ANA2 - Lucky Summary

Sm!le42

11 mai 2021

Table des matières

1 Diagramme d'activité

1.1 Transitions automatiques

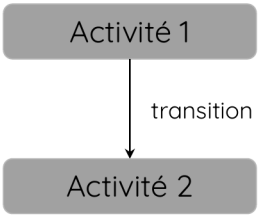

FIGURE 1 – Transitions automatiques

Une fois l'activité 1 terminée, on passe automatiquement à l'activité 2.

1.2 Début et Fin

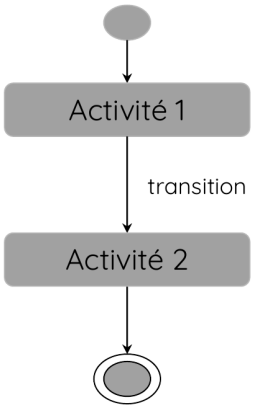

FIGURE 2 – Début et Fin

Le *rond gris* représente le **début** du programme, et le *rond gris entouré d'un cercle* représente la **fin** du programme (fin normale ou fin après erreur).

1.3 Conditions

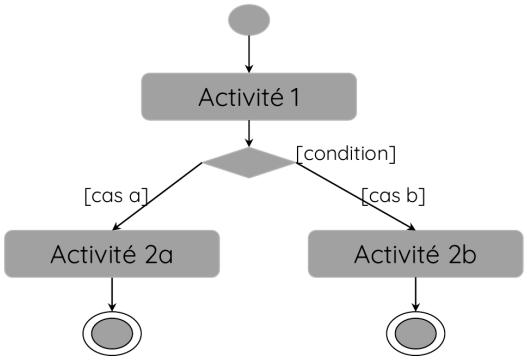

FIGURE 3 – Conditions

Arrivé au *losange (=condition)*, il y a **plusieurs possibilités**, ici soit **cas a**, soit **cas b**. (If, Then, Else)

1.4 Itérations (boucles)

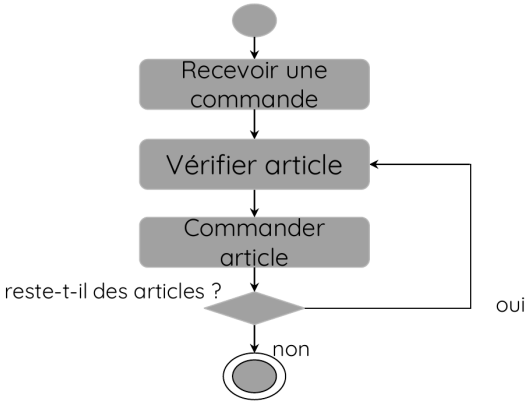

FIGURE 4 – Itérations

Les *instructions* seront exécutées **en boucle** tant que la *condition (losange)* est **vraie**. (For, While)

2 Classes et objets

2.1 Classes

```java
//Exemple de classe Compte en Java
public class Compte {
   //solde > decouvertMax
   private int numero;
   private int solde;
   private int decouvertMax;

   public Compte(int numero, int solde) {
      //...
   }

   public int consulterSolde() {
      //...
      return int;
   }

   public void crediter(int somme) {
      //...
   }

   public void debiter(int somme) {
      //... (solde > decouvertMax)
   }
}
```

Représentation UML de la classe Compte :

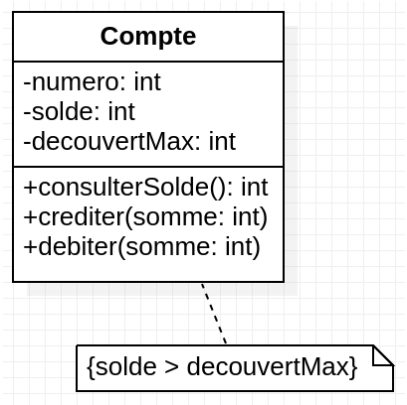

FIGURE 5 – Classes

2.1.1 Visibilités

Il existe 4 types de visibilité :

+	public
–	private
~	package
#	protected

1. **public +**
   La visibilité *public* signifie que les données sont accessibles depuis n'importe où.
2. **private –**
   La visibilité *private* signifie que les données ne sont accessibles que depuis la même classe.
3. **package ~**
   La visibilité *package* signifie que les données ne sont accessibles que depuis le même package.

4. **protected #**
   La visibilité *protected* signifie que les données ne sont accessibles que depuis le même package ou les sous-classes.

2.1.2 Attributs

Les attributs d'une classe sont notés de la manière suivante :

```
Visibilite Type Nom;
```

Ex :

```
private int numero;
```

2.1.3 Opérations

Les opérations d'une classe sont notées de la manière suivante :

```
Visibilite Type de retour Nom (Parametres) {...}
```

Ex :

```
public String getNom(int numero) {...}
```

(Cette méthode prend donc un int en paramètre, et retournera un String)

2.2 Objets

Exemple d'objets de la classe **Compte** (vue plus haut) :

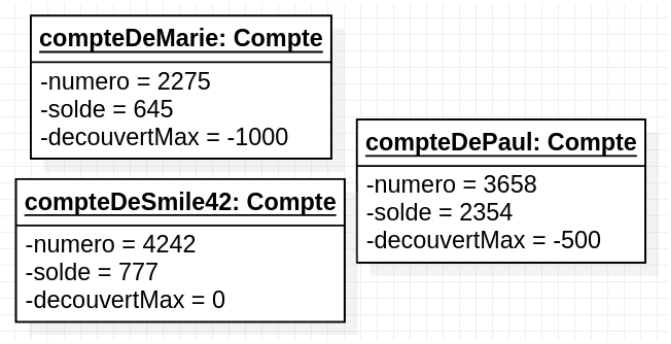

FIGURE 6 – Objets

1. **Identité :**
   Permet d'identifier un objet.
2. **État :**
   Caractéristiques de l'objet à un moment donné (Valeurs des attributs).
3. **Comportement :**
   Ensemble des opérations qu'un objet peut exécuter ou subir.
   — Create
   — Read
   — Update
   — Delete

3 Associations

Connexions sémantiques durables entre des classes.

3.1 Associations 1-1 et 1-N

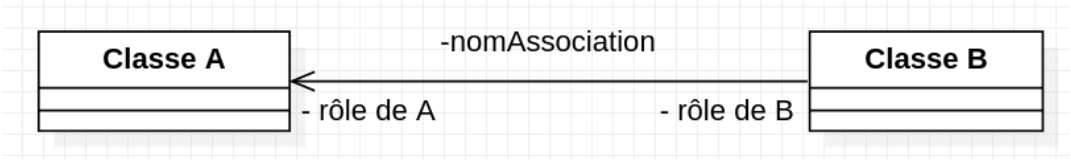

FIGURE 7 – Associations 1-1 1-N

1. **Exemple "banque" :**
   Principe de banque avec des clients et des comptes *(Java + UML)* :

```java
//Classe Client
public class Client {
   private final String nom;
   private final String prenom;
   private final Compte compte;

   public Client(String nom, String prenom, Compte compte) {...}
   public String getNom() {...}
   public String getPrenom() {...}
   public Compte getCompte() {...}
}
```

```java
//Classe Compte
public class Compte {
   private final String numero;
   private int solde;
   private final Client titulaire;

   public Compte(String numero, int solde, Client titulaire) {...}
   public String getNumero() {...}
   public int getSolde() {...}
   public Client getTitulaire() {...}
}
```

   Représentation UML de l'association des classes Client et Compte :

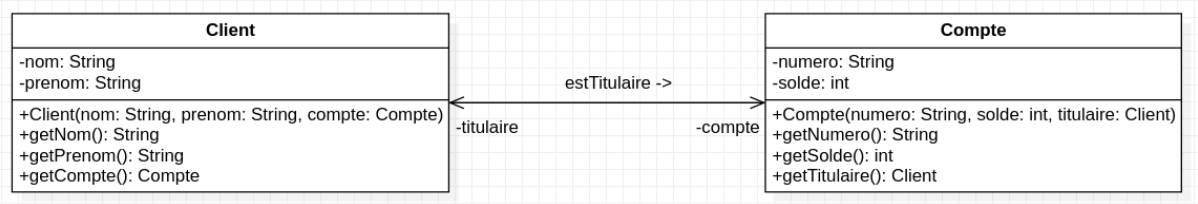

FIGURE 8 – Associations Client Compte (Exemple)

   Ainsi, si on créé un Client "pierre" et qu'on lui attribue le compte "c1", on peut dire ceci :

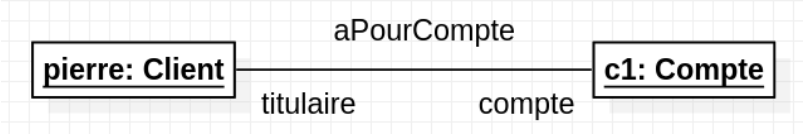

FIGURE 9 – Pierre : c1

   — pierre a pour compte c1
   — c1 joue le rôle de compte pour pierre
   — pierre joue le rôle de titulaire pour c1
   — le *(un des)* compte *(s)* de pierre est c1
   — le *(un des)* titulaire *(s)* de c1 est pierre

2. **Exemple "memory" :**
   Petit jeu de mémoire nommé "memory" *(Java + UML)* :

```java
//Classe Memory
public class Memory {
   private final Game game;
}
//Classe Card
```

```java
public class Card {
   private boolean hidden;
   private int value;
}
//Classe Game
public class Game {
   private Card cards; //Ici une seule carte possible!
}
```

Représentation UML du jeu "memory" :

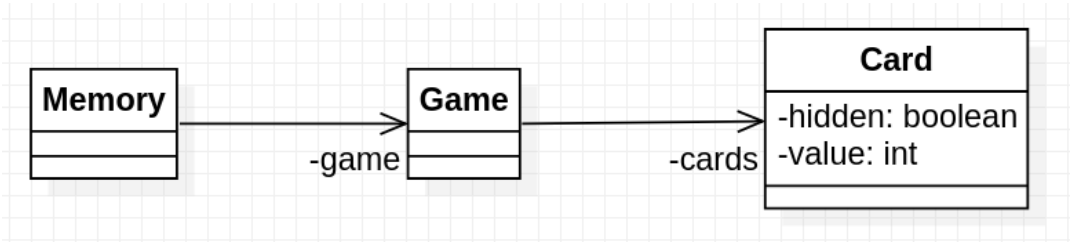

FIGURE 10 – Associations "memory" (Exemple)

3.2 Associations N-N

3.2.1 Multiplicités

Désigne le **nombre d'objets** qui peuvent participer à une **association**.

Syntaxe : {minimum} .. {maximum}

Les plus courants :
- **0..1** (= 0 ou 1)
- **1..1** ou **1** (= 1)
- **0..n** ou **n** ou **0..*** ou ***** (= tout)
- **1..n** ou **1..*** (= 1 ou plus)

Autres :
- **7..14** (= de 7 à 14)
- **0..21** (= de 0 à 21)
- **42..*** (= 42 ou plus)

Remarques :
- ***** = **0..*** (= tout)
- **3** = **3..3** (= 3)

1. **Exemple "memory" :**
   Reprenons l'exemple "memory (vu plus haut) :

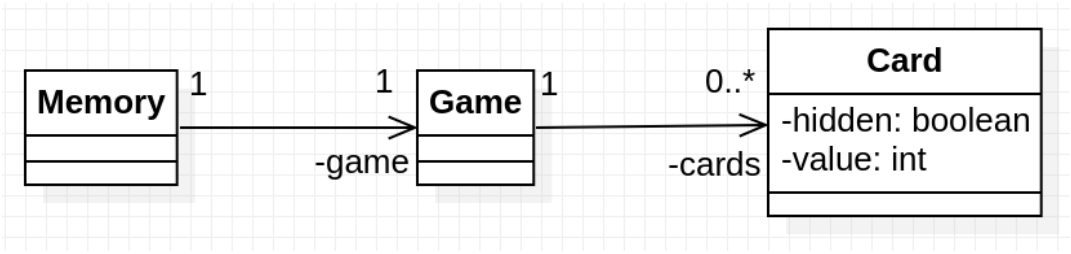

FIGURE 11 – Associations "memory" multiplicités (Exemple)

   À gauche :
   - À un objet Memory correspond un et un seul objet Game
   - À un objet Game correspond un et un seul objet Memory

   À droite :
   - À un objet Card correspond un et un seul objet Game
   - À un objet Game correspond 0 à plusieurs objets Card

2. **Exemple "banque" :**
   Prenons un exemple similaire à l'exemple "banque" (vu plus haut) :

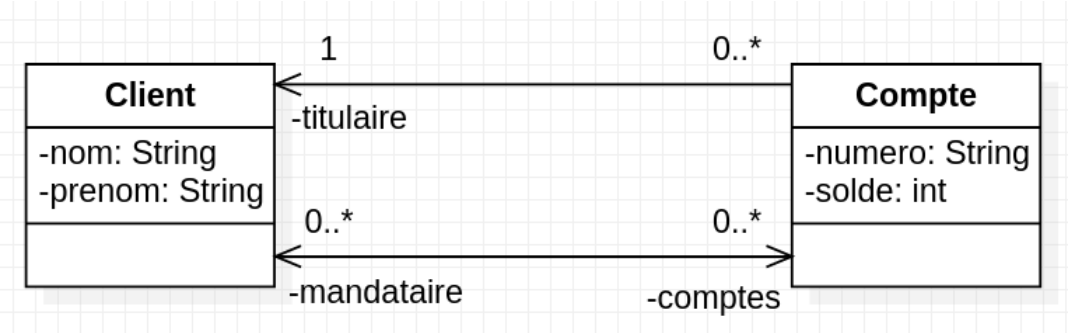

FIGURE 12 – Associations Client/Comptes multiplicités (Exemple)

On en déduit que :
— Un compte a un et un seul titulaire
— Un client peut n'être titulaire d'aucun compte, de un, ou plusieurs comptes
— Un compte peut n'avoir aucun mandataire, ou un, ou plusieurs
— Un client peut n'être mandataire d'aucun compte, de un, ou de plusieurs comptes

3. **Exemple "université" :**
Une université veut informatiser son administration pour gérer plus facilement les dossiers **étudiants**. Les étudiants ont un *identifiant* et un *nom*. Un étudiant fait partie d'un **département**. Ces derniers sont regroupés dans des **facultés**.
Voici un diagramme UML représentant cette structure :

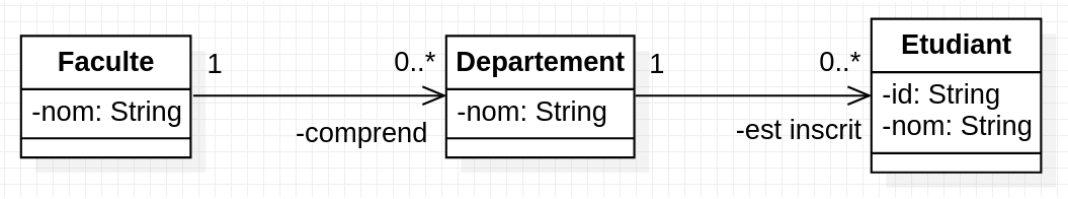

FIGURE 13 – Associations "université" (Exemple)

Et son code Java ressemblerait à ceci :

```java
//Classe Faculte
public class Faculte {
   private String nom;
   private List<Departement> departements;
   //...
}
//Classe Departement
public class Departement {
   private String nom;
   private List<Etudiant> etudiants;
   //...
}
//Classe Etudiant
public class Etudiant {
   private String id;
   private String nom;
   //...
}
```

4 Énumérations, structures et compositions

4.1 Énumérations

Une énumération n'a **jamais** d'association.

1. **Exemple "saison" :**
Code Java :

```java
//Exemple Saison
public enum Saison {
   ETE, AUTOMNE, HIVER, PRINTEMPS;
}
```

Diagramme UML :

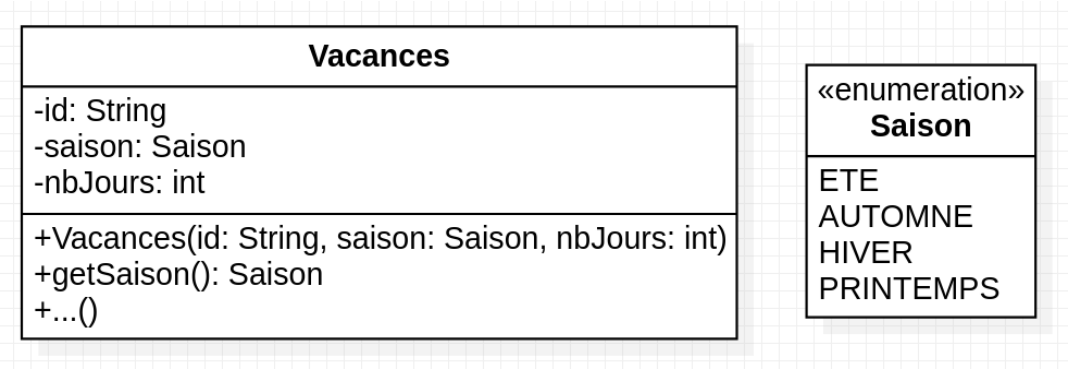

FIGURE 14 – Énumérations de saisons (Exemple)

2. **Exemple "saison2" :**
   Classe saison2 *(Java + UML)* :

```java
//Exemple Saison2
public enum Saison2 {
   PRINTEMPS (21, 3), ETE (22, 6), AUTOMNE (23, 9), HIVER (21, 12);

   private final LocalDate dateDebut;

   private Saison2(int jour, int mois) {
      LocalDate now = LocalDate.now();
      this.dateDebut = LocalDate.of(now.getYear(), mois, jour);
   }

   public LocalDate getDateDebut() {
      return this.dateDebut;
   }

   public Saison2 next() {
      Saison2[] saisons2 = this.values();
      return saisons2[(this.ordinal() + 1) % saisons2.length];
   }
}
```

Code UML :

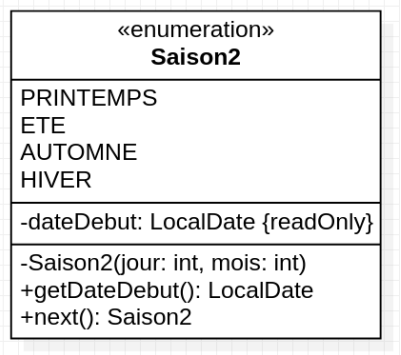

FIGURE 15 – Saison2

4.2 Structures

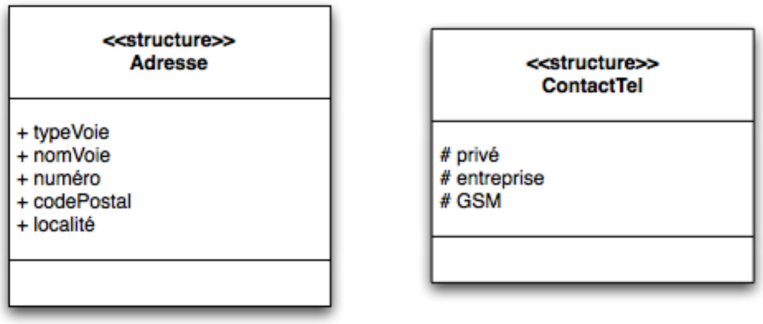

FIGURE 16 – Structures (Exemple)

Code Java de la structure *Adresse* :

```java
//Classe Adresse
public class Adresse {
   public TypeVoie typeVoie;
   public String nomVoie;
   public int numero;
   public int codePostal;
   public String localite;
}
//Enumeration TypeVoie
public enum TypeVoie {
   AVENUE, RUE, CHAUSSEE, BOULEVARD;
}
```

4.3 Compositions

"Est composé de…", "fait partie de…"

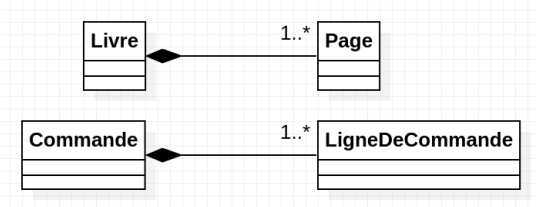

FIGURE 17 – Compositions (Exemples)

5 Classes d'association

Élément de modélisation *UML* qui a les propriétés d'une **classe** et d'une **association**.

1. **Exemple "emplois" :**
   On donne des emplois dans des sociétés à des employés, et on veut que les employés puissent avoir **deux emplois maximum**, mais **pas dans la même société** :

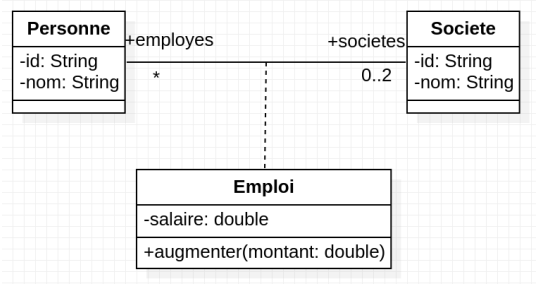

FIGURE 18 – Classes d'association (1 seul emploi par société)

Ceci correspond à cette représentation *UML* :

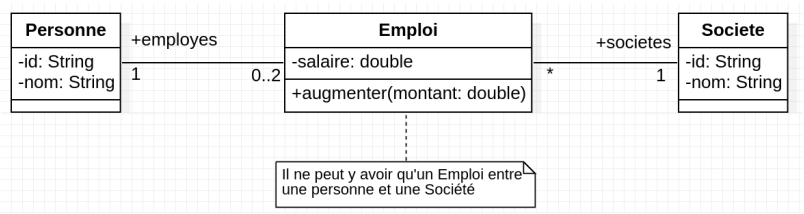

FIGURE 19 – Classes d'association 2 (1 seul emploi par société)

Le code Java de la classe Personne ressemblerait alors à ceci :

```java
//Classe Personne
public class Personne {
   private String id;
   private String nom;
   private Emploi[] emplois; //Liste des emplois actifs
   private int nbEmplois; //Nombre d'emplois actifs
   //Constructeurs, getter, setters...

   public void ajouterEmploi(Emploi e) {
      if (nbEmplois == 2) {
         throw new IllegalArgumentException("Deux emplois max!");
      }
      if (estEmployéDans(e.getSociete()) {
         throw new IllegalArgumentException("Un seul emploi par société!");
      } else {
         emplois[nbEmplois] = e;
         nbEmplois++;
      }
   }

   public boolean estEmployéDans(Societe s) {
      for (Emploi e : emplois) {
         if (e.getSociete().equals(s)) {
            return true;
         }
      }
      return false;
   }
}
```

5.1 Associations N-aire

Associations ternaires, quaternaires, etc...

1. **Exemple "cours" :**
   Diagramme *UML* des liens entre un professeur, un local, une matière et des étudiants :

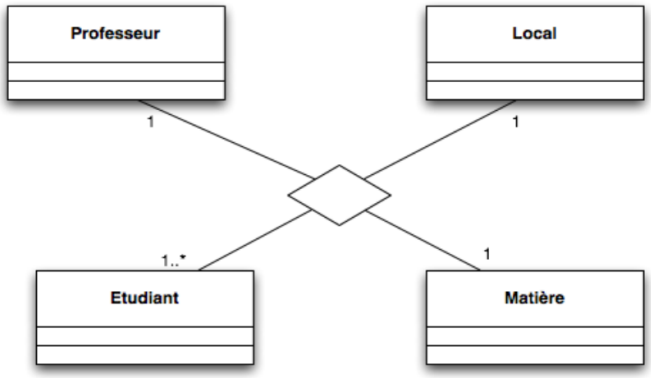

FIGURE 20 – Associations N-aire

Représentation détaillée :

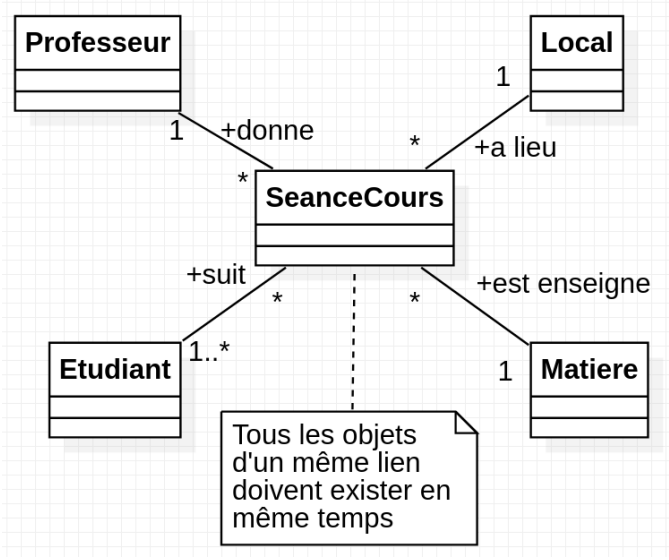

FIGURE 21 – Associations N-aire 2

6 Héritages

Classification des classes dans une hiérarchie.

1. **Exemple "point" :**
   Diagramme *UML* d'une classe *Point* et d'une classe *ColoredPoint* qui **hérite** de *Point* :

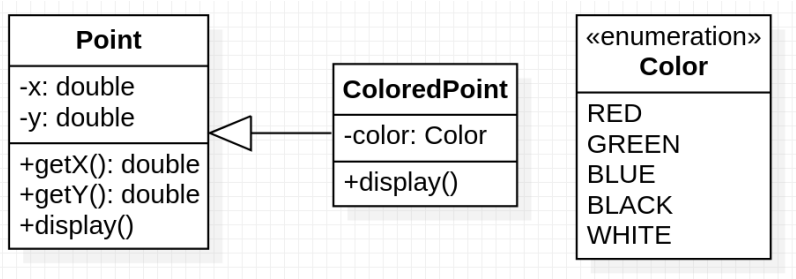

FIGURE 22 – Heritages

Le code Java ressemblerait à ceci :

```
1 public class Point{
2    private double x;
3    private double y;
4
```

```
    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    //Getters, setters...
    public void display() {
        System.out.println("(" + x + ", " + y +")");
    }
}
```

```
public class ColoredPoint extends Point {
    private Color color;

    public ColoredPoint(double x, double y, Color color) {
        super(x, y);
        this.color = color;
    }
    //Getters, setters...
    @Override
    public void display() {
        super.display();
        System.out.println(" Color: " + this.color);
    }
}
```

7 Interfaces

Collection de méthodes qui décrit le service d'une classe ou d'un composant.

1. **Exemple "animal" :**
   Compagnon est une **interface**, et *Chat* et *Chien* **héritent** de *Animal* :

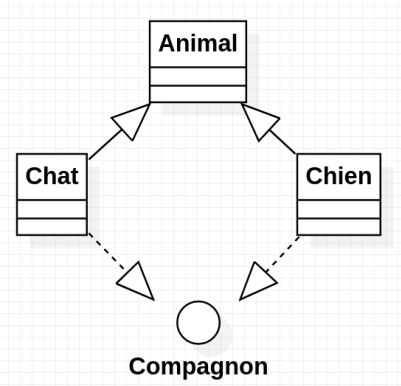

FIGURE 23 – Interfaces

Le code Java ressemblerait à ceci :

```
public class Animal {
    //Constructeur, getters, setters...
    public void manger() {...}
}

public class Chien extends Animal implements Compagnon {
    //Constructeur, getters, setters
    public void etreAmical() {...}
    public void jouer() {...}

    @Override
    public void manger() {...}
```

```
13 }
14
15 public interface Compagnon {
16    public abstract void etreAmical();
17    public abstract void jouer();
18 }
```

8 Packages

Éléments d'organisation.

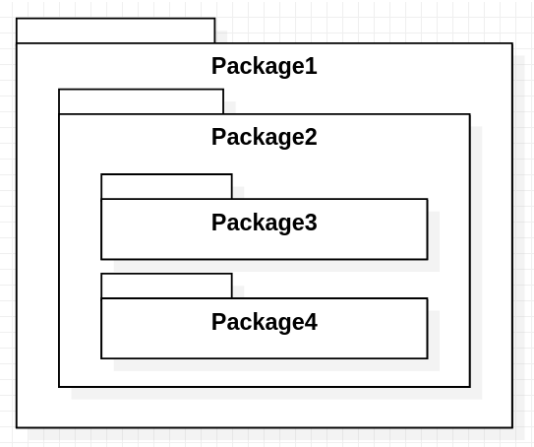

FIGURE 24 – Packages