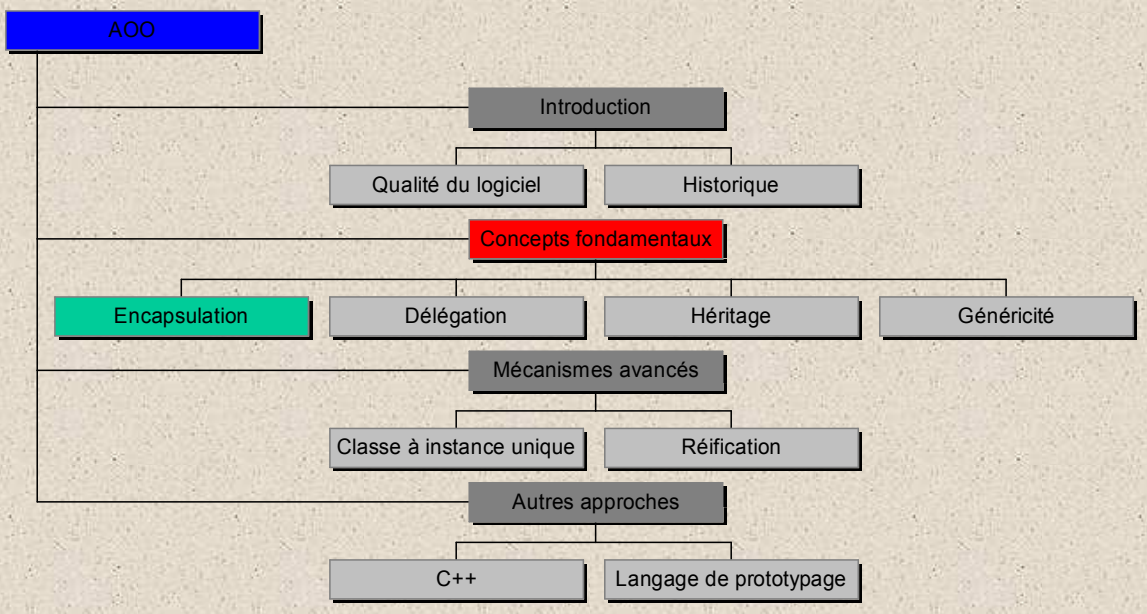
L'ENCAPSULATION

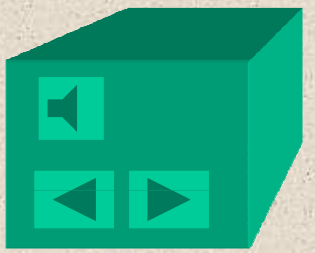
Sommaire



Modularité : Les deux visions

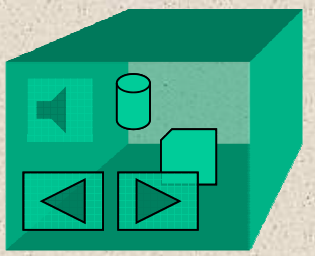
Le module constitue une entité logicielle indépendante

Il est appréhendable de deux façons :



- l'approche externe (interface)
  - permet de manipuler le module au travers de l'interface proposé par lui.
  - On fait appel à des services offerts par le module
  - Les services sont de nature procédurale
    - les structures de données ne sont pas des services
- l'approche interne (réalisation)
  - permet de manipuler le module au travers de sa réalisation
  - On accède à la description des services rendus par le module.

On dit que le module encapsule les données qu'il manipule.

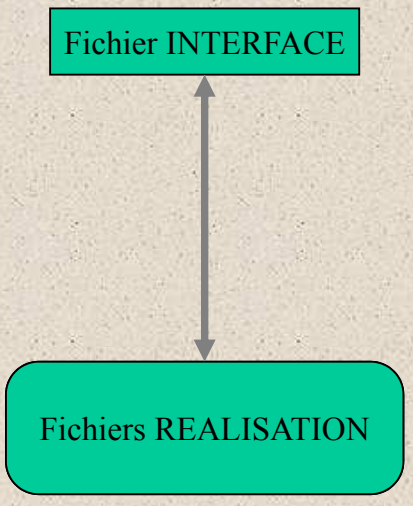


- Seules les procédures du module sont habilitées à modifier les données encapsulées.
  - Accès non dispersé aux données
  - flexibilité accrue

La classe

C'est l'unité modulaire dans l'approche objet

- Elle décrit à la fois l'interface et la réalisation du module.
  - Dans certains langages la séparation peut exister
    - ce qui permet d'avoir plusieurs réalisations d'une même intrerface et donne une certaine souplesse.
    - Qui peut introduire des confusions en fonction de quand et par qui est faite l'association entre l'interface et une réalisation.
  - Certains associent la notion de "TYPE" à l'interface et la notion de "CLASSE" à la réalisation.

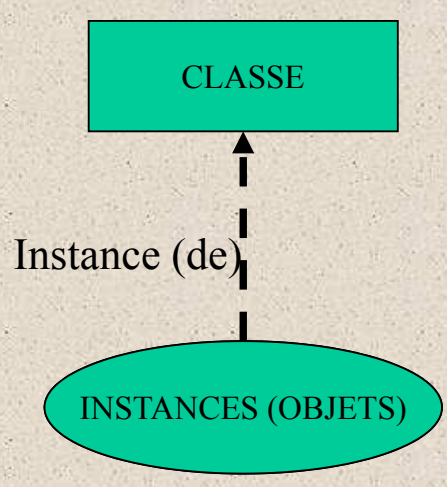


La classe

C'est un modèle

– Elle décrit la structure d'une **famille** d'objets que l'on nomme les **instances** de cette classe.

– Tout objet **est instance** d'une classe.



La classe

Elle est constituée de descriptions de :

– données : que l'on nomme **attributs**

– procédures : que l'on nomme **méthodes**

– on utilisera indistinctement les termes suivants pour désigner attributs et méthodes (uniformité)

- **compétence**, **caractéristique**, ….
- certains langages ne permettent pas cette abstraction, ce qui introduit au niveau de l'interface une information sur la réalisation.
  – Par exemple pour une représentation polaire, la norme est un attribut ; pour une représentation cartésiennne, la norme est une méthode.
  – La plupart du temps des modèles sont proposés pour palier à des inconvénients comme celui-ci.
  » Bean avec la notion de propriété
  » Java avec son concept d'interface.

Structure d'une Classe

class NOM

- Portée
  - accessibilité de la classe
- Indexation
  - ensemble de couples attribut-valeur permettant de qualifier la classe pour des manipulations globales
- Héritage
  - caractérisation des liens avec d'autres classes
- Instanciation
  - caractérisation des mécanismes de création d'instances
- Compétence
  - liste des caractéristiques spécifiques de la classe
- Invariant
  - propriétés que tout objet (instance) de cette classe doit respecter

INDICE

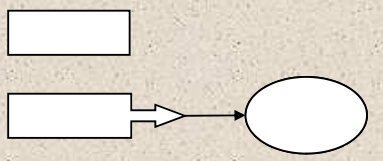
```
/**
* décrit les entiers compris dans une plage de valeurs
[binf..bsup].
*   La plage de valeurs est fixée pour chaque indice au
moment de sa création.
*   La valeur initiale de l'objet indice est égale à la
borne inférieure de sa plage de
*   valeurs.
* @author  P.Morat ou http://imag.fr/Philippe.Morat ...
* @version 1.0
* date : 1/9/99
* @invariant Cohérent : binf ≤ valeur ≤ bsup
* @motcle  indice, rang, intervalle
* @see <a href="Indice.java">Indice</a>
*/
<portée> class Indice {

}
```


Les attributs

Une entité est soit :

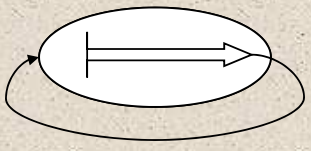
– un attribut qui contient :

- une valeur de type primitif (int, char, boolean, etc)
- une référence à un objet d'un type non primitif



– une référence particulière

- this qui désigne l'objet courant (self, current)
- super (voir chapitre héritage)



Une référence peut posséder la valeur null

– Aucun objet n'est accessible par cette référence

Il existe des notations pour les valeurs des types primitifs.

Les méthodes

Une méthode réalise soit une action , soit une fonction qui possède alors au moins une instruction return.

```
/**
* <description informelle de la méthode>
* @param <description du paramètre>
* @require <Nom> : <Expression>
* @ensure <Nom> : <Expression>
*/

<TYPE ou void> NOM ( {PARAMETRES} ) {

}
```


INDICE

```
class Indice {
   /**
   * création d'un indice dans l'intervalle [bI..bS]
   */
   <portée> Indice(int bI, int bS) {
   }
   /**
   * fait progresser l'indice d'une unité s'il n'est pas sur
bsup
   */
    <portée> void succ() {
   }
   /**
   * affecte l'indice avec la valeur de l'argument e. Celle-
ci doit respecter
   *   les bornes
   */
    <portée> void set(int e) {
   }
   /**
   * restitue la valeur courante de l'indice
   */
    <portée> int get() {
   }
   // Valeur et bornes de l'indice.
    <portée> int valeur, binf, bsup;
}
```


Structure de l'Objet

C'est une instance d'une classe

– Il est constitué d'une partie "Statique" et d'une partie "Dynamique"

Partie Dynamique :

Elle ne varie pas d'une instance de classe à une autre

Il n'y a qu'un seul exemplaire pour l'ensemble des instances d'une classe

Elle permet d'activer l'objet

Elle est constituée des méthodes de la classe

Partie Statique :

Elle varie d'une instance de classe à une autre

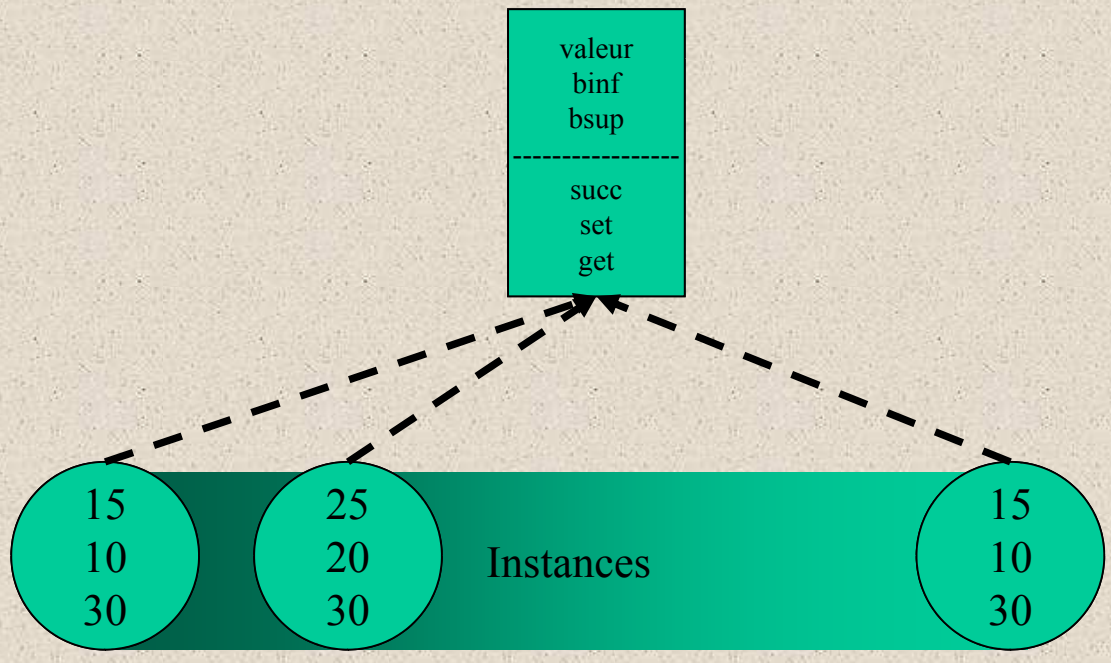
Elle varie durant la vie d'un objet

Elle est constituée d'un exemplaire de chaque attribut de la classe.

PARTIE DYNAMIQUE
ensemble des méthodes associées à l'objet permettant de l'activer

PARTIE STATIQUE
ensemble des attributs possédés par l'objet permettant de caractériser son état

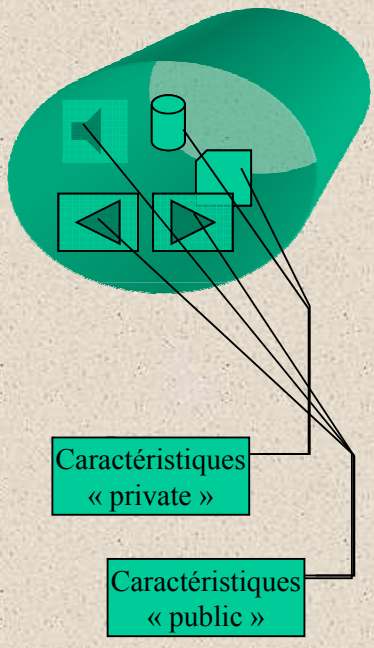
Classe & Objets



Portée d'une caractéristique

L'objet peut être considérée comme une "boite noire"

Cette boite noire dispose de "boutons" externes permettant de l'activer (communiquer, ordonner, …). Ces pattes sont les services accessibles depuis l'extérieur.



la portée d'une caractéristique (on détaillera plus dans le chapitre héritage)

- public : toute classe, caractéristiques publiques
- private : aucune classe, caractéristiques privées,
- protected : caractéristiques exportées uniquement à la descendance

INDICE

```
public class Indice {
   /**
    * création d'un indice dans l'intervalle [bI..bS]
    */
    public Indice(int bI, int bS) {
   }
   /**
    * fait progresser l'indice d'une unité s'il n'est pas sur
bsup
    */
    public void succ() {
   }
   /**
    * affecte l'indice avec la valeur de l'argument e. Celle-
ci doit respecter
    *   les bornes
    */
    public void set(int e) {
   }
   /**
    * restitue la valeur courante de l'indice
    */
    public int get() {
   }
   // Valeur et bornes de l'indice.
    private int valeur, binf, bsup;
}
```


Constructeur d'objets

L'instanciation permet de créer un objet, instance d'une classe.

- Obtention de l'espace nécessaire à la partie Statique de l'objet (mémoire)
- Initialisation de cette mémoire

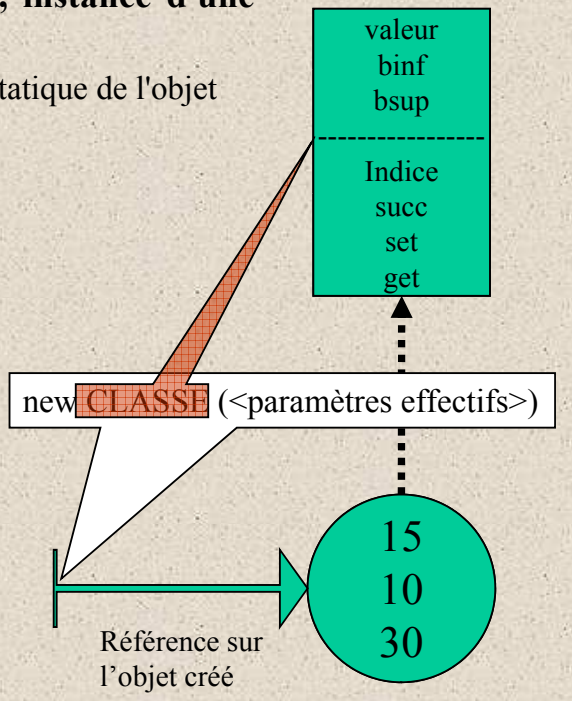
Il existe un constructeur par défaut

- réduit au 1er point
- inexistant si un autre constructeur existe

Gestion mémoire

Libère le programmeur de la gestion de l'espace mémoire

Garantie une restitution correcte



L'envoi de message

Seul mécanisme de communication entre objets

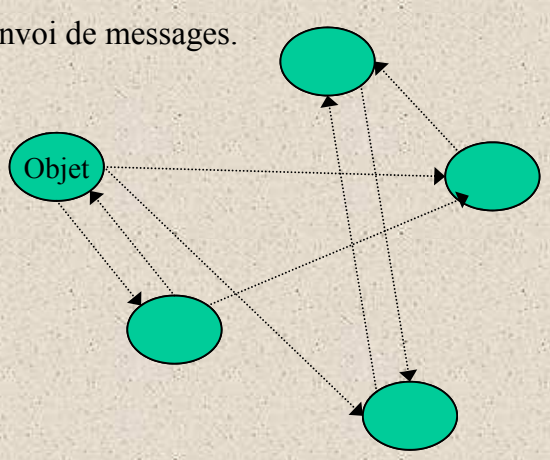
- Les objets constituent un réseau de communication
- Ils s'activent sur ce réseau par le mécanisme d'envoi de messages.

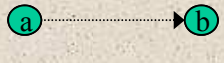
Le message est constitué de :

- un destinataire
  - valeur d'objet
  - référence d'objet
- un nom de méthode
- des arguments complémentaires

Le réseau n'est pas statique

- il est constitué dynamiquement
- il évolue en cours de l'exécution
- l'envoi de message peut être réaliser de manière
  - procédurale
  - distribué
  - …



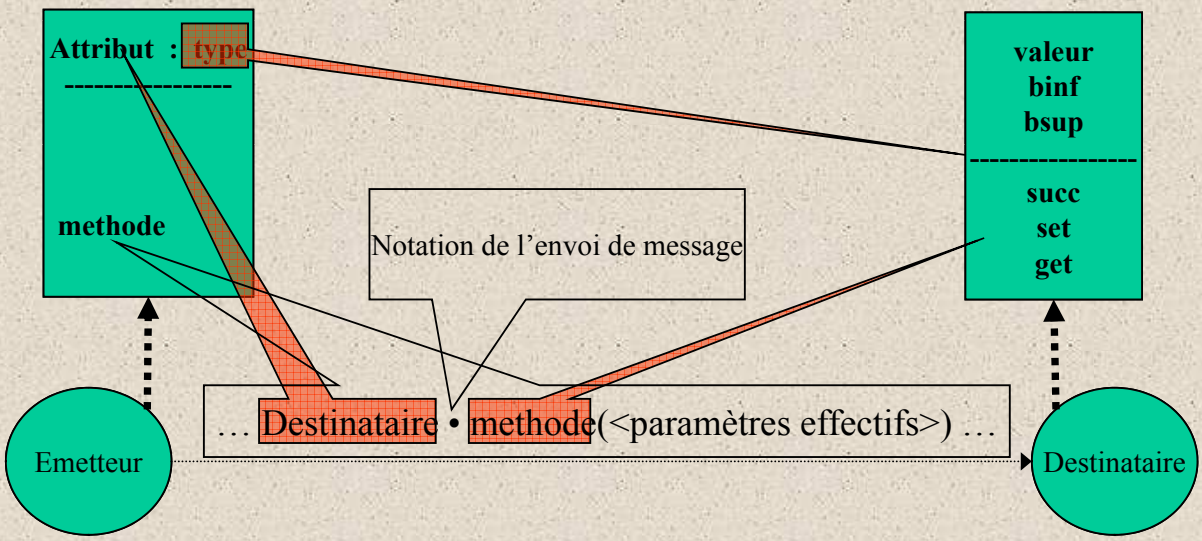


Existence d'un support potentiel de communication pour l'envoi de messages de a vers b

Classes, Objets et envoi de Message

La communication est réduite à l'interface proposée par le destinataire

- Toute instance doit répondre aux messages que son modèle (classe) propose



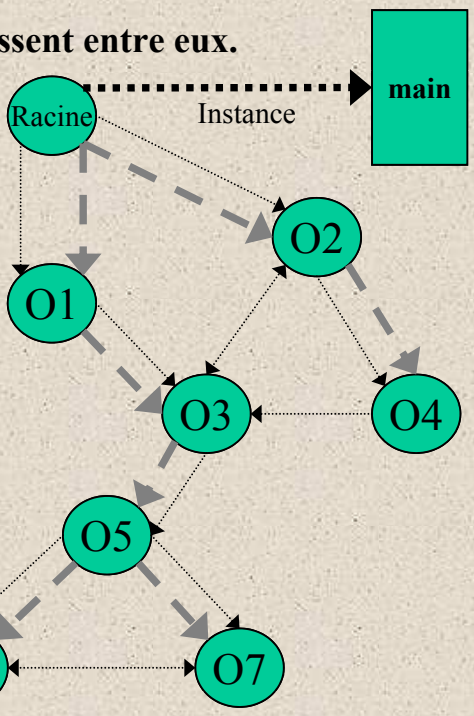
Application à Objet

Une application est faite d'objets qui interagissent entre eux.

Un objet particulier est désigné comme la racine de l'application

Cet objet est instance unique d'une classe modélisant l'application. Cette classe définit une méthode correspondant à l'exécution de l'application

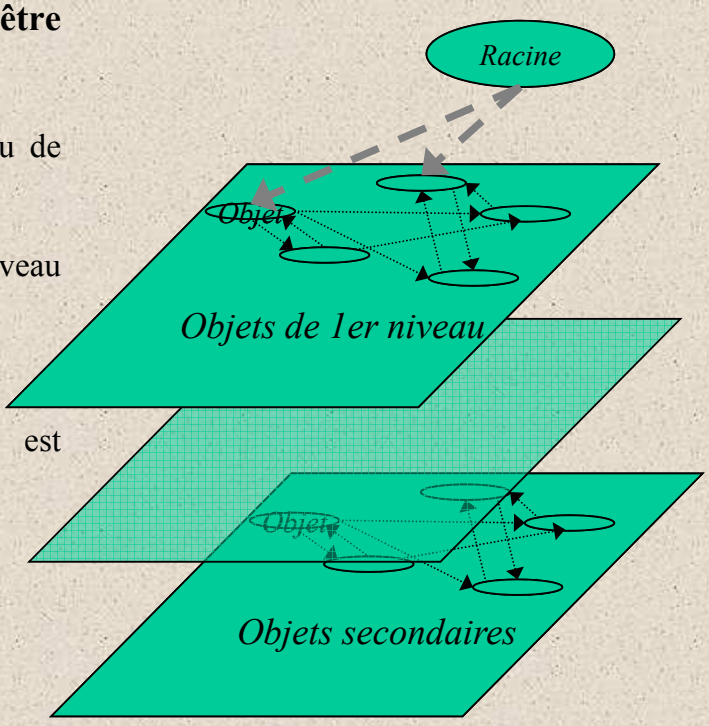
Chaque objet est créé dynamiquement à l'exécution à partir de la racine.



Application à Objet (hiérarchie)

L'ensemble des objets peut être structuré hiérarchiquement

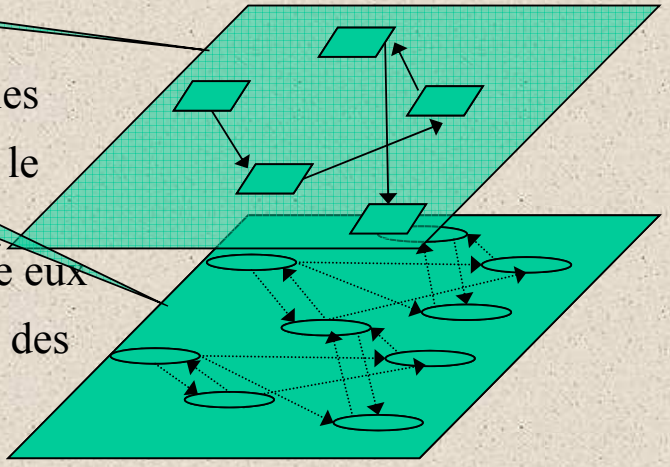
- Chaque objet appartient au niveau de compétence qui est le sien
- Chaque couche caractérise un niveau d'abstraction
- La racine de l'application est élémentaire



Relation Classe/Objet

L'espace de modélisation contient les classes servant à décrire les concepts de l'application

L'espace d'exécution contient les objets qui assurent le comportement de l'application.



Les objets communiquent entre eux

Les classes dépendent les unes des autres

La puissance du modèle objet tient à ce que la relation n'est pas de un pour un entre l'espace de modélisation (CLASSE) et celui de l'exécution (OBJET).

Constructions primitives du langage

Déclarations

- <Type> <Nom>;
- <Type> <Nom>(<Paramètres formels>) { <corps>};

Instructions

- Création d'objet
  - "new <classe>(<parametres effectifs>)"
- affectation
  - "="
- appel de méthode
  - "<destinataire>•<méthode >(<parametres effectifs>)"
    - le destinataire peut être absent, dans ce cas la méthode s'applique à l'objet courant. On peut aussi utiliser this comme destinataire.

Constructions primitives du langage

Structures de contrôle

- Conditionnelle
  - if else
  - switch
- Itérateur
  - while, do-while
  - for
  - «étiquette»
  - break, continue
  - return

Constructions primitives du langage

Les expressions

- [] . () «expr»++ «expr»--
- ++«expr» --«expr» + - ~ !
- new«type»(«expr»)
- * / %
- << >> >>>
- < > <= >= instanceof
- == !=
- &
- ^
- |
- &&
- ||
- ?:
- = += -= *= /= %= >>= <<= >>>= &= ^= |=

Classe INDICE

```
public class Indice {
   /**
   *  création d'un indice dans l'intervalle [bI..bS]
   */
   public Indice(int bI, int bS) {binf=bI; bsup=bS; valeur=binf;
   }
   /**
   * fait progresser l'indice d'une unité s'il n'est pas
sur bsup
   */
    public void succ {if(valeur <bsup) valeur++;
   }
   /**
   * affecte l'indice avec la valeur de l'argument e.
Celle-ci doit respecter
   *  les bornes
   */
    public void set(int e) {if(binf <= e && e <= bsup) valeur=e;
   }
   /**
   * restitue la valeur courante de l'indice
   */
    public int get() {return valeur;
   }
   // Valeur et bornes de l'indice.
    private int valeur, binf, bsup;
}
```


La surcharge

La détermination de l'opération associée à un envoi de message se fait en fonction du :

- nom de la méthode
- type du destinataire

Soit C1 une classe ayant une méthode M ayant une sémantique M_{C1}
Soit C2 une classe ayant une méthode M ayant une sémantique M_{C2}

C1 V1; // V1 est un attribut de type C1
C2 V2; // V2 est un attribut de type C2

V1•M(); // exécute la sémantique M_{C1}
V2•M(); // exécute la sémantique M_{C2}

La surcharge

Une méthode de nom M peut avoir plusieurs sémantiques au sein d'une même classe. Celles-ci se distinguent par le nombre et les natures de leurs paramètres.

- La sémantique d'une méthode dépend donc de :
  - Nom de la méthode
  - Type du destinataire
  - Nombre et types des arguments
- Attention : Le type d'une fonction n'est pas une caractéristique discriminante.
- Exemples :
  - int X;
  - void X() {…};
  - void X(int …, …) {…};
  - int X() {…};

INDICE

```
public class Indice {
    /**
    * création d'un indice dans l'intervalle [binf..bsup]
    */
    public Indice(int binf, int bsup) {
       this.binf=binf; this.bsup=bsup; valeur=binf;
    }

    /**
    * restitue la valeur courante de l'indice
    */
    public int valeur() {return valeur;}
    /**
    * affecte l'indice avec la valeur de l'argument e. Celle-ci
doit respecter
    *   les bornes
    */
    public void set(int e) {if(binf <= e && e <= bsup) valeur=e;}
    /**
    * affecte l'indice avec la valeur de l'indice argument.
    * celle-ci doit respecter les bornes
    */
    public  void set(Indice i) {set (i.valeur());}
    /**
    * restitue la représentation textuel de l'objet
    */
    public String  toString() {return '['+binf+'/'+valeur+'/'+bsup+']';}
    // Valeur et bornes de l'indice.
     private int valeur, binf, bsup;
}
```

On lève l'ambiguïté en utilisant le sélecteur d'objet this

On lève l'ambiguïté en utilisant la surcharge

Méthode usuelle délivrant une représentation textuelle de l'objet