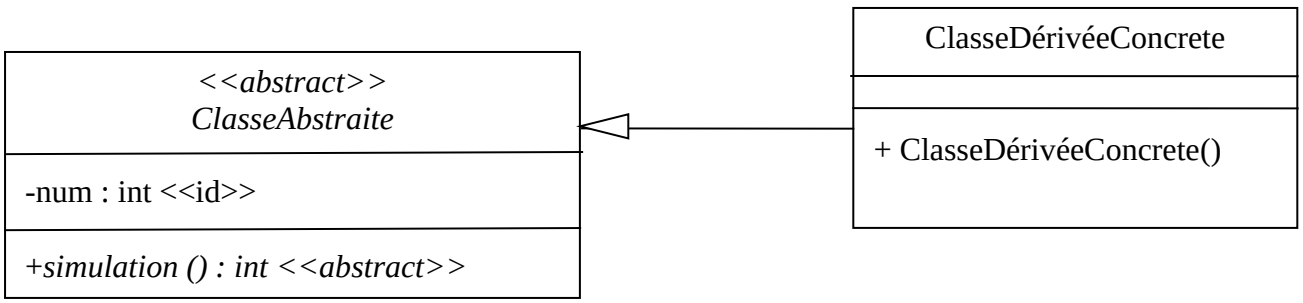
- la classe spécialisée possède toutes les caractéristiques (attributs, associations ou méthodes) de sa classe parent, mais elle ne peut accéder aux caractéristiques privées de cette dernière ;
- toutes les associations de la classe générale s'appliquent aux classes dérivées ;
- un objet d'une classe peut être utilisée partout où une instance de sa classe parent est attendue ;
- *polymorphisme* : une classe enfant peut redéfinir une ou plusieurs méthodes de la classe parent (avec la même signature). Sauf indication contraire, un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes.

La *surcharge* d'opérations (même nom, mais signatures des opérations différentes) est possible dans toutes les classes.
Lors de la conception ne pas confondre la notion « d'héritage » avec la notion être « du type ».



Classe abstraite

Une opération est dite *abstraite* lorsqu'on connaît sa signature mais pas la manière dont elle peut être réalisée.
Une classe est dite *abstraite* lorsqu'elle a au moins une méthode abstraite.
Une classe abstraite ne peut pas être instanciée. Elle peut avoir un constructeur.



Interface dans la notation UML

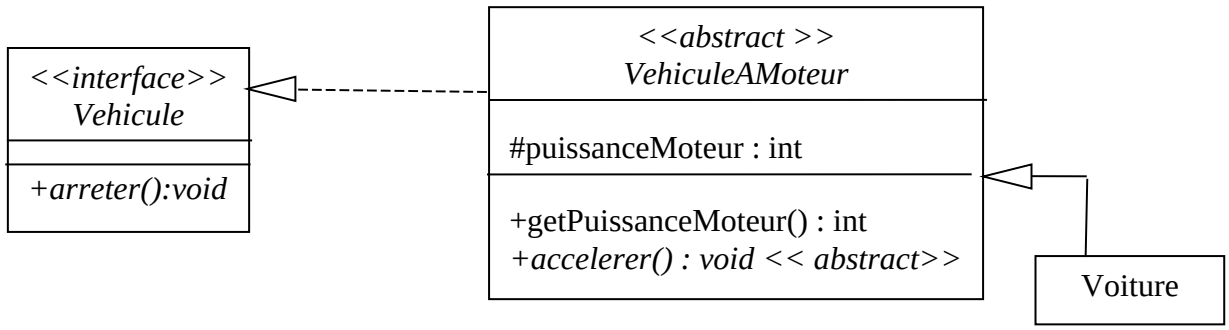
Ref : https://www.ibm.com/support/knowledgecenter/fr/SS5JSH_9.5.0/com.ibm.xtools.modeler.doc/topics/cinterfc.html

Les interfaces masquent et protègent le code en déclarant publiquement une liste de services (méthodes). Les classes réalisent les interfaces en implémentant ce comportement.

Le développement d'applications est simplifié car les développeurs de logiciels qui écrivent le code client ont uniquement besoin de connaître les interfaces, mais pas les détails de l'implémentation.

Si vous remplacez, les classes qui implémentent des interfaces, vous n'avez pas besoin de refaire la conception de l'application client car les nouvelles classes implémentent les mêmes interfaces.

Une interface est une classe contenant uniquement des attributs et méthodes de classes ainsi que des méthodes abstraites.
Une interface ne peut pas être instanciée et n'a pas de constructeur.




```
public interface Vehicule {
      public void arreter () ;
}

public abstract class VehiclueAMoteur implements Vehicule {
     protected int puissanceMoteur;
     abstract public void accelerer();
     public int getpuissanceMoteur() {
           return puissanceMoteur;
    }
}

public class Voiture extends VehiculeAMoteur {
  public void accelerer() {}
  public void arreter() {}

}
```

1. **Question.** Ecrivez le code Java correspondant au diagramme de classes C3478

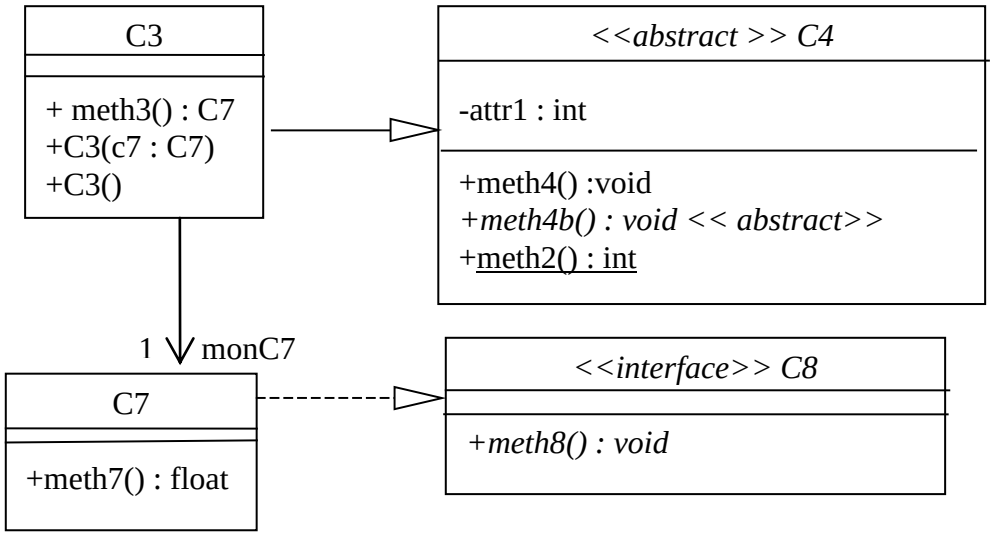


Diagramme de Classes C3478

2. **Question.** Commentez les lignes de code en Java suivante en fonction du diagramme de Classes C3478. Ce code a été réalisé par un programmeur débutant qui peut avoir fait des erreurs.

```
public class MainC3478 {
  public static void main(String[] args) {
        C8 o8 ;
        o8 = new C8() ;
        o8 = new C7();
        System.out.println(C4.meth2());
        C4 o4 ;
        o4 = new C4() ;
        o4 = new C3() ;
        o4.meth4b();
        o4.meth3();
        C7 o7 ;
        o7 = new C7() ;
        o7.meth8() ;
        System.out.println(o7.meth7());
        C3  o3 = new C3(c7) ;
        o3.meth4b();
        o3.meth4() ;
        C3.meth3() ;
        o3.meth3().meth8() ;
        System.out.println(o3.meth3().meth7());
  }}
```