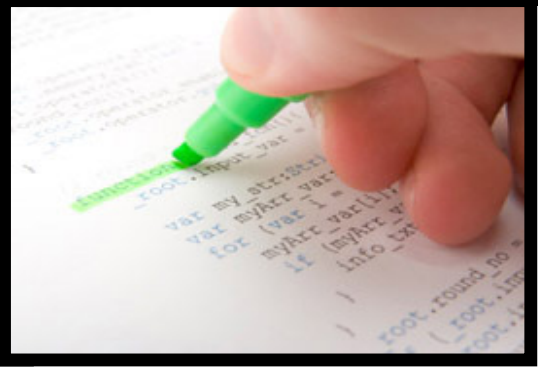
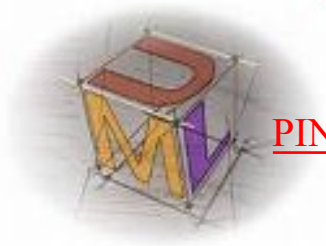
UNIVERSITE DE CORSE
2023-2024
Licence SPI 2ème année

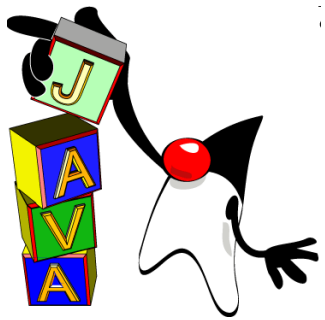
UE Programmation Orientée Objet

CH 3 – COMMUNICATION ENTRE OBJETS





Evelyne VITTORI
vittori_e@univ-corse.fr
Paul PINA-GHERARDI
PINA-GHERARDI_p@univ-corse.fr



Programmation Orientée objet
Plan du Cours

CH1 – PARADIGME ORIENTE OBJET

CH2 – OBJETS et CLASSES

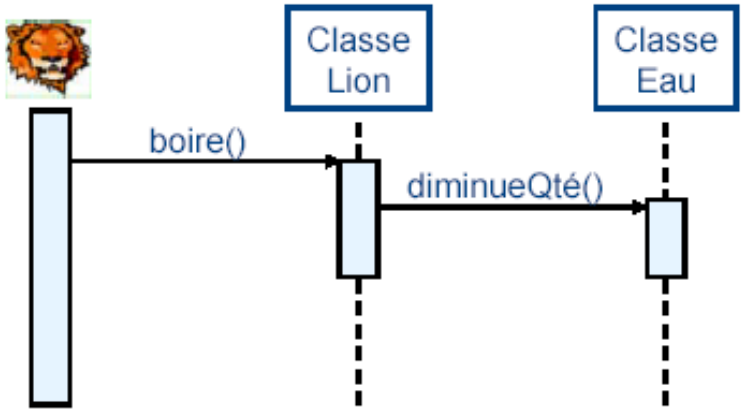
CH3 – COMMUNICATION ENTRE OBJETS

CH4 – HERITAGE et POLYMORPHISME

CH3 – COMMUNICATION ENTRE OBJETS

Un programme orienté objet est uniquement constitué de classes interagissant par envoi de messages

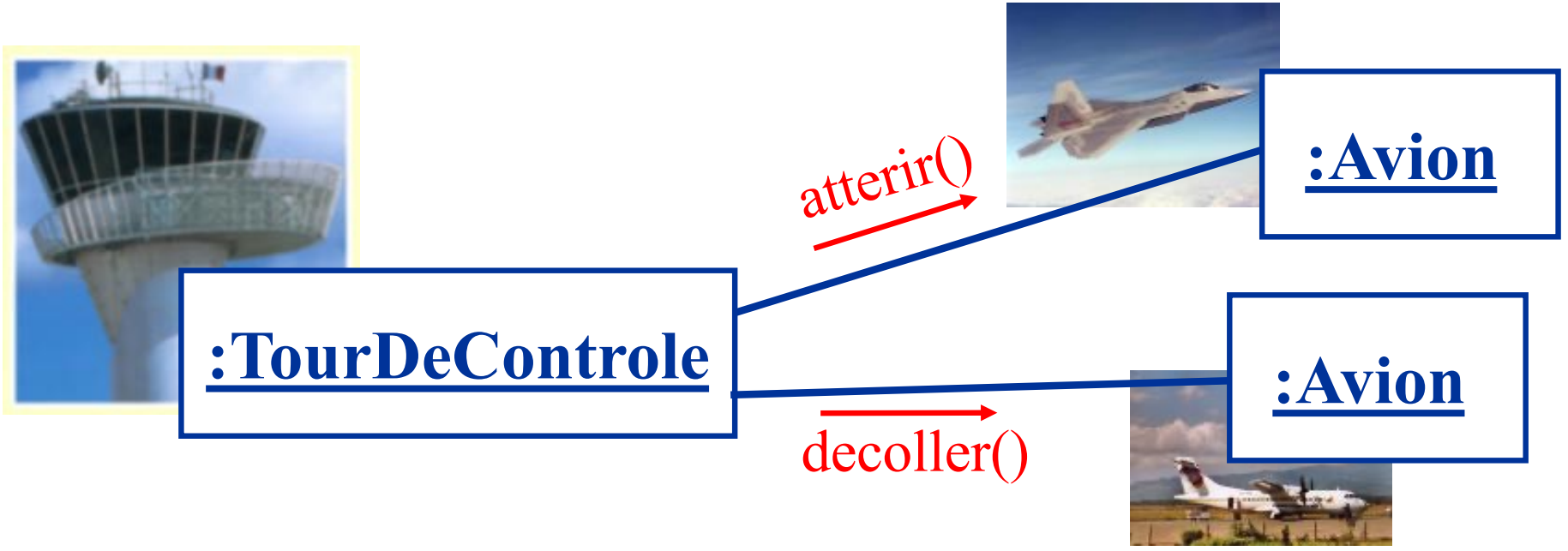
Les objets parlent donc aux objets.



Mais comment!!

CH3 – COMMUNICATION ENTRE OBJETS

Pour pouvoir s'envoyer des messages les objets doivent se connaître.



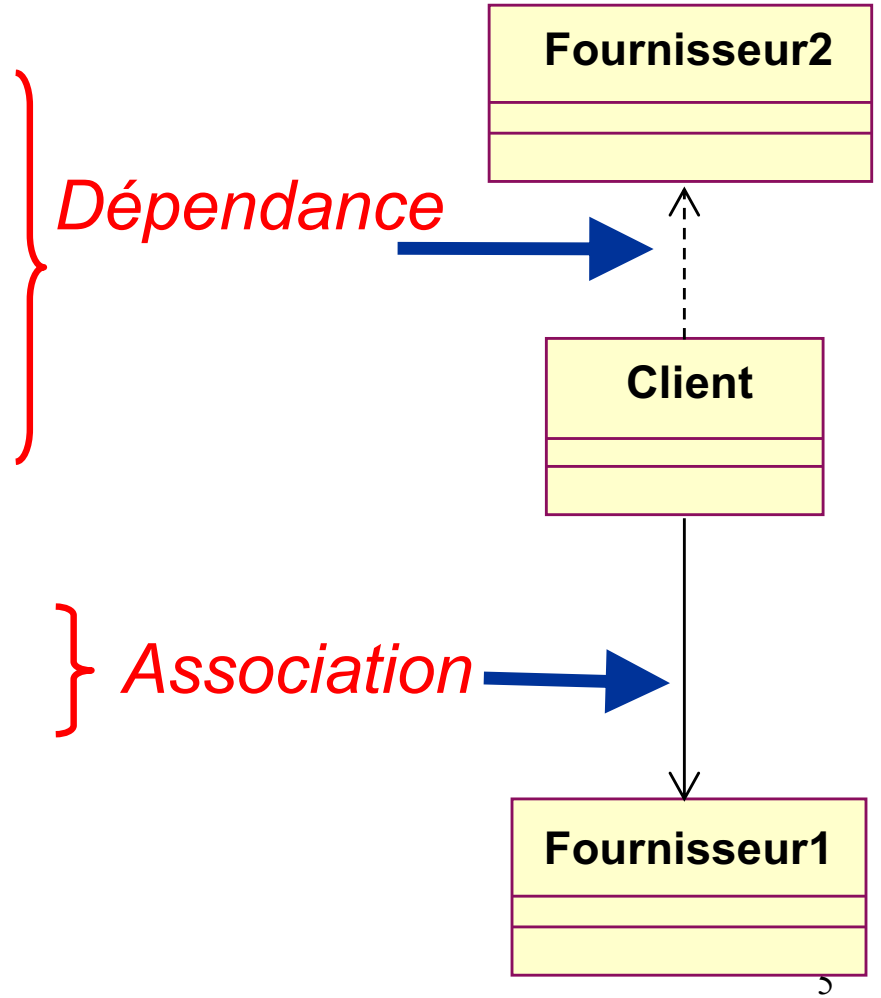
Existence d'une relation de communication entre classes

- **Relation permanente : ASSOCIATION** — attributs
- **Relation passagère: DEPENDANCE** — Variables locales

CH3 – COMMUNICATION ENTRE OBJETS

- Pour “se connaître” les objets doivent être visibles
  - –Objet variable locale
  - –Objet paramètre
  - –Objet résultat

  - –Objet variable d’instance



CH3 – COMMUNICATION ENTRE OBJETS

UML définit quatre principaux types de relations entre classes

- Association:
  - Association simple
  - Agrégation
  - Composition
- Dépendance

cf. cours UML L3 informatique

- Généralisation (Héritage)
- Réalisation

cf. CH4

CH3 – COMMUNICATION ENTRE OBJETS

3.1 - Relations de communication en UML

1. Associations
   - Définition et Représentation
   - Représentation détaillée

     Noms, Rôles, Cardinalités, Navigabilité
2. Dépendance

3.2 - Représentation en Java

Associations en UML

Définition et Représentation

Une association représente une relation structurelle entre des classes dont les instances ont besoin de communiquer tout au long de leur vie.

Une association entre classes

Associations en UML

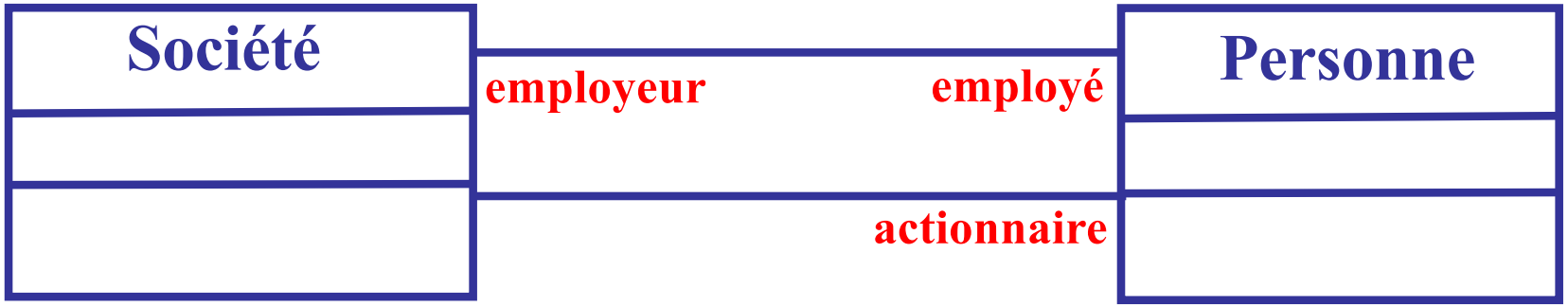
Nommage

- Nom de l'association en italique au milieu de la ligne
- Forme verbale active ou passive

Associations en UML

Rôles

- Un rôle définit la manière dont une classe intervient dans une association

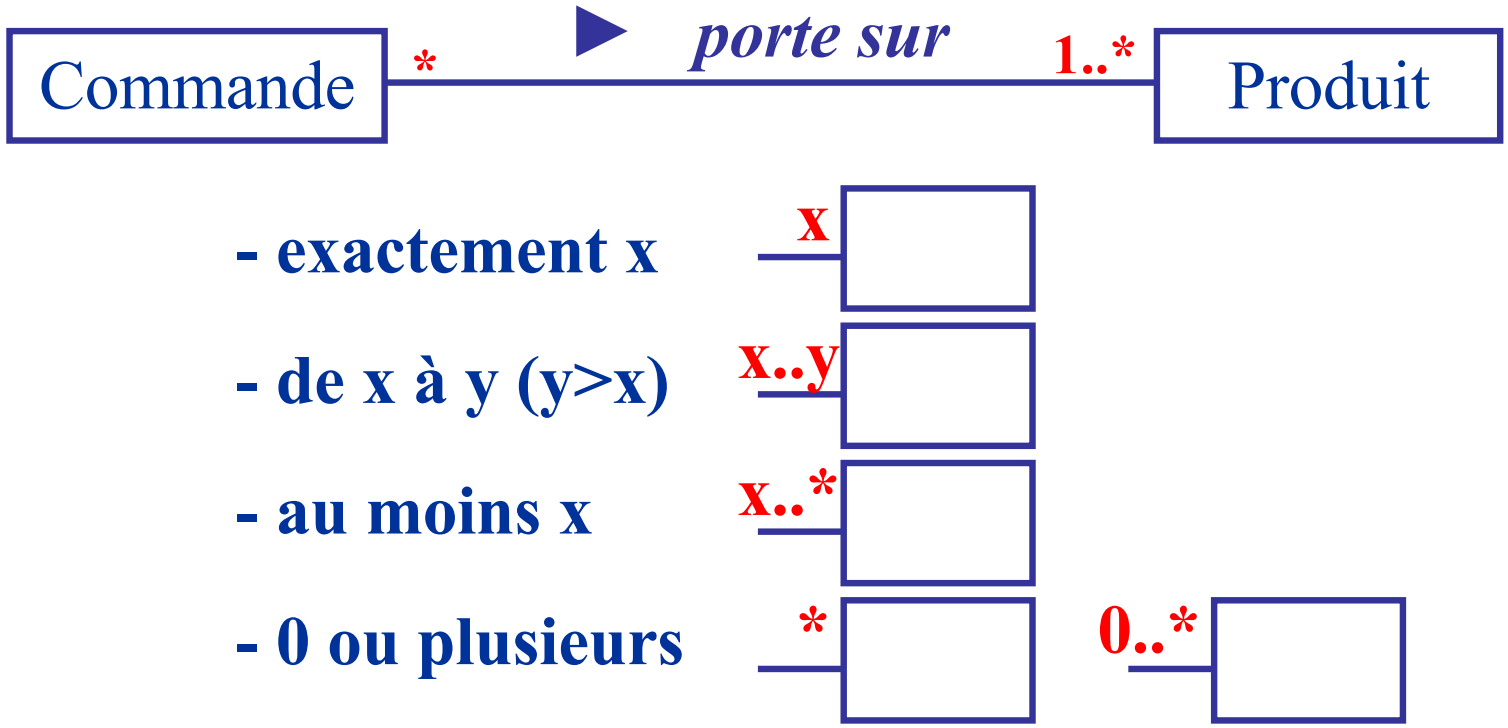


- Le rôle est indispensable lorsqu'il y a plusieurs associations entre 2 classes

Associations en UML

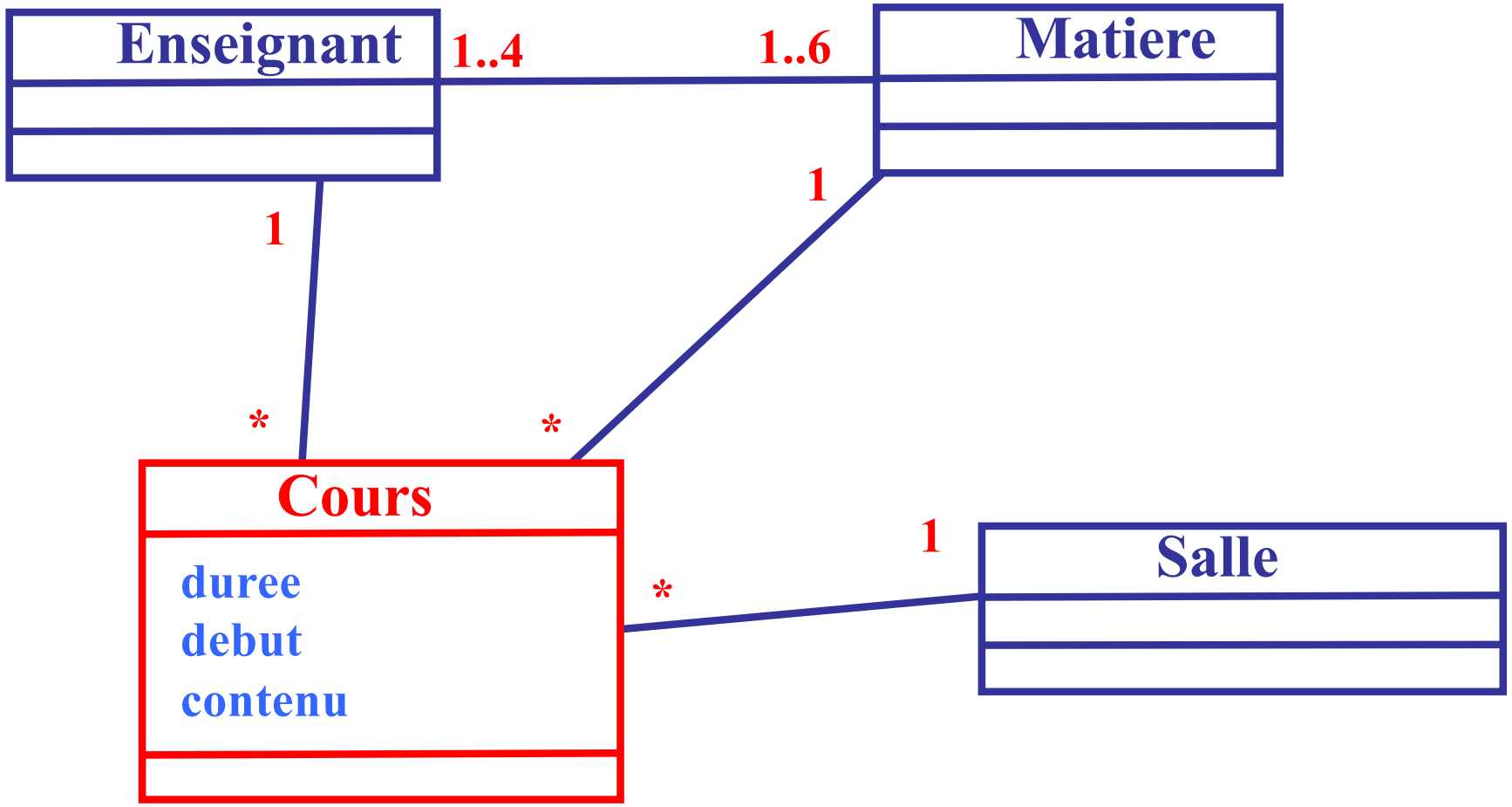
Cardinalités

- La cardinalité d'une association précise le nombre d'instances qui participent à une relation



Associations en UML

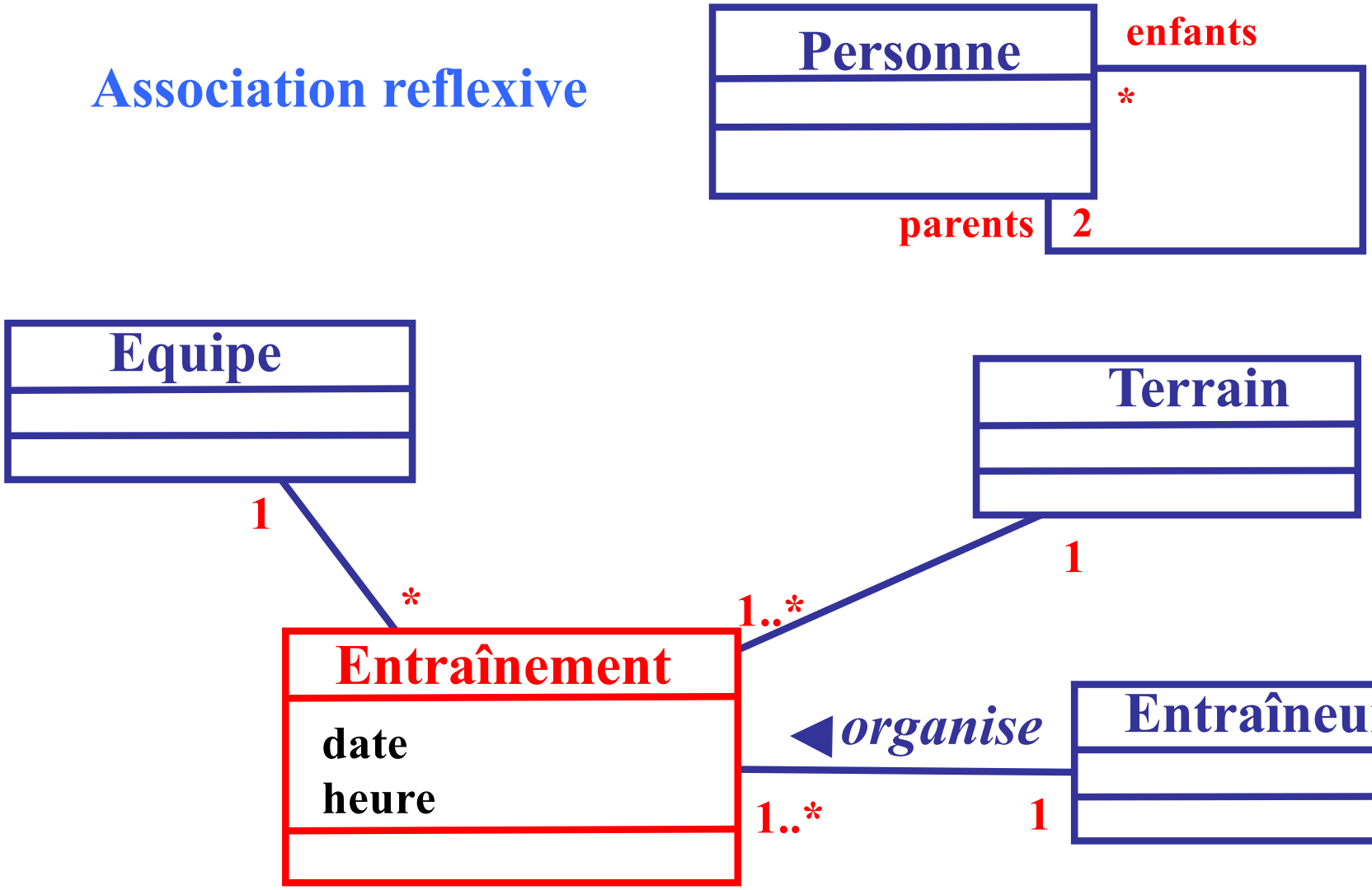
Cardinalités



Associations en UML

Cardinalités

Association reflexive



Associations en UML

Navigabilité

- Par défaut, une association est navigable dans les deux sens
- on peut exprimer le fait que les instances d ’une classe ne « connaissent » pas les instances d ’une autre classe

CH3 – COMMUNICATION ENTRE OBJETS

3.1 - Relations de communication en UML

1. Associations

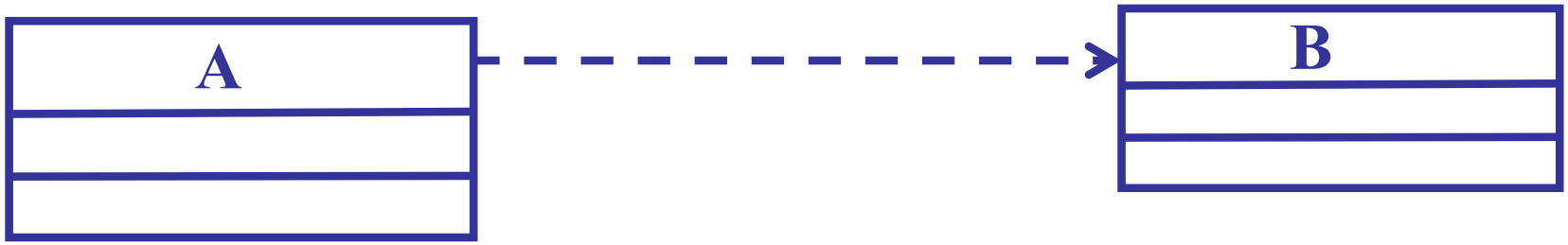
   - Définition et Représentation
   - Représentation détaillée

     Noms, Rôles, Cardinalités, Navigabilité

2. Dépendance

3.2 - Représentation en Java

Dépendances



Quelque part dans A on manipule des instances de B

Dans la classe A, il existe une (ou plusieurs) méthode(s) qui possède(nt):

- une variable locale de type B

et/ou

- un paramètre de type B

et/ou

- un résultat de méthode de type B

Tout changement dans B peut avoir des répercussions sur A.

Dépendance et association

- L'association représente une connexion structurelle entre les objets des classes associées.

Une classe possède des attributs définis sur une autre classe

- La dépendance est une relation d'usage, une communication momentanée, limitée dans le temps

Une classe manipule/utilise des instances d'une autre classe dans ses méthodes

CH3 – COMMUNICATION ENTRE OBJETS

3.1 - Relations de communication en UML

3.2 - Représentation en Java

1. Principe général
2. Associations monovaluées
   - Aspects statiques
   - Aspects dynamiques
3. Associations multivaluées

Représentation des associations

Principe général

Aspect statique :

structure des classes « associées »

Les associations sont traduites par des attributs dans les classes :

- **Associations monovaluées:**

  multiplicités 0 ou 1: références

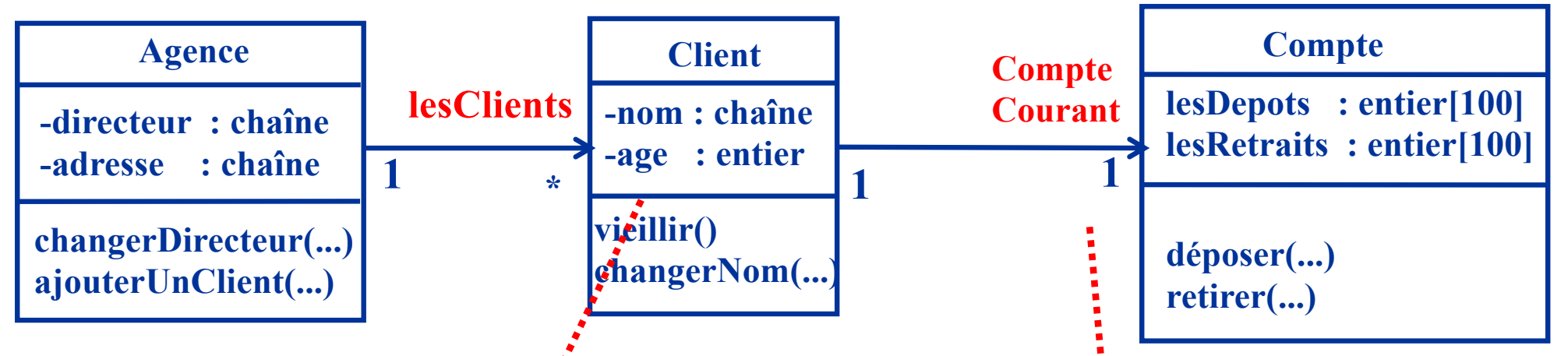
- **Associations multivaluées:**

  multiplicités > 1 : tableaux de références, vecteurs, …

Représentation des associations

Principe général

Aspect statique




```
class Agence {
   ....
   private Client[] lesClients;
   ...
}
```

```
class Client {
   ....
   private Compte compteCourant;
   ...
}
```

Représentation des associations

Principe général

Aspects dynamiques :

Instanciation des classes « associées »

La localisation de la création des objets associés dépend des cardinalités.

CH3 – COMMUNICATION ENTRE OBJETS

3.1 - Relations de communication en UML

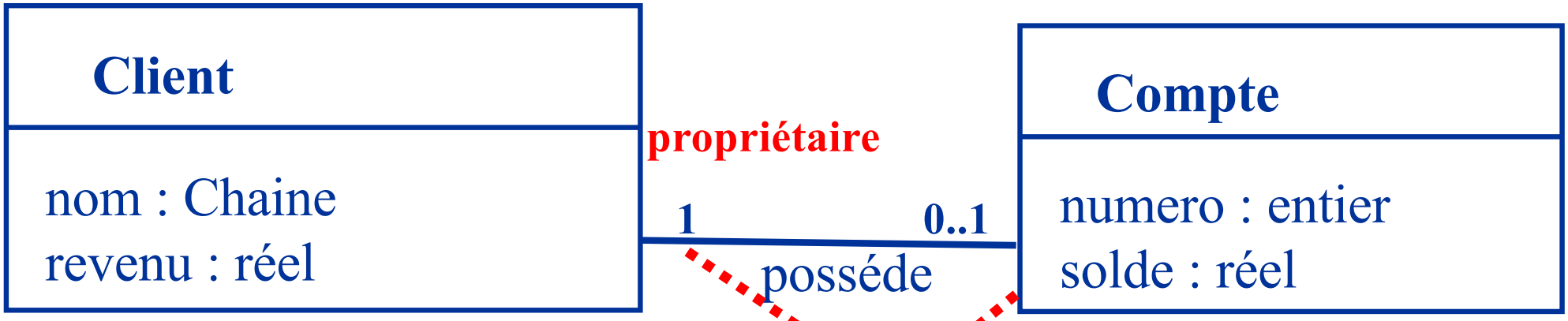
3.2 - Représentation en Java

1. Principe général
2. Associations monovaluées
   - Aspects statiques
   - Aspects dynamiques
3. Associations multivaluées

Associations monovaluées

ASPECTS STATIQUES

Création d'attributs références



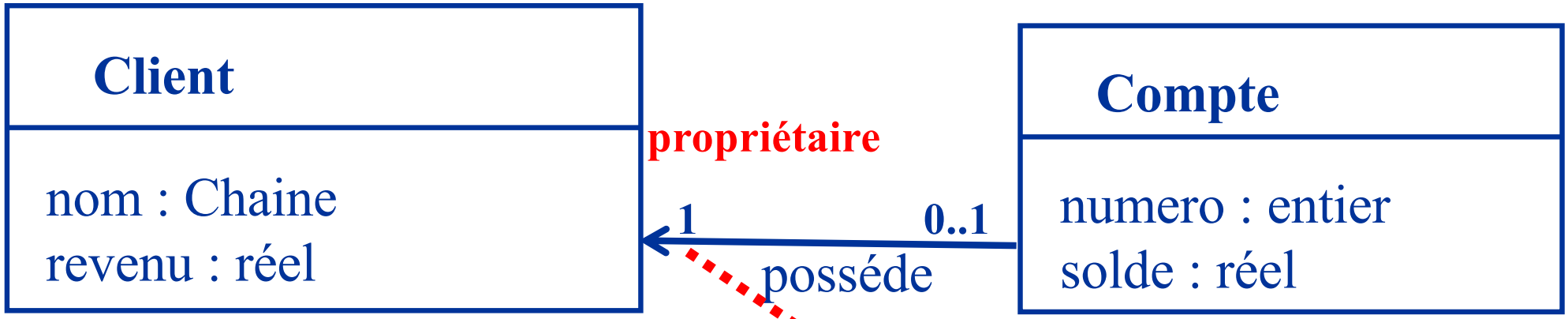

```
class Client {
   private String nom;
   private  double revenu;
   private Compte compteCourant;
     …
}
```

```
class Compte {
   private int   numero;
   private double solde;
   private Client proprietaire;
     …
}
```

Associations monovaluées

ASPECTS STATIQUES

Navigabilité restreinte



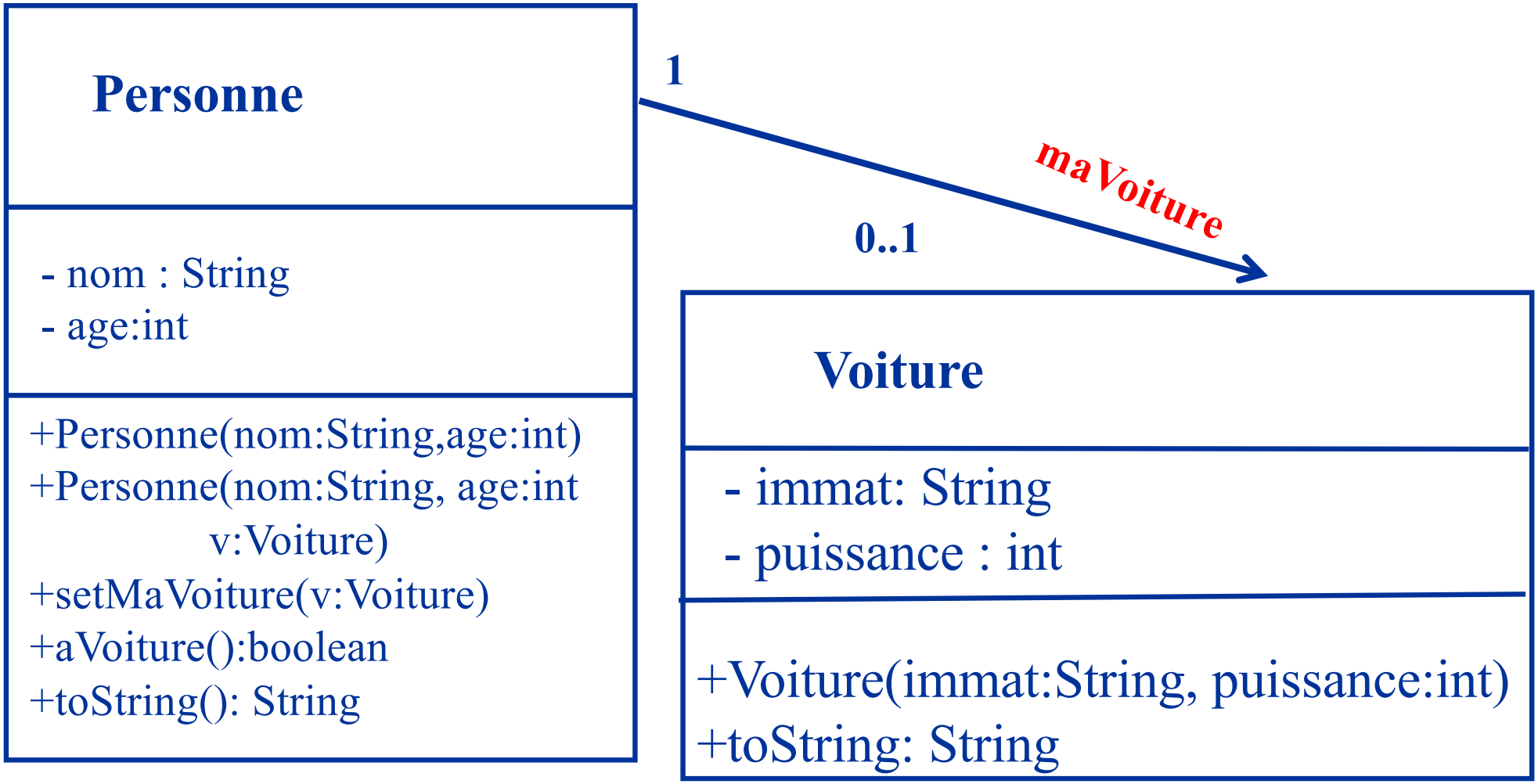

```
class Client {
   private String nom;
   private  double revenu;
…
}
```

```
class Compte {
   private int   numero;
   private double solde;
   private Client proprietaire;
     …
}
```

Absence d'attribut référence

TPCours - Exercice 3.1

Programmer en java les classes suivantes
(vous pouvez réutiliser votre classe Personne du chapitre 2)



TPCours - Exercice 3.1

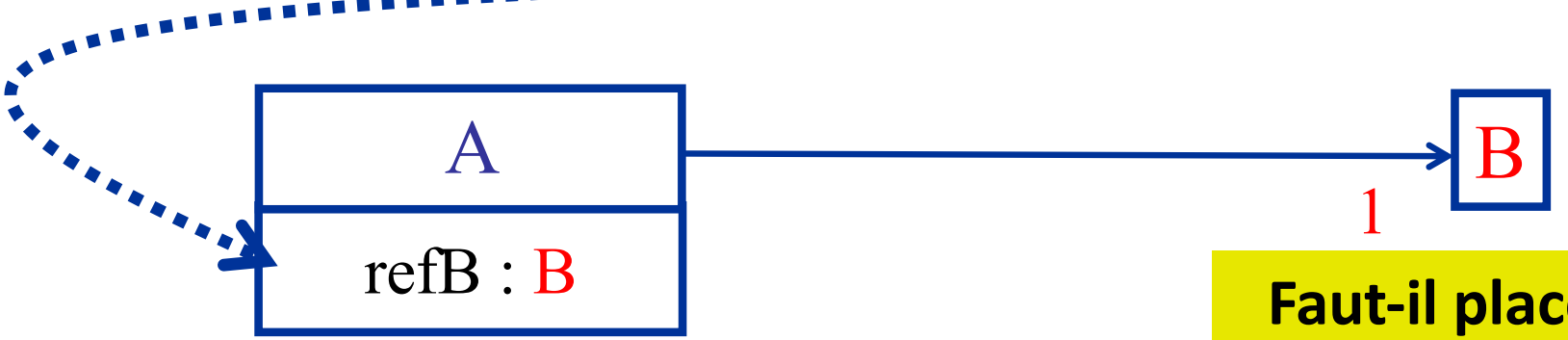
- Redéfinissez les méthodes toString dans vos classes Personne et Voiture:
  - Méthode toString de Voiture: retourne «voiture XXXIIJ34 de puissance 5 »
  - Méthode toString de Personne retourne «Titi 19 ans conduit la voiture XXXIIJ34 de puissance 5 »
- Définissez un programme de Test dans une classe TestPersVoit:
  - Création d'une voiture, création d'une personne possédant cette voiture
  - Affichage la personne créée
  - Création et affichage d'une deuxième personne sans voiture

```
Titi 19 ans  conduit la voiture XXXIIJ34  de  puissance 5
Toto 20 ans  n'a pas de voiture
```

Associations mono-valuées

ASPECTS DYNAMIQUES

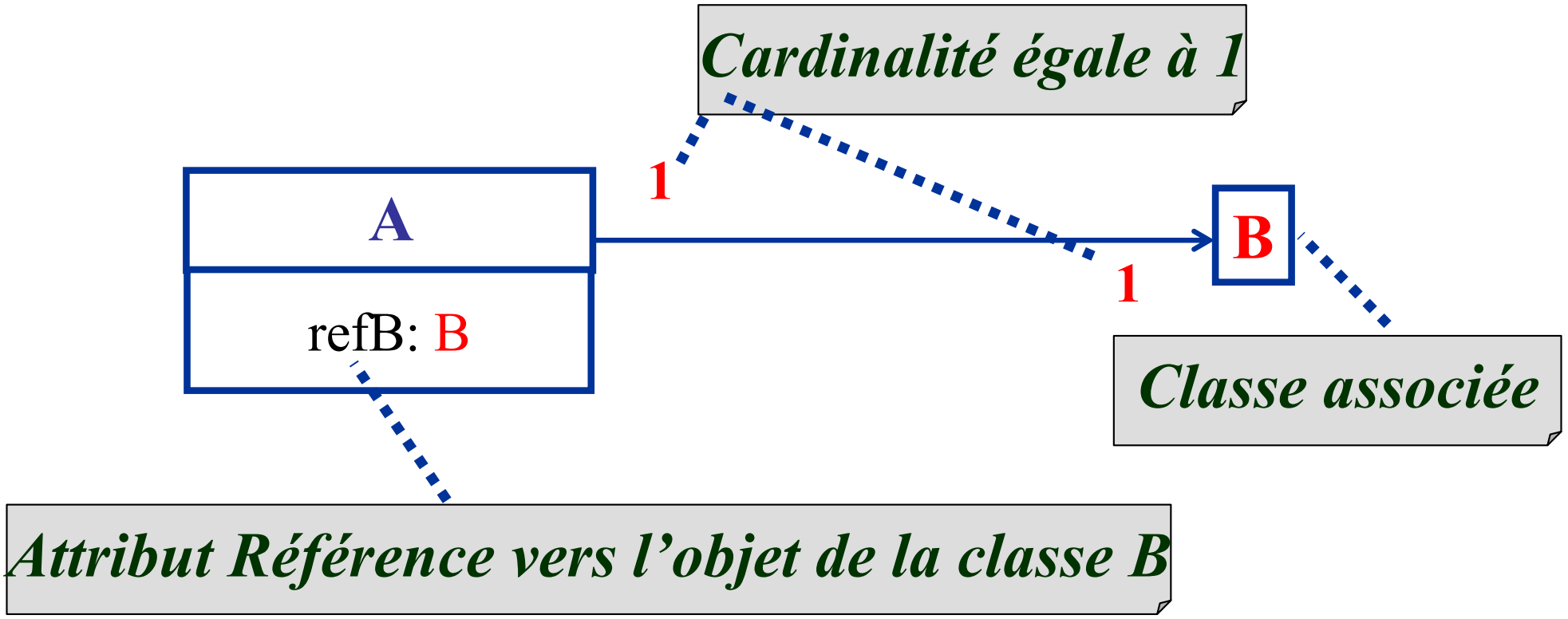
Localisation de la création de l'objet référencé



Faut-il placer le new B() dans le constructeur de A?

1. **Objet Obligatoire (cardinalité 1)**
   a. Instanciation dans le constructeur (cardinalité 1 – 1) (new B())
   b. Transmission d'une référence en paramètre dans le constructeur (l'objet B est créé avant et passé en paramètre)
2. **Objet Facultatif (cardinalité 0..1)** : Affectation d'une référence en dehors du constructeur

1.a- Représentation des associations obligatoires 1-1



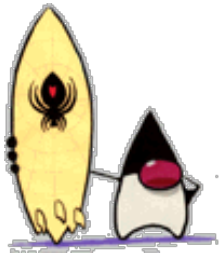
Instanciation de l'objet associé (refB) dans le constructeur de la classe A.

1.a - Représentation des associations Obligatoires

Instanciation de l'objet associé dans le constructeur

```
class A {
   …
   private B   refB;

   public A(…){
      refB=new B(..); //création de l'objet associé
     …
}
```

```
class B {//Classe associée
…
}
```

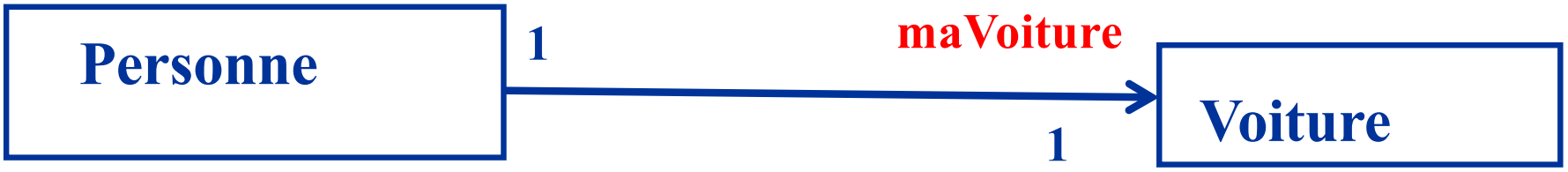


Exercice 3.2

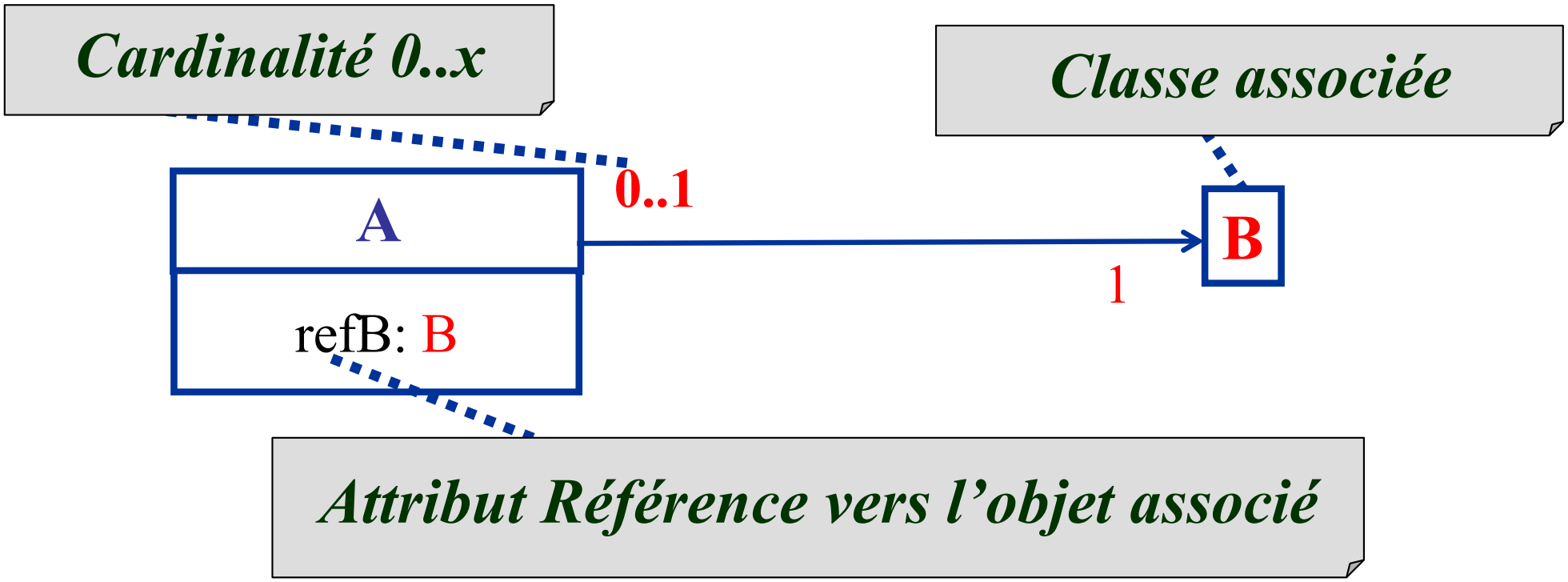
Reprenez vos classes de l'exercice 3.1

1 – Définissez une première version de votre classe Personne correspondant au cas diagramme suivant :

- Cas 1.a – association obligatoire



1.b - Associations obligatoires

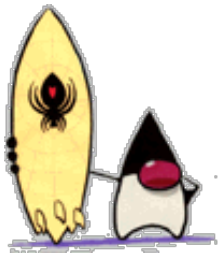


Transmission de la référence de l'objet associé (et non instanciation!!) en paramètre dans le constructeur de la classe A

1.b - Associations obligatoires

Transmission de la référence de l'objet associé en paramètre

```
class    {…

   private B   refB;

   public A(B objetB){  //objetB existe déjà
       refB= objetB;
     …
}
```



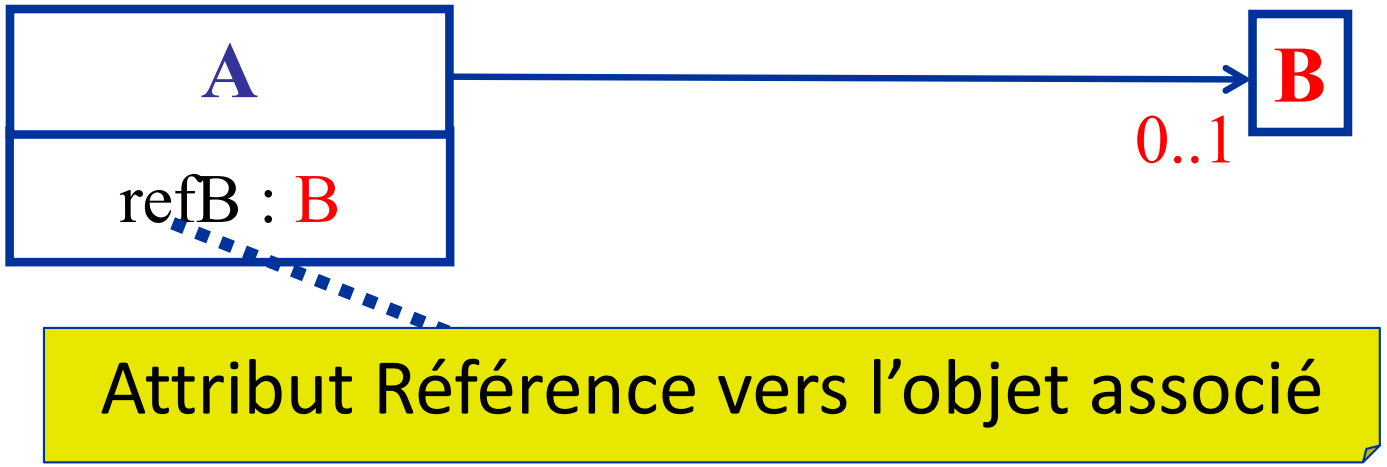
```
class B { //classe associée
…
}
```

Exercice 3.2

2 - Définissez une deuxième version de votre classe Personne correspondant au cas diagramme suivant :

- Cas 1.b – association obligatoire

2 - Associations Non Obligatoires



- Aucune liaison n'est établie obligatoirement à la création d'un objet A
- Affectation d'une référence par invocation d'une méthode spécifique

2 -Associations Non Obligatoires

L'objet associé est initialisé à la valeur par défaut (Null)

```
class    {
   …
   private B   refB;

   public A(…){
      // refB n'est pas mentionné dans le constructeur
      // ou initialisé avec une valeur Null
   }
   public setRefB(B objetB){
        refB=objetB;
   }
…
```

Le constructeur initialisant refB à null est obligatoire mais il peut y avoir d'autres constructeurs

```
class B {//classe « associée »
…
}
```

Exercice 3.2

3 - Définissez une troisième version de votre classe Personne correspondant au cas diagramme suivant :

- Cas 2 – association non obligatoire

CH3 – COMMUNICATION ENTRE OBJETS

3.1 - Relations de communication en UML

3.2 - Représentation en Java

1. Principe général
2. Associations monovaluées
3. Associations multivaluées
   - Tableaux statiques en java
   - Traduction des associations multivaluées
   - Associations et Tableaux statiques
   - Associations et Tableaux dynamiques en Java

Déclaration et Création d'un tableau

tableau

```
int []    unTableau = new int [nbElt] ;
```

Type des Composants
Type primitif ou **Classe**

nom de la variable

Allocation de Mémoire pour:
les **valeurs** de type primitif.
ou les **références** de type Classe.

- Taille du tableau mémorisée dans :
  - unTableau**.length**
  - Nombre d'éléments
    - Dernier élément : unTableau[unTableau.length-1]
  - Non modifiable

    unTableau.length = 5 ;

Tableau d'éléments de type primitif

int [] unTableau = new int [nbElt] ;

- unTableau[indice]

  unTableau[i] = 10;

  *// i compris entre 0 et nbElt-**1***

  int x;

  x = unTableau[i];

Éléments numérotés à partir de 0

- Erreur classique : être en dehors des limites du tableau !
- Les indices débutent en 0

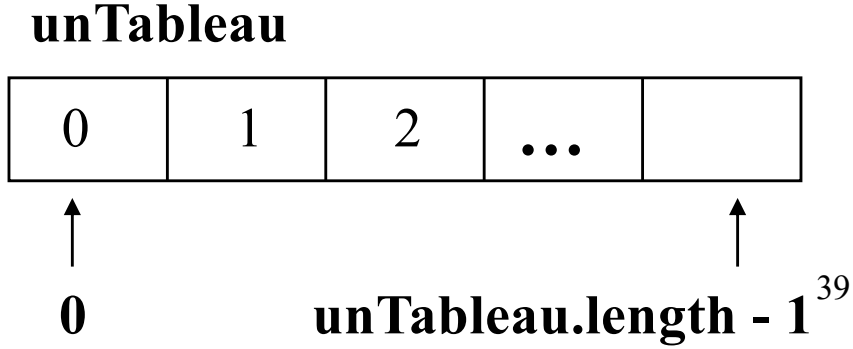
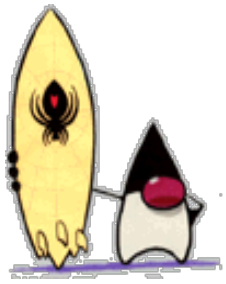


Tableau d'éléments de type primitif

```
int []   monTableau = { 1, 2, 4, 8, 16 };
```



1	2	4	8	16
0				4

Indice

Taille du tableau : 5
indice de 0 à 4

- Création du tableau et Initialisation des éléments avec les valeurs spécifiées

Manipulation de tableau

exemple Tableau d'entiers

```
int NBMAX=10; //nb d'éléments maximum
//Déclaration du tableau
int []    unTableau = new int[NBMAX] ;
int    i,    somme = 0 ;
int nbElem=0;
//Saisie clavier du nombre effectif d'éléments
nbElem= Clavier.lireInt("Entrez le nombre
  d'éléments à saisir dans le tableau : ");
//Saisie clavier des éléments du tableau
for ( i = 0; i < nbElem; i++ ) {
   unTableau[i] = Clavier.lireInt("Entrez la valeur
     de l'element " +i +   " : ");}
//Calcul de la somme des éléments du tableau
for ( i = 0; i < nbElem; i++ ) {
   somme = somme + unTableau[i];}
System.out.println("la somme vaut : " + somme );
```

Manipulation de tableaux

Méthodes utilitaires

- La classe Arrays (du package java.util) contient plusieurs méthodes statiques permettant de manipuler des tableaux:

  Exemples:

  - Tri du contenu d'un tableau d'entiers
    - public static void sort(int[] a) : tri le tableau a passé en paramètre dans l'ordre croissant
    - public static void sort(int[] a, int ideb, int ifin) : tri la partie du tableau a comprise entre les cases ideb et ifin dans l'ordre croissant
  - Test d'égalité
    - public static boolean equals(int[] a, int[] a2)

CH3 – COMMUNICATION ENTRE OBJETS

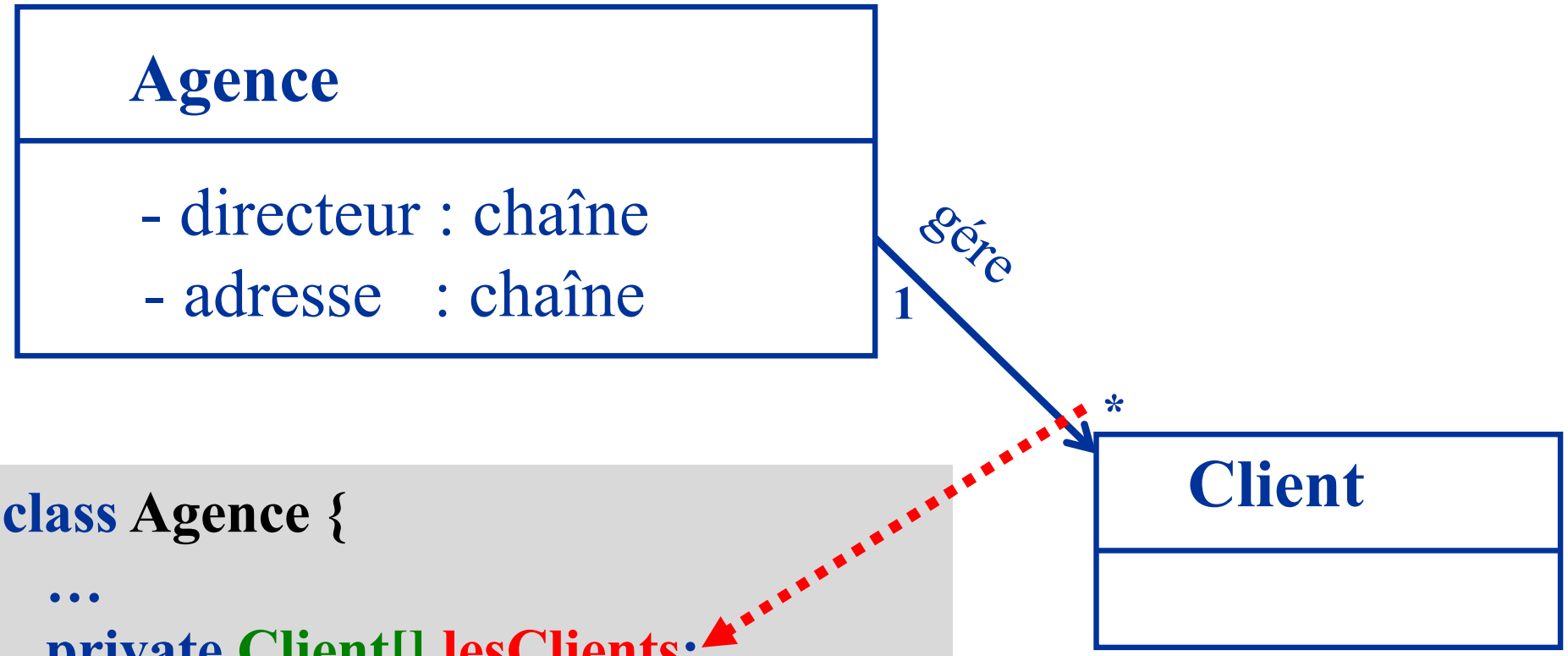
3.1 - Relations de communication en UML

3.2 - Représentation en Java

1. Principe général
2. Associations monovaluées
3. Associations multivaluées
   - Tableaux statiques en java
   - Traduction des associations multivaluées
   - Associations et Tableaux statiques
   - Associations et Tableaux dynamiques en Java

Associations multivaluées

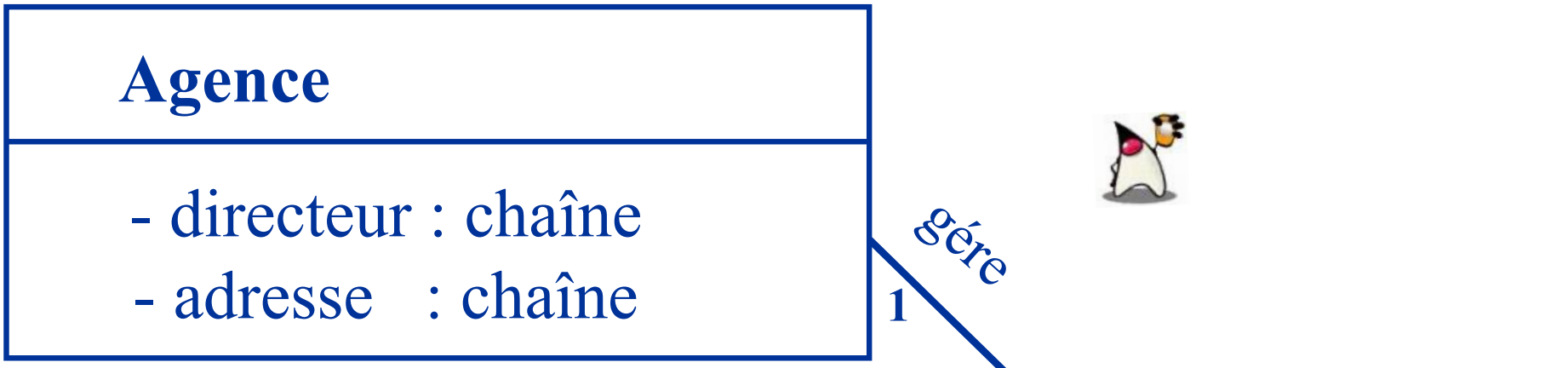
Création d'un attribut tableau de références




```
class Agence {
   …
   private Client[] lesClients;
     …
}
```

Associations multivaluées

Création d'un attribut collection d'objets




```
class Agence {
   …
   private ArrayList<Client> lesClients;
     …
}
```

ArrayList est un exemple de type Collection
Il en existe de nombreux autres : Set ….

CH3 – COMMUNICATION ENTRE OBJETS

3.1 - Relations de communication en UML

3.2 - Représentation en Java

1. Principe général
2. Associations monovaluées
3. Associations multivaluées
   - Tableaux statiques en java
   - Traduction des associations multivaluées
   - Associations et Tableaux statiques
   - Associations et Tableaux dynamiques en Java

Tableau d'objets

1 - Déclaration

```
Client[] mesClients;
```

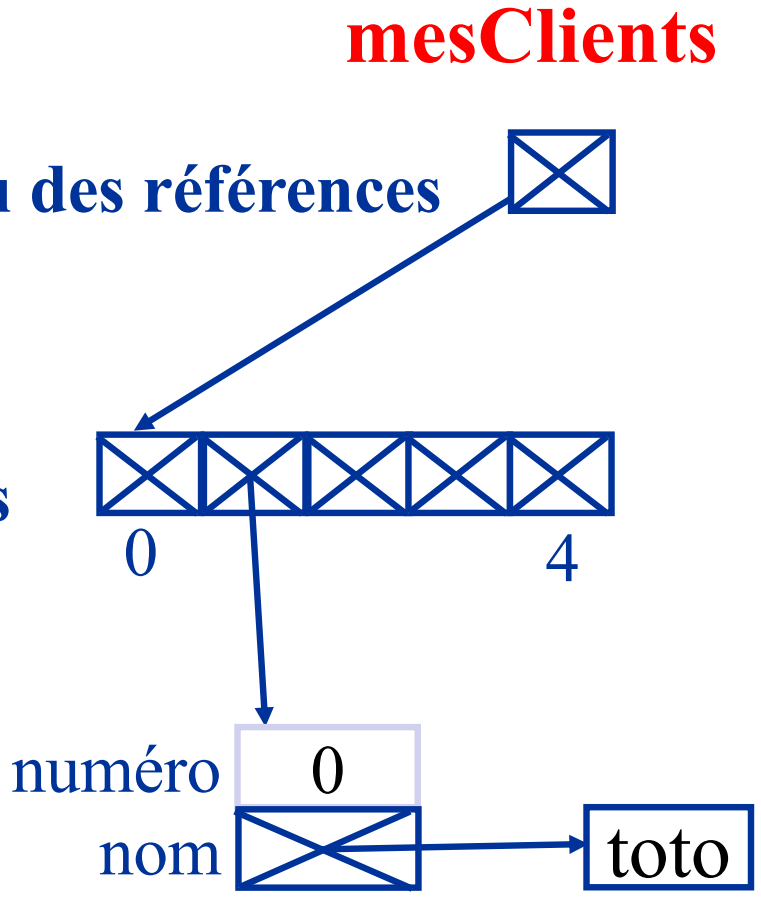
2 – Instanciation: Création du tableau des références

```
mesClients= new Client[5];
```

3 – Création (ou affectation) des objets

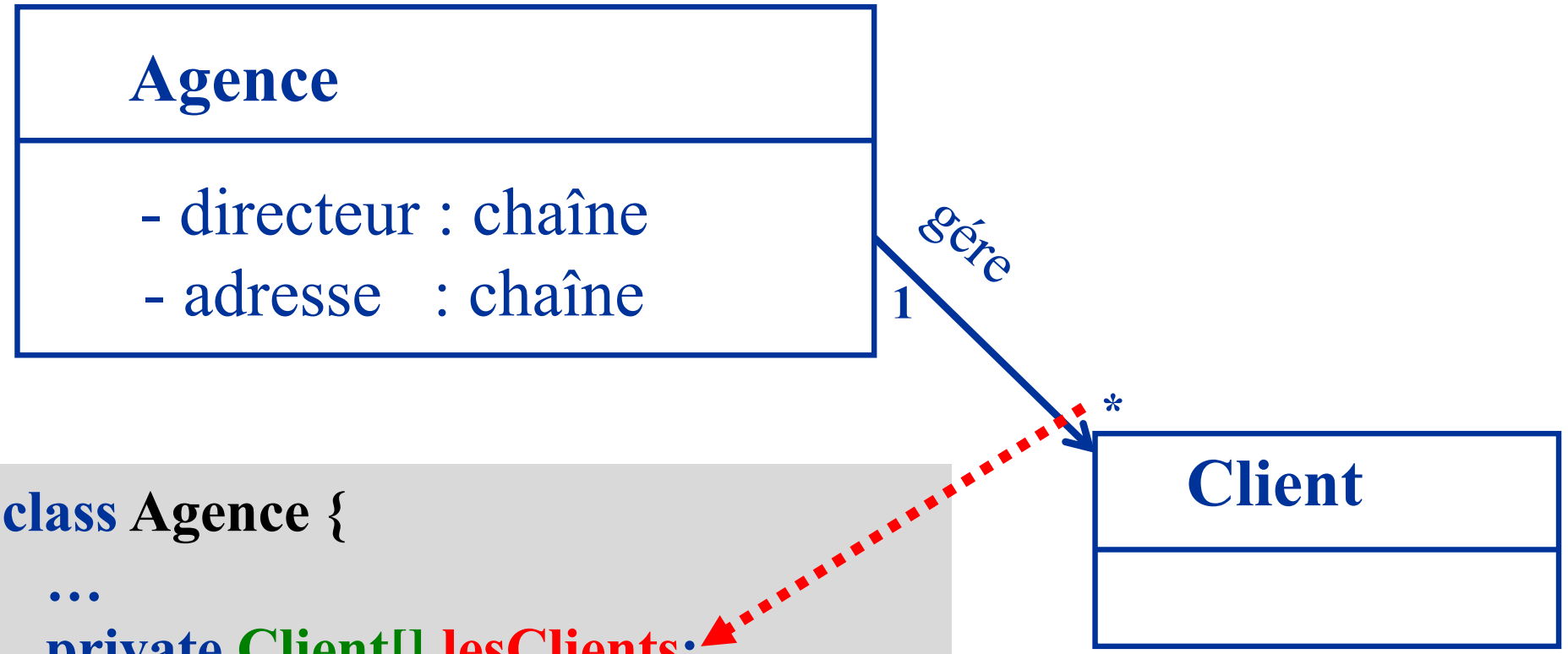
```
mesClients[1]= new Client();
mesClients[1].nom="toto";
```

```
mesClients[1].setNom("toto");
```



Tableaux et Associations multivaluées

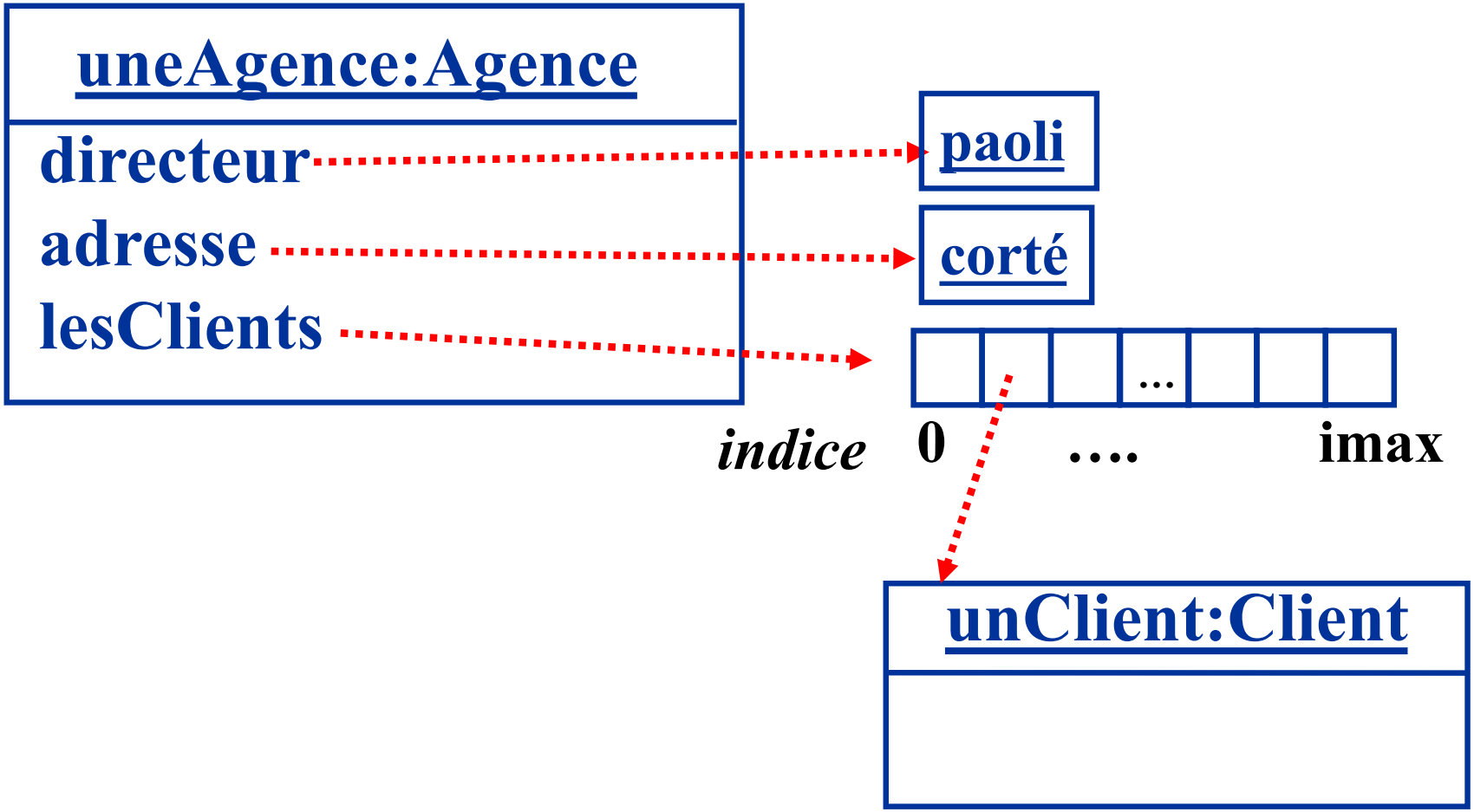
Création d'un attribut tableau de références




```
class Agence {
   …
   private Client[] lesClients;
     …
}
```

Traduction des Associations multivaluées

Tableau de références sur des objets



Tableaux d'objets en JAVA
Déclaration et instanciation

```
class Client {
   private String   nom;
   private int age;
   public Client(…….)
   ….
}
```

```
class Agence {
    public static final int NBMAX=500;
    private String   directeur;
    private String   adresse;
    private int nbClients;
    private Client[]   lesClients;
     public Agence( String directeur, String adresse)
     {
          lesClients = new Client[NBMAX];
          …
     }
```

Création du tableau pas des objets!!

Tableaux d'objets en JAVA

```
class Agence {
  public static final int NBMAX=500;
  private String   directeur;
  private String   adresse;
  private int nbClients = 0 ;
  private Client[]    lesClients;
  public Agence( String directeur, String adresse) {
      this.directeur= directeur;
      this.adresse=adresse;
      lesClients = new Client[NBMAX]; …}

   public void  ajouterUnClient(Client unClient) {
      if (nbClient<NBMAX){
          lesClients[nbClients] = un Client;
          nbClients ++ ;
      }
   }}
```

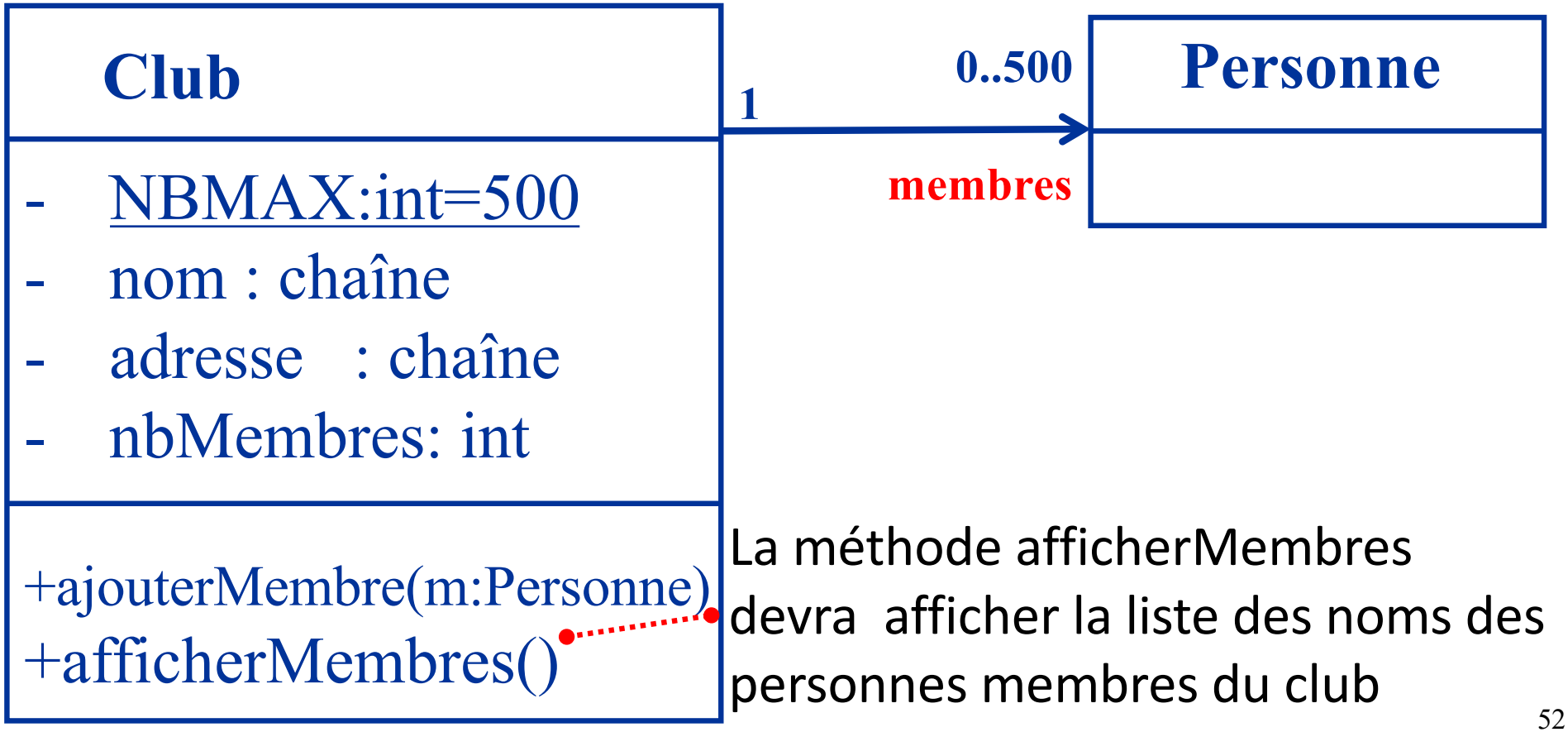
Nombre d'éléments effectifs du tableau

Instanciation du du tableau

Méthode d'ajout de références

TPCours – Exercice 3.3

Programmer en java la classe Club.
Définissez une classe de test TestClub pour tester le fonctionnement de votre classe Club.



La méthode afficherMembres devra afficher la liste des noms des personnes membres du club

CH3 – COMMUNICATION ENTRE OBJETS

3.1 - Relations de communication en UML

3.2 - Représentation en Java

1. Principe général
2. Associations monovaluées
3. Associations multivaluées
   - Tableaux statiques en java
   - Traduction des associations multivaluées
   - Associations et Tableaux statiques
   - Associations et Tableaux dynamiques en Java

Tableaux dynamiques en Java

Un tableau dynamique est un objet de la classe ArrayList (java.util.ArrayList)

- ArrayList est une des nombreuses classes de l'API Java permettant de définir des collections ou containers d'objets
- Notion de liste de références d'objets
- Taille extensible en fonction des besoins

- Avant Java 5, ArrayList d'objets non différenciés (cf. compléments)

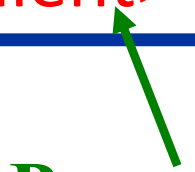
ArrayList : *la liste peut contenir des objets de différentes classes*

Tableaux dynamiques en Java

(à partir de Java 5)

A partir de la version 5, grâce à la généricité, Java permet de spécifier le type d'objets contenus dans un ArrayList lors de sa déclaration:

ArrayList <String> : *la liste ne peut contenir que des chaînes*
ArrayList <Client> : *la liste ne peut contenir que des Clients*



Paramètre de généricité

ArrayList

Déclaration et Instanciation d'un ArrayList

```
ArrayList<Element> a= new ArrayList<Element> ()
```

a est un objet de type ArrayList contenant des objets de type Element

phrase

Exemples:

Création d'une liste vide de capacité initiale 10

```
ArrayList<String> phrase= new ArrayList<String> ()
```

Création d'une liste vide de capacité initiale 50

```
ArrayList<String> phrase= new ArrayList<String> (50)
```

Méthodes de manipulation d'un ArrayList

- Ajout :
  - à la fin :

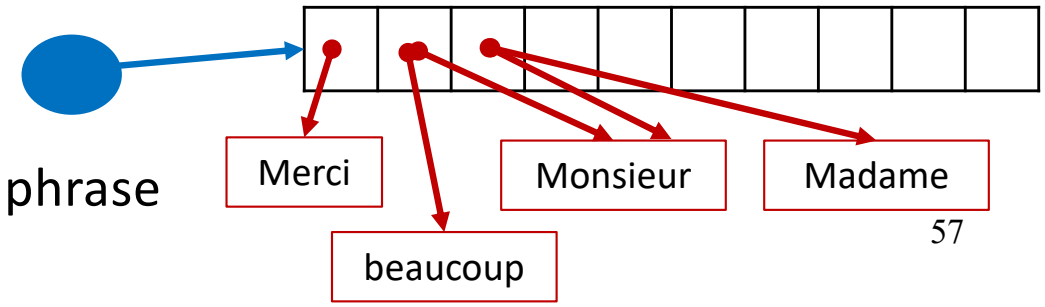
    boolean **add**(**Element** obj)
  - à la position index :

    void **add** (int index, Element obj,)

    index est compris entre 0 et nombre d 'éléments de la liste -1
- Remplacement :
  - à la position index: renvoie l'élément précédemment situé à cette position

    Element **set**(int index, Element obj,)

```
phrase.add("Merci" );
phrase.add(" Monsieur");
phrase.add(1," beaucoup" );
phrase.set( 2," Madame" );
```



Méthodes de manipulation d'un ArrayList

- Déterminer le nombre d'éléments (taille « utile »)

  `int size ()` : renvoie toujours 0 après la création de la liste

- Tester si la liste est vide

  `boolean isEmpty ()`

- Obtenir un élément

  `Element get( int index )`

- Rechercher
  - `boolean contains ( Element obj )`
  - `int indexOf ( Element obj )` : retourne -1 si la recherche n'aboutit pas
  - `int lastIndexOf ( Element obj )`: index de la dernière occurrence

Méthodes de manipulation d'un ArrayList

- Suppression de l'élément situé à la position index

  Element remove (int index)

  Renvoie l'élément supprimé

- Vidage de l'arrayList

  void clear ()

- Suppression de la première occurrence d'un objet

  boolean remove (Element obj)

  Le résultat est false si l'objet n'est pas trouvé

```
phrase.remove ( "beaucoup" );
```

Etapes de définition d'un ArrayList

1 - Déclaration

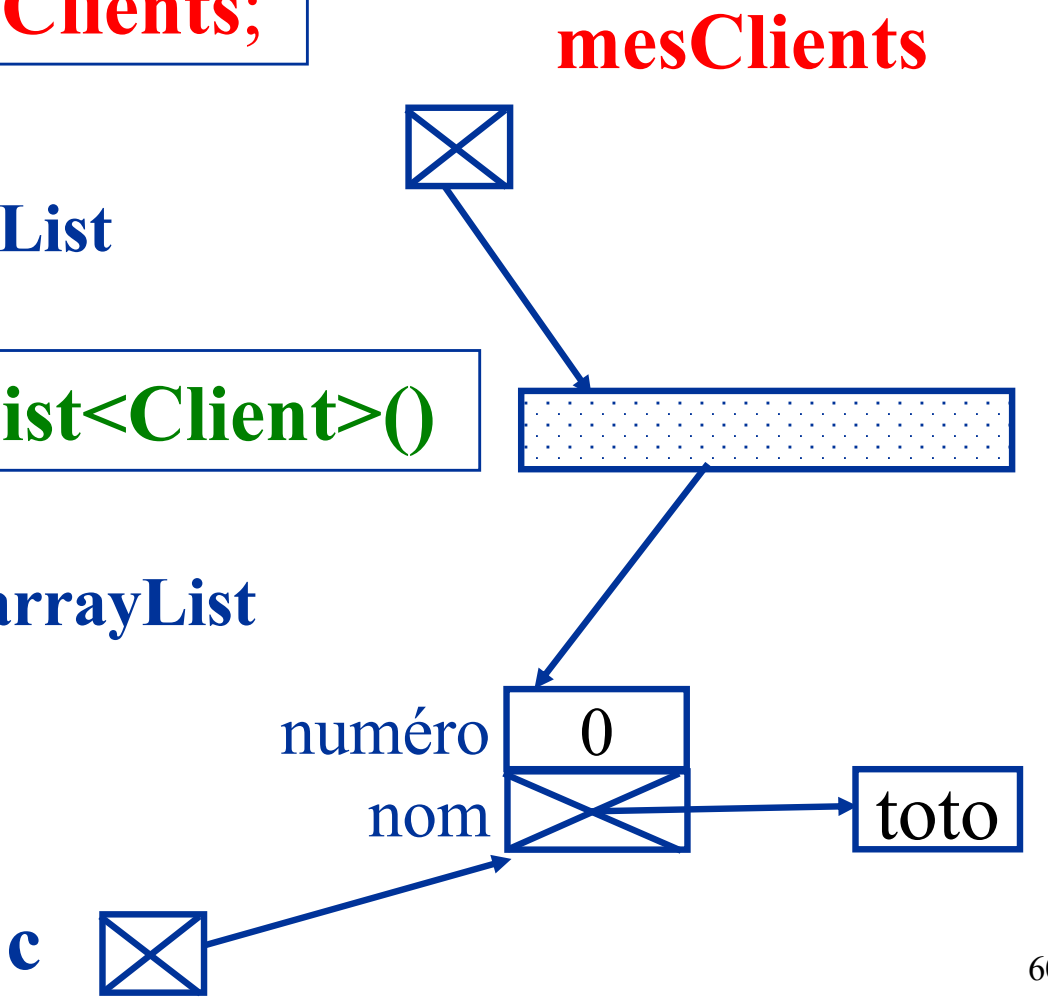
```
ArrayList<Client> mesClients;
```

2 – Création de l'objet ArrayList

```
mesClients= new ArrayList<Client>()
```

3 – Ajout d'un objet dans l'arrayList

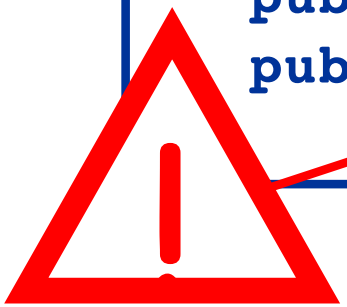
```
mesClients.add(c);
```



Liste d'objets en JAVA

```
class Client {
.... }
```

```
class Agence {
  public static final int NBMAX=500;
  private String   directeur;
  private String   adresse;
  private ArrayList<Client>  lesClients;
  public Agence( String directeur, String adresse)
  {
     lesClients = new ArrayList<Client>();
      ...}
   public void   changerDirecteur(...) { … }
   public void   ajouterUnClient(...) { … }
```



Création de l'objet ArrayList!!

Liste d'objets en JAVA

L'attribut nbClient n'est plus necessaire car le nombre effectif d'éléments d'un arraylist est donné par la méthode size()

```
class Agence {
  public static final int NBMAX=500;
  private String   directeur;
  private String  adresse;
  private ArrayList<Client> lesClients;

  public Agence(String directeur, String adresse) {
      this.directeur= directeur;
      this.adresse=adresse;
      lesClients = new ArrayList<Client>();  }


 public void   ajouterUnClient(Client unClient) {
      if (lesClients.size()<NBMAX)
          lesClients.add(unClient);
   }
}
```

Instanciation de l'objet ArrayList

Liste d'objets en JAVA
Boucle de parcours

```
class Agence {
…..
public void afficheListeNomsClients(){
   System.out.println ("Liste des noms de Clients");
   for (int i=0; i < lesClients.size() ; i++ )
      System.out.println( (lesClients.get(i)).getNom());
}
```

Accès au client d'indice i dans l'arrayList

Parcours d'un ArrayList

```
ArrayList<Element> listeElement;
for (Element var:listeElement){
    //var prend successivement la valeur de chacun des
        éléments de listeElement
}
```

```
public void afficheListeNomsClients(){
  System.out.println ("Liste des noms de
Clients");
  for (Client c:lesClients )
     System.out.println(c.getNom());
}
```

Variable de parcours de lesClients

TPCours – Exercice 3.4

Définissez une deuxième version de votre classe Club en remplaçant le tableau de membres par un arrayList.

Définissez une méthode moyenneAgeMembre() dans la classe Club. Cette méthode calcule la moyenne des âges des membres du club et renvoie un résultat de type double.

Définissez deux versions de cette méthode :

- une version utilisant une boucle for classique
- une version utilisant une boucle du type

```
for (Element var:listeElement)
```

Complétez si necessaire la classe Personne par l'ajout d'une méthode getAge().

ArrayList d’objets indifférenciés (avant Java5)

COMPLÉMENTS (POUR INFORMATION)

Wrapper classes, AutoBoxing, Boxing, Unboxing

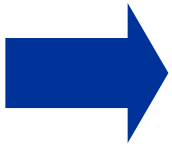
ARRAYLIST DE TYPE PRIMITIF

ArrayList d'éléments de type primitif

Un ArrayList ne stocke que des objets

il n'existe pas d'ArrayList de "int", "double" (les types élémentaires)

Il faut donc Envelopper les types élémentaires : int, double…



Utiliser les classes enveloppes

Wrapper classes

ArrayList d'éléments de type primitif

Depuis la version 5, Java permet la transformation automatique (Autoboxing) d'une variable d'un type primitif en un objet du type de la classe Enveloppe associée (BOXING) et l'inverse (UNBOXING).

L'utilisation des classes enveloppes devient transparente

```
int i=3;
Integer   enveloppe = i;
int x = enveloppe;
```

AUTOBOXING

UNBOXING

ArrayList d'éléments de type primitif

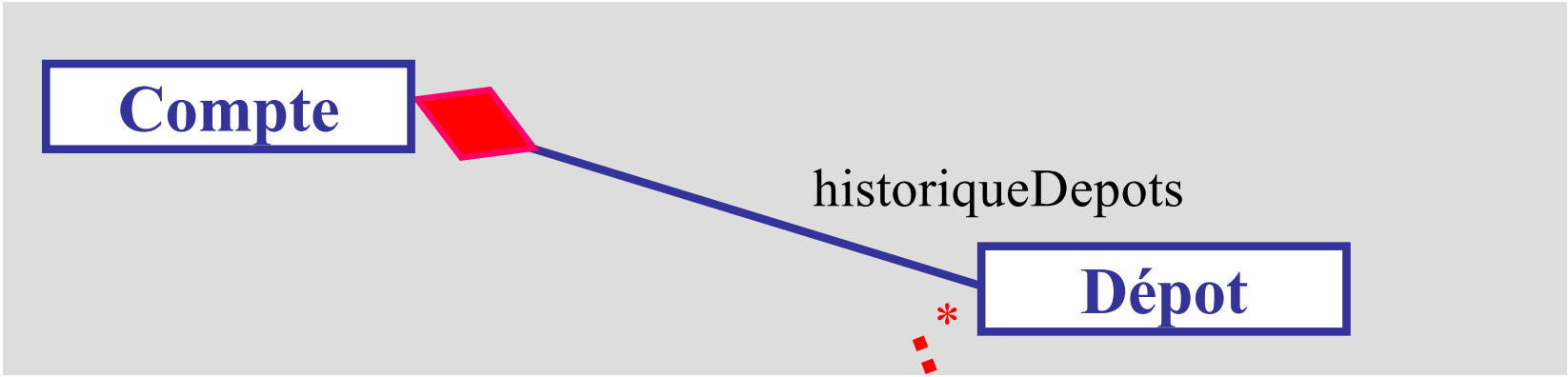
On peut créer des collections (ArrayList) de type primitifs de manière naturelle

```
ArrayList<Integer>  unArrayList =new ArrayList <Integer>( );

int x =  1;
unArrayList.add ( x );
unArrayList.add ( 10 );

int y = unArrayList.get( 0 ) ;
```

BOXING

UNBOXING

ArrayList<Double>




```
class Compte {
   …
   private ArrayList<Double> historiqueDepots;
   …
```

ArrayList paramétré

```
import    java.util.* ;
class Compte {
   private ArrayList<Double> historiqueDepots;
   public Compte()  {
       historiqueDepots =  new ArrayList<Double>();
   }
   public void deposer (double montant ) {
       historiqueDepots.add(montant)   );
   }
   public double sommeDesDepots ()  {
       double       sommeDesDepots = 0;
       for ( Double depot: historiqueDepots) {
                  sommeDesDepots = sommeDesDepots + depot;
              }
        return sommeDesDepots;
   }
}
```

Boxing

UnBoxing

Autres Collections Java

ArrayList n'est qu'un exemple de « Collection » Java

- Java propose un"Framework" de gestion des collections (regroupement d'objets)
  - Rendre l'utilisation indépendante de l'implémentation
  - Uniformiser l'interface de manipulation des ensembles d'objets
  - Augmenter l'interopérabilité entre les APIs
  - Réduire l'effort de programmation

ArrayList (avant Java 5)

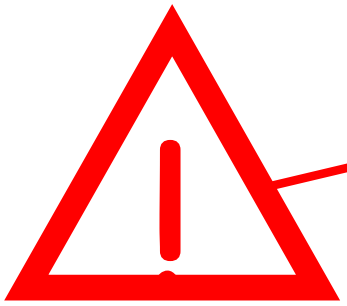
- Un même arrayList peut stocker simultanément des objets de différentes classes
  - il est donc nécessaire de typer explicitement les objets extraits d'un arrayList avant de les utiliser
    - utilisation du "cast" :

(type) get(index)

Liste d'objets indifférenciés en JAVA

```
class Client {
  String   nom;
  int age;
  public Client(…….)
  void vieillir() { … }
  ….
}
```

```
class Agence {
     String   directeur;
     String   adresse;
     ArrayList    lesClients;
     public Agence( String directeur, String adresse)
     {
          lesClients = new ArrayList();
      ...}
      void   changerDirecteur(...) { … }
      void   ajouterUnClient(...) { … }
}
```



Création de l'objet ArrayList!!

Liste d'objets indifférenciés en JAVA

```
class Agence {
      String   directeur;
      String   adresse;
      int nbClients = 0 ;
     ArrayList    lesClients;
     public Agence( String directeur, String adresse) {
            this.directeur= directeur;
            this.adresse=adresse;
            lesClients = new ArrayList(); …}

     void   ajouterUnClient(Client unClient) {
            lesClients.add(unClient);
            nbClients= lesClients.size();
      }
}
```

Nombre d'éléments du tableau

Instanciation du du ArrayList

Méthode d'ajout de références dans la liste

Liste d'objets indifférenciés en JAVA

```
class Agence {
…..
public void afficheListeNomsClients(){
  System.out.println ("Liste des noms de Clients");
  for (int i=0; i < lesClients.size() ; i++ )
     System.out.println( ((Client)lesClients.get(i)).nom);
}
```

CAST nécessaire car un ArrayList Contient des éléments de type Object