POO EN JAVA

Pr. Abdelmajid HAJAMI

BTS-DSI

2014

LA GESTION DES EXCEPTIONS

INTRODUCTION

```
int  puissance (int a, int b,){ // calcul de a^b
                int S =1;
                for (int i =0; i<b; i++)
                S*=a;
                return S;
                }
```

Si b n'est pas un entier ?!!!

Il faut donc gérer cette exception,

PREMIER EXEMPLE D'EXCEPTION

1- Comment déclencher une exception avec throw

Considérons une classe Point, munie d'un constructeur à deux arguments et d'une méthode affiche. Supposons que l'on ne souhaite manipuler que des points ayant des coordonnées non négatives.

```
class Point
{ public Point(int x, int y) throws ErrConst // peut provoquer une exception
  { if ( (x<0) || (y<0)) throw new ErrConst() ; // pour déclencher l'exception
    this.x = x ; this.y = y ;
  }
  public void affiche()
  { System.out.println ("coordonnees : " + x + " " + y) ;
  }
  private int x, y ;
}
class ErrConst extends Exception // pour l'objet fourni à throw
{ }
```

PREMIER EXEMPLE D'EXCEPTION

2-Utilisation d'un gestionnaire d'exception

```
public class Except1
{ public static void main (String args[])
  { try
    { Point a = new Point (1, 4) ;
      a.affiche() ;
      a = new Point (-3, 5) ;
      a.affiche() ;
    }
    catch (ErrConst e)
    { System.out.println ("Erreur construction ") ;
      System.exit (-1) ;
    }
  }
}
```

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<0) || (y<0)) throw new ErrConst() ;
 this.x = x ; this.y = y ;
  }
  public void affiche()
  { System.out.println ("coordonnees : " + x + " " + y) ;
  }
  private int x, y ;
}
class ErrConst extends Exception { }
```

```
coordonnees : 1 4
Erreur construction
```

GESTION DE PLUSIEURS EXCEPTIONS 1/2

On ajoute à notre classe Point une méthode deplace qui s'assure que le déplacement ne conduit pas à une coordonnée négative ; si tel est le cas, elle déclenche une exception ErrDepl (on crée donc, ici encore, une classe ErrDepl) :

```
class Point
{ public Point(int x, int y) throws ErrConst
  { if ( (x<0) || (y<0)) throw new ErrConst() ;
    this.x = x ; this.y = y ;
  }
 public void deplace (int dx, int dy) throws ErrDepl
  { if ( ((x+dx)<0) || ((y+dy)<0)) throw new ErrDepl() ;
    x += dx ; y += dy ;
  }
  public void affiche()
  { System.out.println ("coordonnees : " + x + " " + y) ;
  }
  private int x, y ;
}
```

```
class ErrConst extends Exception
{ }
class ErrDepl extends Exception
{ }
```

GESTION DE PLUSIEURS EXCEPTIONS 2/2

```
public class Except2
{ public static void main (String args[])
{ try
{ Point a = new Point (1, 4) ;
a.affiche() ;
a.deplace (-3, 5) ;
a = new Point (-3, 5) ;
a.affiche() ;
}
catch (ErrConst e)
{ System.out.println ("Erreur construction ") ;
System.exit (-1) ;
}
catch (ErrDepl e)
{ System.out.println ("Erreur deplacement ") ;
System.exit (-1) ;
}}}
```

```
coordonnees : 1 4
Erreur deplacement
```

TRANSMISSION D'INFORMATION AU GESTIONNAIRE D'EXCEPTION

1/2 Par l'objet fourni à l'instruction throw

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<0) || (y<0)) throw new ErrConst(x, y) ;
this.x = x ; this.y = y ;
}
public void affiche()
{ System.out.println ("coordonnees : " + x + " " +
y) ;
}
private int x, y ;
}
```

```
class ErrConst extends Exception
{ ErrConst (int a, int b)
{ this.a = a ; this.b = b ;
}
public int a, b ;
}
```

TRANSMISSION D'INFORMATION AU GESTIONNAIRE D'EXCEPTION

1/2 Par l'objet fourni à l'instruction throw

```
public class Exinfo1
{ public static void main (String args[])
{ try
{ Point a = new Point (1, 4) ;
a.affiche() ;
a = new Point (-3, 5) ;
a.affiche() ;
}
catch (ErrConst e)
{ System.out.println ("Erreur construction Point") ;
System.out.println (" coordonnees fournies : " + e.a + " " + e.b) ;
System.exit (-1) ;
}
}
}
```

```
coordonnees : 1 4
Erreur construction Point
coordonnees fournies : -3 5
```

TRANSMISSION D'INFORMATION AU GESTIONNAIRE D'EXCEPTION

2/2 Par le constructeur de la classe exception

```
class Point
{ public Point(int x, int y) throws ErrConst
{ if ( (x<0) || (y<0))
throw new ErrConst("Erreur construction avec coordonnees " + x + " " + y) ;
this.x = x ; this.y = y ;
}
public void affiche()
{ System.out.println ("coordonnees : " + x + " " + y) ;
}
private int x, y ;
}
```

```
class ErrConst extends Exception
{ ErrConst (String mes)
{ super(mes) ;
}
}
```

TRANSMISSION D'INFORMATION AU GESTIONNAIRE D'EXCEPTION

2/2 Par le constructeur de la classe exception

```
public class Exinfo2
{ public static void main (String args[])
{ try
{ Point a = new Point (1, 4) ;
a.affiche() ;
a = new Point (-3, 5) ;
a.affiche() ;
}
catch (ErrConst e)
{ System.out.println (e.getMessage()) ;
System.exit (-1) ;
}
}
}
```

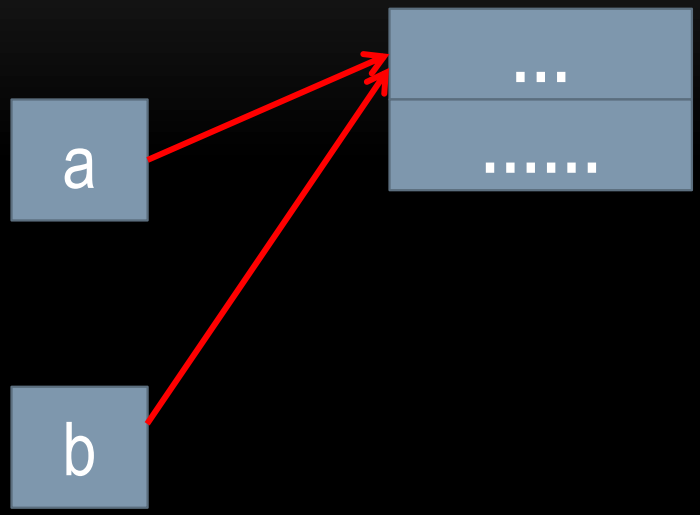
```
coordonnees : 1 4
Erreur construction avec coordonnees -3 5
```

CLONER EN JAVA

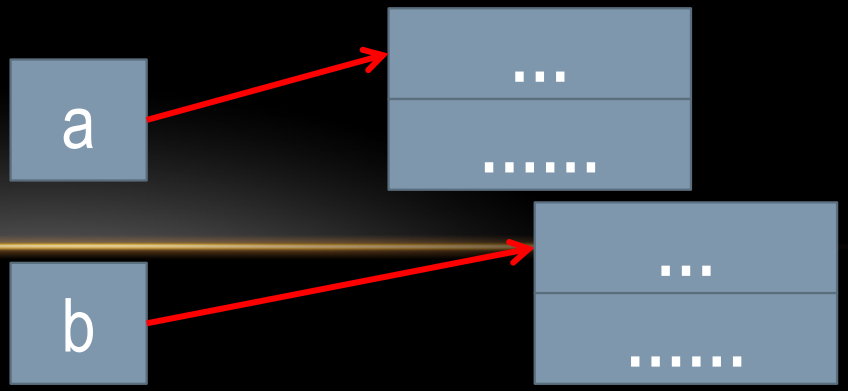
CLONER EN JAVA

Introduction

```
classA  a;
a = new classA();
classA  b;
b = a;
```



Dans la programmation orientée objet, il arrive que l'on doive cloner un objet. Le "clonage" d'un objet pourrait se définir comme la création d'une copie par valeur de cet objet.



CLONER EN JAVA

L'interface Cloneable

- Pour pouvoir être clonée, une classe doit implémenter l'interface *Cloneable*.
- Celle-ci indique que la classe peut réécrire la méthode clone() héritée de la classe *Object*.

Par convention, les classes implémentant l'interface *Cloneable* doivent réécrire la méthode *Object*.clone()

CLONER EN JAVA

La méthode clone()

La méthode clone() doit retourner une copie de l'objet que l'on veut cloner. Cette copie dépend de la classe de l'objet

Par convention, l'objet retourné doit être indépendant de l'objet cloné, c'est à dire que tous les attributs devront être eux aussi clonés.

CLONER EN JAVA

Exemple


```
class Point implements Cloneable{
public Point(int a, int b){ x=a; y=b; }
public Point clone(){
Point point = null;
try{
point = (Point) super.clone();
}
catch(CloneNotSupportedException e) {
      System.out.println (e.getMessage()) ;
     System.exit (-1) ;
    }
return point;
}
public void deplace(int dx, int dy){  x+= dx; y+= dy; }
public void affiche(){
System.out.println("je suis le point de coordonnées x= "+x+" et y= "+y);
}
private int x;
private int y;}
```

CLONER EN JAVA

Exemple

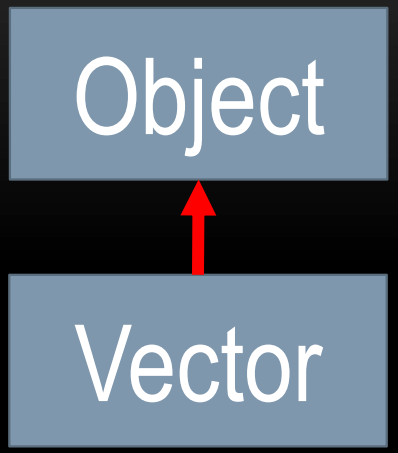
```
public class Cloner{ public static void main(String [] args){
Point a;
Point b;
Point c;
b =new Point(0,0);
a= new Point(3,8);
a.affiche();
a.deplace(3, 8);
b=a.clone();
c=a;
b.affiche();
c.affiche();
System.out.println(a.toString());
System.out.println(b.toString());
System.out.println(c.toString());
}}
```

```
je suis le point de coordonnées x= 3 et y= 8
je suis le point de coordonnées x= 6 et y= 16
je suis le point de coordonnées x= 6 et y= 16
Point@dc6a77
Point@d1e89e
Point@dc6a77
```

LA CLASSE VECTOR
LA CLASSE ARRAYLIST

LA CLASSE VECTOR

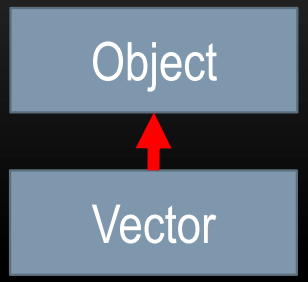
La classe Vector permet de gérer des listes.




```
//Attributs :
 int capacityIncrement
        // capacité ajoutée automatiquement au vecteur lorsqu'il est saturé.
        //Si cette valeur est nulle, la capacité est doublée en cas de besoin.
int elementCount
         // nombre d'éléments du vecteur
Object elementData []
         // tableau des éléments du vecteur
```

LA CLASSE VECTOR

Quelques méthodes :



Vector()

// Crée un vecteur vide

Vector(int initialCapacity)

// Crée un vecteur vide dont la capacité initiale est égale à l'argument
//spécifié.

Vector(int initialCapacity, int capacityIncrement)

// Crée un vecteur vide dont la capacité est égale à initialCapacity, et
//dont la capacité s'accroît de capacityIncrement lorsqu'il est saturé.

void **addElement(Object obj)**

// ajoute l'élément obj au vecteur

Object **elementAt(int index)**

// renvoie l'objet d'indice index.

Object **firstElement()**

// renvoie le premier élément du vecteur.

LA CLASSE VECTOR

Quelques méthodes :

void **insertElementAt(Object obj, int index)**

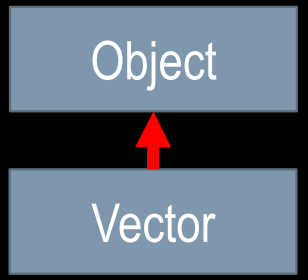
// insère l'objet obj de façon qu'il occupe l'indice index, les objets situés //à la suite étant décalés.

boolean **isEmpty()**

// renvoie true si le vecteur est vide, sinon false.

Object **lastElement()**

// renvoie le dernier élément du vecteur.



void **removeAllElement()**

// supprime tous les éléments du vecteur et remet à 0 le nombre de ses //éléments.

boolean **removeElement(Object obj)**

// supprime la première occurrence de l'objet obj et avance d'un indice //tous les éléments situés derrière.

// renvoie true si l'objet existe, sinon false.

LA CLASSE VECTOR

Quelques méthodes :

void **removeElementAt(int index)**

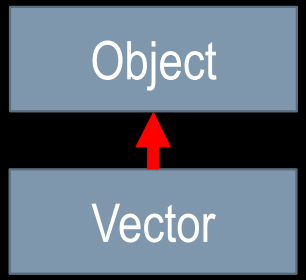
// Supprime l'élément d'indice index et avance d'un indice tous les //éléments situés derrière.

void **setElementAt(Object obj, int index)**

// remplace l'élément d'indice index par l'objet obj.

int **size()**

// donne le nombre d'éléments du vecteur



LA CLASSE VECTOR

Exemple

```
public class LISTE{
public static void main(String [] args){
Vector<Point>  Liste;
Liste = new Vector<Point>(5);
Point a,  b,  c;
a =new Point(9,-5);
b= new Point(3,8);
c =new Point(-9,-25);
System.out.println("La taille de la liste = "+Liste.size());
Liste.add(a);
Liste.add(b);
Liste.add(d);
System.out.println("La taille de la liste = "+Liste.size());
a.deplace(3, 8);
for (int i=0; i<Liste.size(); i++){
System.out.println(Liste.elementAt(i).toString()) ;
Liste.elementAt(i).affiche();
}}}
```

LA CLASSE VECTOR

Exemple

```
public class LISTE{
public static void main(String [] args){
Vector<Point>  Liste;
Liste = new Vector<Point>(5);
Point a,  b,  c;
a =new Point(9,-5);
b= new Point(3,8);
c =new Point(-9,-25);
System.out.println("La taille de
Liste.add(a);
Liste.add(b);
Liste.add(d);
System.out.println("La taille de
a.deplace(3, 8);
for (int i=0; i<Liste.size(); i++){
System.out.println(Liste.elementAt(i).toString()) ;
Liste.elementAt(i).affiche();
}}}
```

```
La taille de la liste = 0
La taille de la liste = 3
Point@d1e89e
je suis le point de coordonnées x= 6 et y= 16
Point@ff057f
je suis le point de coordonnées x= 9 et y= -5
Point@c1f10e
je suis le point de coordonnées x= -9 et y= -25
```

LA CLASSE ARRAYLIST


```
ArrayList<Point>  Liste;
Liste = new ArrayList<Point>(10);
Liste.add(a);
Liste.add(b);
for (int i=0; i<Liste.size(); i++){
System.out.println(Liste.get(i).toString()) ;
Liste.get(i).affiche();
```

LA CLASSE ARRAYLIST

Création d'un ArrayList

il est possible d'indiquer la taille initiale dans le constructeur

Il y a 2 constructeurs :

ArrayList()

ArrayList(int initialCapacity)

Modification d'éléments

Il y a deux manières d'ajouter un élément :

à la fin d'un ArrayList avec la méthode

boolean add(Object newElement)

à une position donnée

void add(int index, Object newElement)
throws IndexOutOfBoundsException

le paramètre index indique où insérer le nouvel élément si position incorrecte, une exception est lancée.

LA CLASSE ARRAYLIST

Modification d'éléments

pour remplacer un objet à une position donnée

Object set(int index, Object newElement)
throws IndexOutOfBoundsException

cette méthode fonctionne comme void add(int index, Object newElement) sauf que l'élément à la position index est remplacé

Accès aux Éléments

pour accéder à un élément il n'y a pas d'indexation comme pour les tableaux il faut utiliser la méthode spécialisée

Object get(int index) throws IndexOutOfBoundsException

exemple :

```
ArrayList aList = new ArrayList();
aList.add(new PacMan());
aList[0].display(); //  interdit !
aList.get(0).display(); // ok
```

LA CLASSE ARRAYLIST

A ccès aux Eléments

pour tester le contenu, il existe la méthode

boolean is Empty()

pour connaître le nombre d' éléments dans la liste, il faut utiliser la méthode : **int size()**

exemple :

```
if (!aList.isEmpty()) {
  for  (int i=0; i < aList.size( ); i++){
 System.out.println( aList.get( i));
}
}
```

LA CLASSE ARRAYLIST

Suppression d'éléments

Pour supprimer un élément à une position donnée, il faut utiliser la méthode

Object remove(int index)
throws IndexOutOfBoundsException

Recherche d'éléments

pour savoir si un objet est présent ou non dans une liste, il faut utiliser la méthode

boolean contains(Object obj).

pour connaître la position d'un élément dans une liste, on peut utiliser deux méthodes

pour avoir la *première occurrence , il faut utiliser*

int indexOf(Object obj)

pour avoir la *dernière occurrence , il faut utiliser*

int lastIndexOf(Object obj)

PHASE D'IMPLÉMENTATION

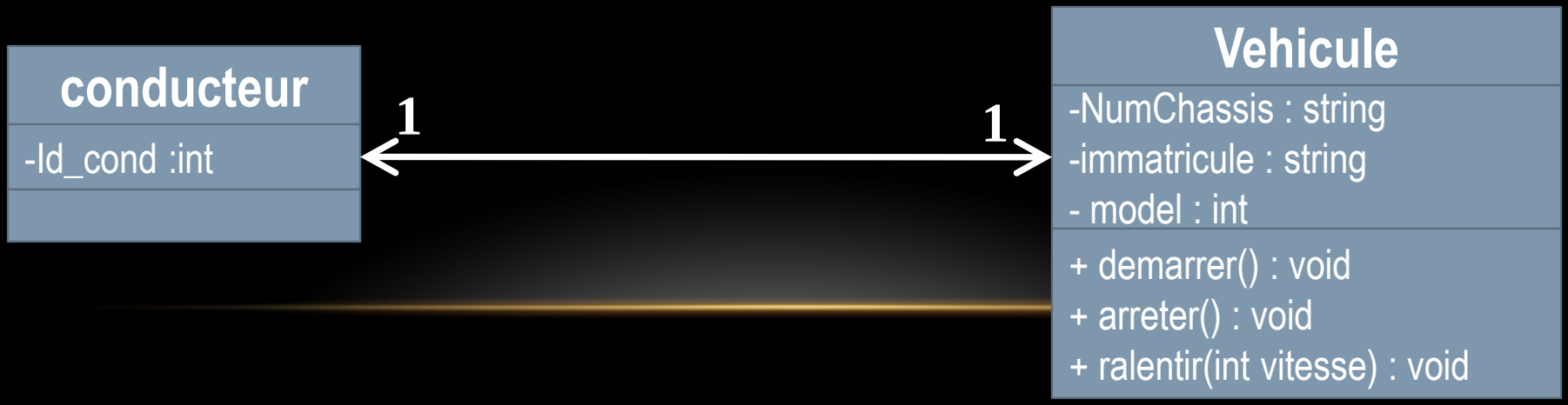
Implémentation d'une association 1 à 1

Il faut voir le type de cette association :

bidirectionnelle : les deux objets se connaissent

unidirectionnelle : un seul des objets qui reconnait l'autre

Dans le cas d'une association bidirectionnelle chacun des deux objets doit contenir une référence sur l'autre objet



PHASE D'IMPLÉMENTATION

Implémentation d'une association 1 à 1 bidirectionnelle 1/2

```
public abstract class Vehicule {
        private   string     numChassis ;
        private    string    immatricule ;
        private   int        model ;
       private    Conduteur          cond;
       // getter /  setter
       public     void       addConducteur(Conducteur c){
         if (c != null){
            if (c.getVehicule() != null) {
                    // si le conducteur c est déjà associé à un autre véhicule
                    c.getVehicule().setConducteur(null);
                    // cet autre véhicule doit se dissocier
                    }
           this.setConducteur(c);
           c.setVehicule(this);
              }
      }
}
```

PHASE D'IMPLÉMENTATION

Implémentation d'une association 1 à 1 bidirectionnelle 2/2

```
public class Conducteur {
        private int id_cond;
        private   Vehicule          v;
        // getter /  setter
        public    void      addVehicule(Vehicule v){
          if (v != null){
             if (v.getCondusteur() != null) {
                    // si le véhicule v est déjà associé à un autre conducteur
                    v.getConducteur().setVehicule(null);
                    // cet autre conducteur doit se dissocier
                    }
              this.setVehicule(v);
              v.setConducteur(this);
                 }
        }
}
```

PHASE D'IMPLÉMENTATION

Implémentation d'une association 1 à 1 unidirectionnelle 1/2

Dans le cas d'une association unidirectionnelle , l'objet émetteur doit connaitre celui récepteur.

Sur cet exemple, la classe véhicule doit contenir une référence vers la classe conducteur.

PHASE D'IMPLÉMENTATION

Implémentation d'une association 1 à 1 unidirectionnelle 2/2

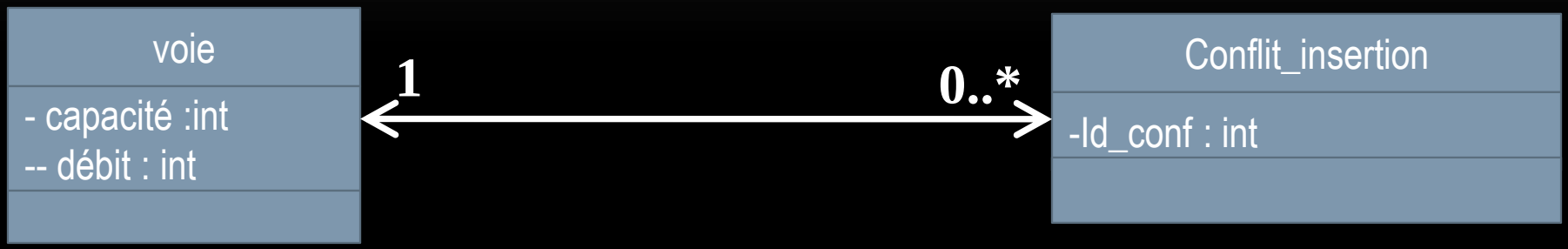
```
public abstract class Vehicule {
        private          string   numChassis ;
        private string   immatricule ;
        private          int      model ;
        private Conduteur         cond;

        // getter /  setter

        public   void     addConducteur(Conducteur c) {
          if ( c != null ) {
                    this.setConducteur(c);
              }
          }
}
```

PHASE D'IMPLÉMENTATION

Implémentation d'une association 1 à plusieurs bidirectionnelle 1/3



L'objet ayant la cardinalité la plus élevée reçoit une référence sur l'objet ayant la cardinalité la plus faible en respectant le sens de l'association

La cardinalité 0..* est traduite par une collection d'objets de types conflit_insertion dans l'objet voie

Dans le cas où le degré de multiplicité est connu (n), la collection peut être remplacée par un tableau de dimension n

PHASE D'IMPLÉMENTATION

Implémentation d'une association 1 à plusieurs bidirectionnelle 2/3

```
public class Conflit_insertion {
        private int id_conf;
        private Voie     voie;
        // getter /  setter

        public   void     addVoie(Voie v){
          if (v != null){
             if (! v.getConflits().contains(this)) {
                   if (voie != null) voie.remove(this);
                   this.setVoie(v)
                   voie.getConflits().add(this);
                   }
           }
}
```

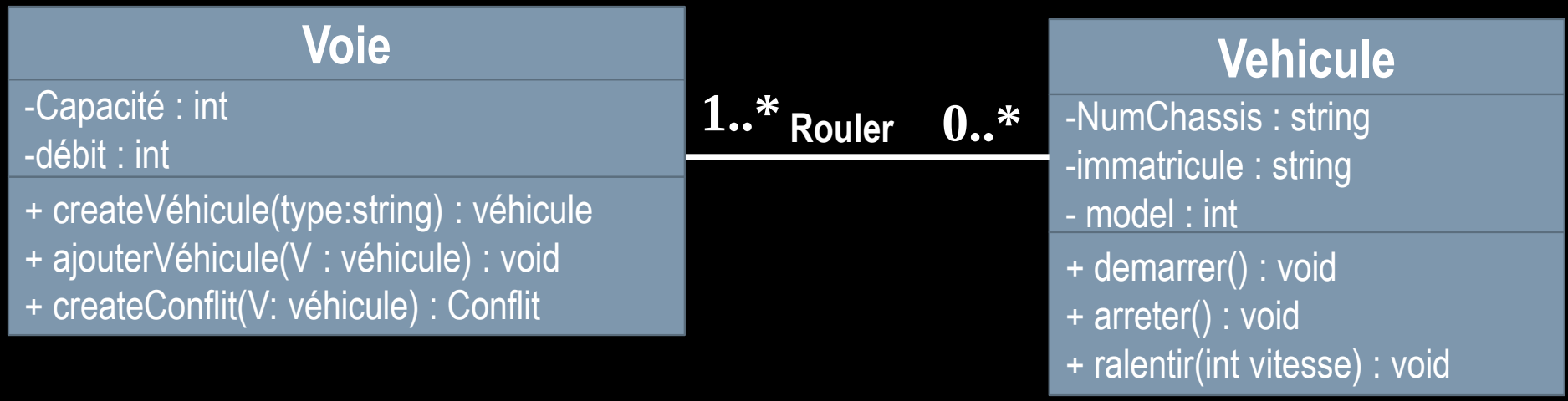
PHASE D'IMPLÉMENTATION

Implémentation d'une association 1 à plusieurs bidirectionnelle 3/3

```
public class Voie {
        private int capacite;
        private   ArrayList<Conflit_insertion> conflits;
        // getter /  setter
        public Voie() {
                conflits = new ArrayList<Conflit_insertion>();
                }
        public    void      addConflit(Conflit_insertion c) {
          if (c != null){
             if (! conflits().contains(c)) {
                if (c.getVoie != null) c.getVoie.remove(c);
                this.conflits.add(c);
                 c.setVoie(this);
                }
             }
        }
}
```

PHASE D'IMPLÉMENTATION

Implémentation d'une association plusieurs à plusieurs 1/2



Dans le cas d'une association bidirectionnelle chacune des deux classes doit contenir une collection d'objets de l'autre classe

PHASE D'IMPLÉMENTATION

Implémentation d'une association plusieurs à plusieurs 2/2

```
public abstract class Vehicule {

   private   string   numChassis ;
   private string     immatricule ;
   private            int          model ;
   privateVector<Voie>  voies;

            // getter /  setter

  public  void       demarer(){}
  public  void       arreter() {}
  public  abstract  void       ralentir(int  v) {}

}
```

```
public class Voie {
            private int capacite;
            private    int debit;
            private    vector<Vehicule> vehicules;

            // getter /  setter

  public Vehicule   createVehicule(String type){}
  public  void      ajouterVehicule(Vehicule v)
{}
  public Conflit    createConflit(Vehicule v){}
}
```