Programmation Impérative 2

Responsable: Nguyen Kim Thang
thang@ibisc.fr

Chargés de TDs: Francesco Belardinelli, Arnaud Fouchet, NKT

http://www.ibisc.fr/~thang/Teaching/ProgImp2/progImp.html

Résumé du cours précédent

- Instructions de contrôles:

  if/then switch ... case

- Boucles/Récursivité

  while/for

- Découpage fonctionelle

  factoriser des parties

Résumé du cours

- Classes et objets simples
- Découpage structurelle et fonctionelle
- Modularité fonctionelle, robustess et maintenance

Example 1: Calculatrice

Disposer de boutons tels que

- Ils possèdent
  une légende, une couleur,
  des coordonnées ...
- Ils sont différents:
  leur légende, leur couleur, ...

Boutons: caractéristiques communes; chacun a ses propre valeur

Il en va de même pour tout ce qu'on a envie de nommer des objets au sens large

Example 2: Enfants

String nom

boolean estGarçon

double poids

- Primitif (int; double; ...)
- Objets (String; ...)

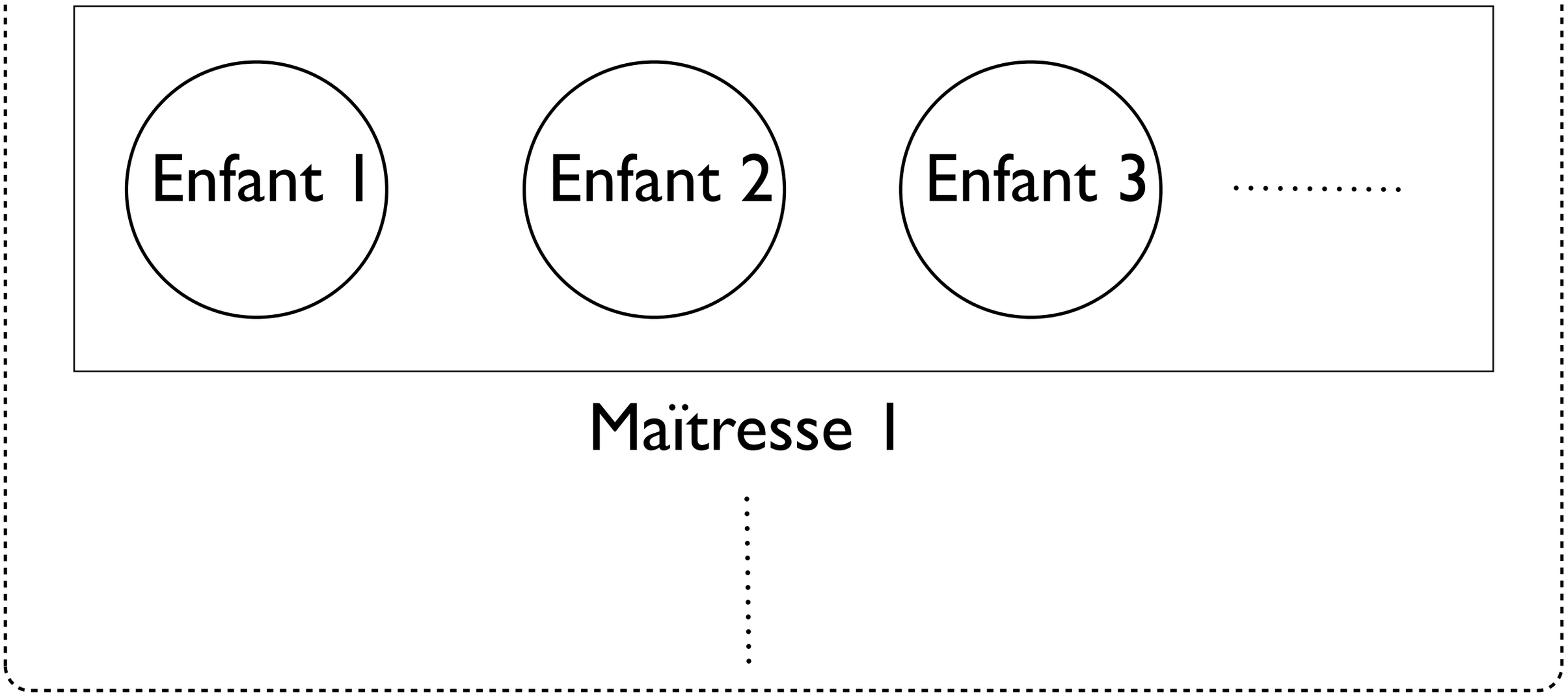
Pourquoi pas juste primitifs?

String Alex String David String David
double 20 kg double 25 kg double 19 kg

Avec 5000 enfants ?!

David2

Example 3: Ecoles



caractéristiques communes; propre valeur

Classes & Objets

Descripteur des attributs des objets

Instances particulières valuées

Un objet = une instance de classe

Bouton	1; 2; 3; ...
Enfant	(David, 20kg); (Antoine, 19kg); ...
Maïtresse	Claire; Marie; ...
Ecole	Evry; Palaiseau; ...

Classes & Objets

```
class X {
```

attributs (variables d'instance)

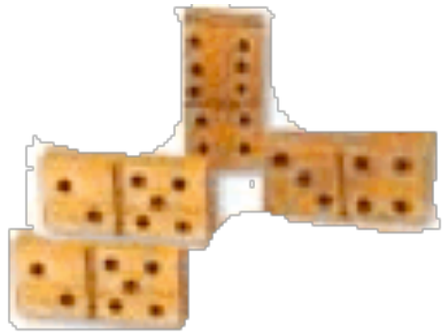
constructeurs

```
}
```

- La manière que les objets doivent être conçus

Dominos

Descripteurs des attributs des objets



Instances particulières valuées

```
/** Classe représentant un domino
  * avec ses 2 marques.
  */
class Domino {
   // Attributs communs
   // à tous les objets
   // (données propres)
   int  m1; // marque 1
   int  m2; // marque 2

   // ...à suivre...
}
```

Déclarations non statique

un domino (2,5)

m1 = **2**
m2 = **5**

un domino (3,4)

m1 = **3**
m2 = **4**

un domino (2,5)

m1 = **2**
m2 = **5**

un domino (6,6)

m1 = **6**
m2 = **6**

Classes & Objets

```
class X {
```

attributs (variables d'instance)

constructeurs

```
}
```

- La manière que les objets doivent être conçus

Constructeur d'objet

Constructeur: même nom que la classe

```
/** Classe représentant un domino avec ses 2 marques. */
class Domino {
   // Attributs
   int  m1; // marque 1
   int  m2; // marque 2
   /** Le constructeur de domino */
   public Domino (int vm1, int vm2){
      // Instructions de construction
      // qui doivent affecter les marques
      // du domino en cours de construction
      // avec les valeurs des paramètres
   }
}
```

La signature des constructeurs est particulière :

elle n'admet pas de type de retour !

Déclaration « public » seulement et non plus « *public static* » !

Constructeur d'objet

this: référencer à l'objet en cours de construction

```
/** Classe représentant un domino avec ses 2 marques. */
class Domino {
  // Attributs
  int  m1; // marque 1
  int  m2; // marque 2
  /** Le constructeur de domino */
  public Domino (int vm1, int vm2){
    // Instructions d'initialisation
    // du domino en cours de construction
    this.m1 = vm1; // marque 1
    this.m2 = vm2; // marque 2
  }
}
```

Notation pointée habituelle

Moyennant l'usage de « this », le constructeur peut accéder aux attributs valués de l'objet en cours de construction.

Constructeur d'objet

différents schémas

- Un domino double
- Un domino par défaut

ICI, on dira qu'il y a **SURCHARGE** de constructeurs.

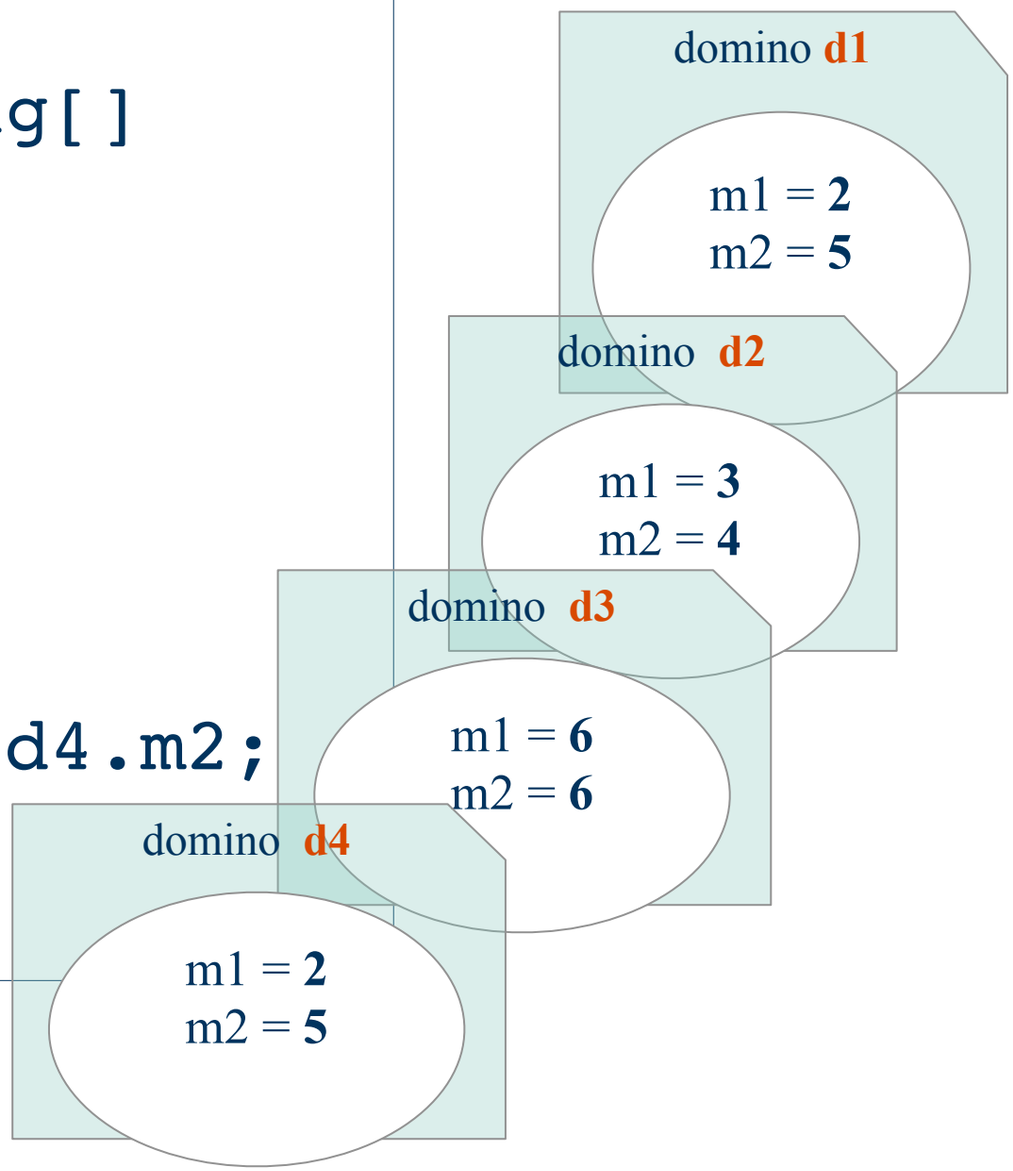
```
/** Classe représentant un domino avec ses 2 marques. */
class Domino {
   // Attributs
   int  m1; // marque 1
   int  m2; // marque 2
   /** Constructeur avec 2 marques */
   public Domino (int vm1, int vm2){
      this.m1 = vm1; this.m2 = vm2;
   }
  /** Constructeur de domino double */
   public Domino (int vm){
      this.m1 = this.m2 = vm;
   }
  /** Constructeur de domino par défaut */
   public Domino (){
      this.m1 = 0; this.m2 = 0;
   }
}
```

Construction d'objet

new <Constructeur> (Paramètres)

```
class MainDominos {
   /** Construction de dominos */
   public static void main(String[]
args) {
    Domino d1 = new Domino(2,5);
    Domino d2 = new Domino(3,4);
    Domino d3 = new Domino(6);
    Domino d4 = new Domino(2,5);
    // Faire quelque chose des dominos...
    if ( d4.m1!=d4.m2 ) d4.m1 = d4.m2;
   }
}
```

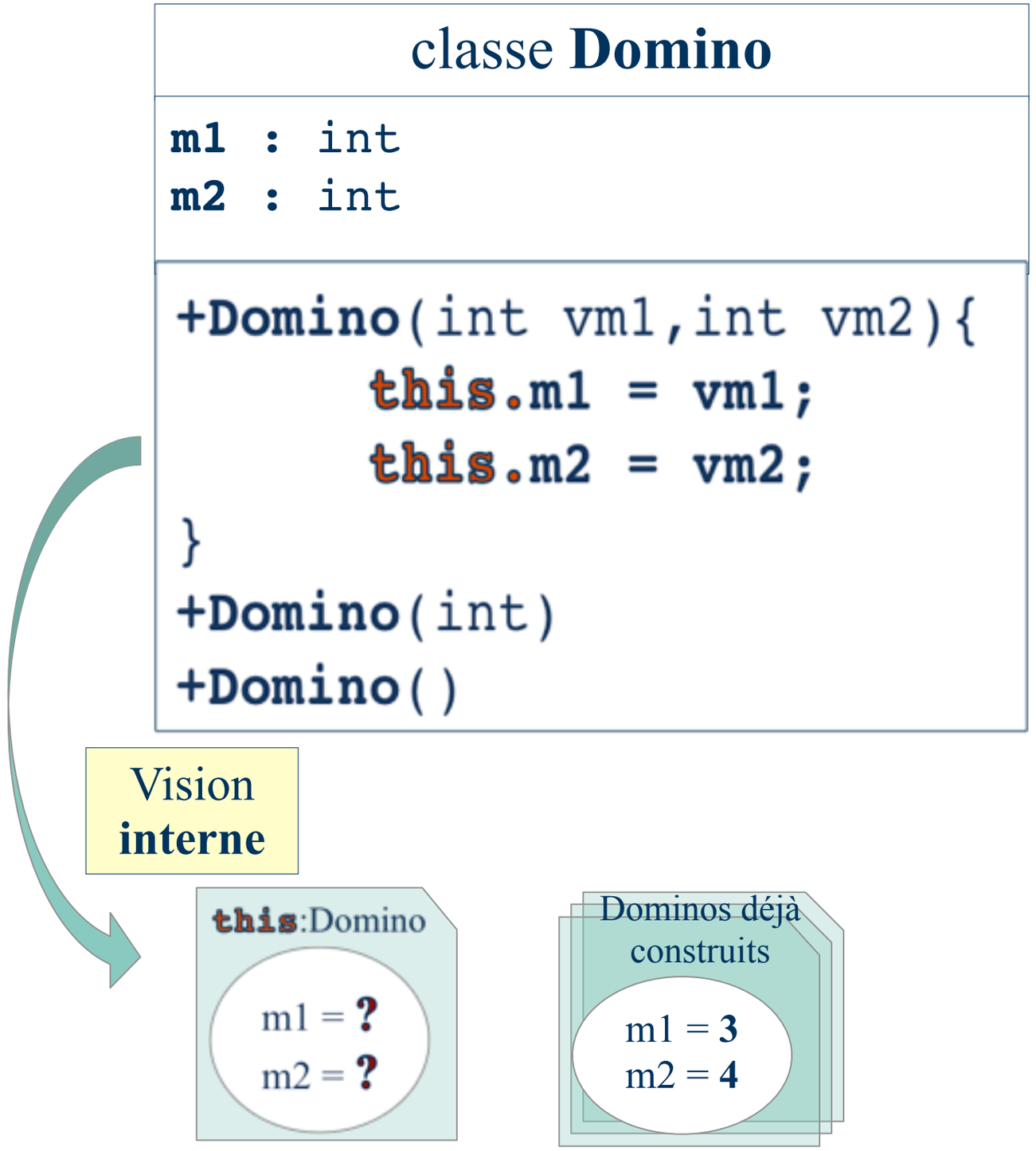
d1 et d4 sont des objets différents !



Vision interne

Quand le code en cours d'exécution se trouve

- Dans la class d'un objet
- On accède aux propriétés de l'objet en cours d'exécution à l'aide de la variable auto-référencement this



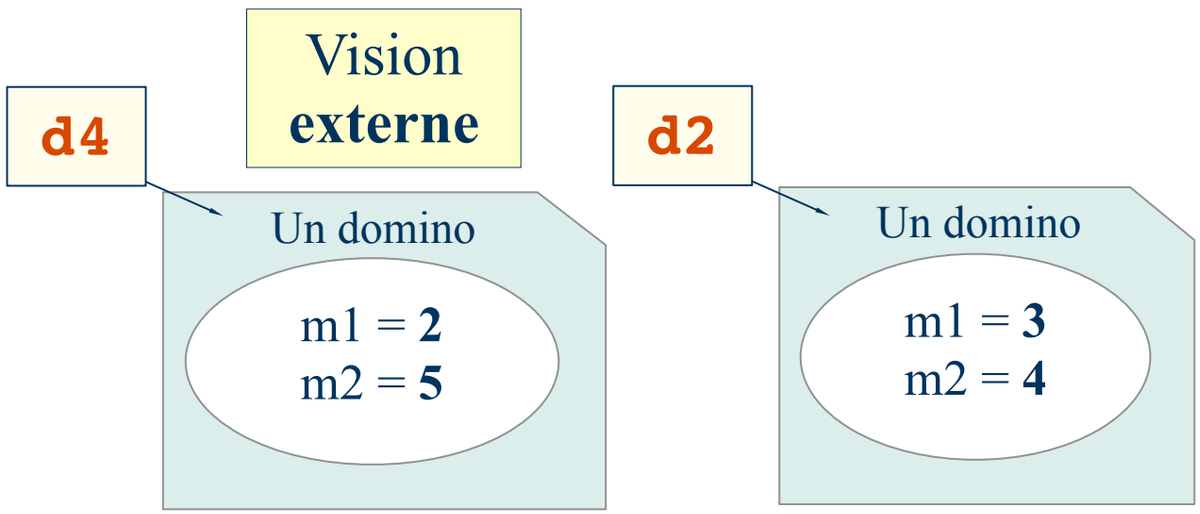
Vision externe

Quand le code en cours d'exécution se trouve

- En dehors de la class d'un objet
- Le code en cours d'exécution accède à l'objet à l'aide d'une référence externe (OID) à l'objet

classe **MainDominos**

```
+$ main(String[]) {
 Domino d2 =
        new Domino(3,4);
 Domino d4 =
        new Domino(2,5);
 if ( d4.m1 != d4.m2 )
    d4.m1 = d4.m2;
}
```



Référence externe (OID) versus this

classe **MainDomino**

main(int,int)

```
Domino d4 =
   new Domino(2,5);
if ( d4.m1 != d4.m2 )
    d4.m1 = d4.m2;
```

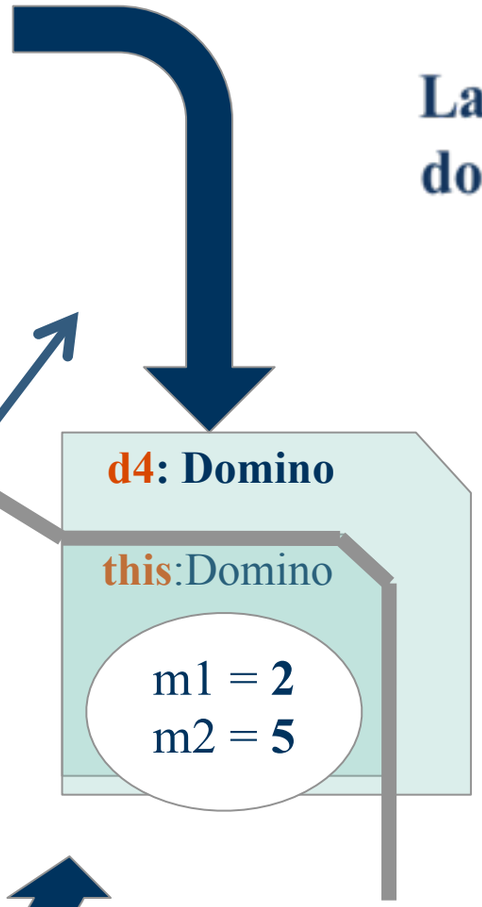
classe **Domino**

Domino(int,int)

En dehors d l'objet,
on utilise objet

Dans l'objet,
on "est" objet

```
this.m1 = vm1;
this.m2 = vm2;
```



La méthode *main* voit le domino au travers de sa référence ou OID.

Ici, d4.

Elle ne peut faire autrement que de l'utiliser car *this* n'a pas de sens pour le domino *d4* dans *main*.

TD 1

- Dominos
  - Classes & Objets
  - Attributs + Contructeurs
- Bon codeur vs excéllent codeur
- Questions