Annexe : Notation UML

Ce document rappelle les correspondances entre la notation UML et le code Java.

Classe

Une classe est représentée par un cadre, muni de trois *compartiments* : le *nom* de la classe, les *attributs* de la classe et les *opérations* (ou méthodes en terminologie Java).

Chaque attribut ou opération est précédé de sa visibilité, via un caractère :

- \+ `public`
- − `private`
- # `protected`
- ∼ `package` (c'est la visibilité par défaut en Java si aucun mot-clé n'est utilisé ; notez que le mot-clé `package` est réservé à indiquer le paquetage d'une classe et n'est pas utilisable avant un attribut ou opération pour spécifier sa visibilité)

Le type de chaque attribut est noté après un « : » (`nomAttribut : typeAttribut`).
Le mot-clé `static` se traduit en soulignant l'attribut ou opération concerné.

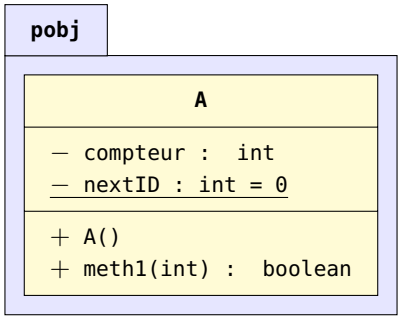

Figure 1: Diagramme UML de la classe `A`.

A.java

```java
package pobj;
public class A {
   private int compteur;
   private static int nextID = 0;
   public A() { compteur = nextID++; }
   public boolean meth1(int valeur) { return compteur == valeur; }
}
```

Interface

L'interface se représente comme une classe, mais n'ayant en général que le compartiment méthodes. Elle se distingue graphiquement des classes en ajoutant le mot-clé `interface` entre chevrons « ».

```java
package pobj;
public interface IA {
   public boolean meth1(int valeur);
}
```

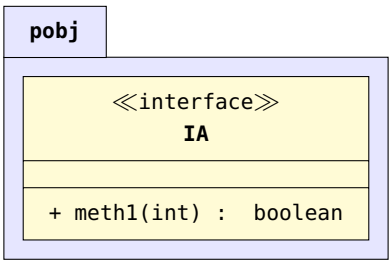

Figure 2: Diagramme UML de l'interface `IA`.

Héritage, implémentation

L'héritage se représente à l'aide d'une flèche munie d'une large tête triangulaire blanche.

Le mot-clé `extends` se traduit par une flèche pleine, tandis que le mot-clé `implements` se représente par une flèche en pointillé. En règle générale, on ne représente pas la relation d'héritage avec la classe `Object`.

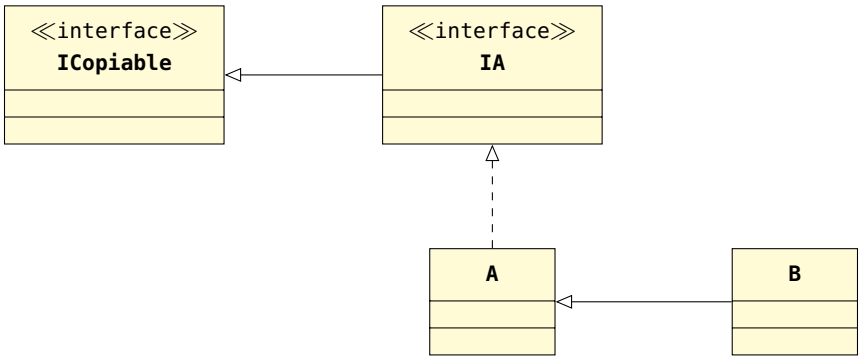

Figure 3: Diagramme UML des interfaces `ICopiable`, `IA` et des classes `A` et `B`.

```
public interface ICopiable {}
public interface IA extends ICopiable {}
public class A implements IA {}
public class B extends A {}
```

Attributs, associations

Les attributs peuvent être représentés dans le compartiment du haut de la classe. Cependant, pour mettre en valeur les liens entre classes, on peut les représenter alternativement par des associations, symbolisées par des flèches pleines munies d'une tête de flèche ordinaire $\rightarrow$.

Les associations ont un nom et une visibilité représentés près de la tête de flèche. Elles ont aussi une multiplicité :

- 1 désigne un attribut simple, par exemple chaque chien a un seul maître ;
- $*$ désigne un attribut énumérable (ou multivalué en terminologie UML), typiquement une liste ou un tableau.

Dans l'UE on se limitera à ces deux multiplicités. Les attributs de multiplicité $*$ seront implémentés de préférence par une `List` de `java.util`.

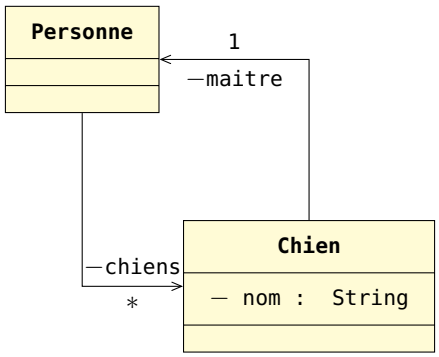

Figure 4: Diagramme UML des classes `Chien` et `Personne`.

Chien.java

```java
public class Chien {
   private String nom;
   private Personne maitre;
}
```

Personne.java

```java
import java.util.List;
public class Personne {
   private List<Chien> chiens;
}
```

Attention : Il ne faut pas représenter les attributs en double : soit on les représente dans le *compartiment attributs* de la classe (par exemple : `- chiens : Chien [*]`) soit on les représente comme des *associations* (flèches →, comme dans la figure 4) mais pas les deux sur le même diagramme. On préférera les associations entre les classes que vous avez développées (comme `Chien`), mais on représentera plutôt comme des attributs les liens sur les classes de l'API Java (comme `String`).