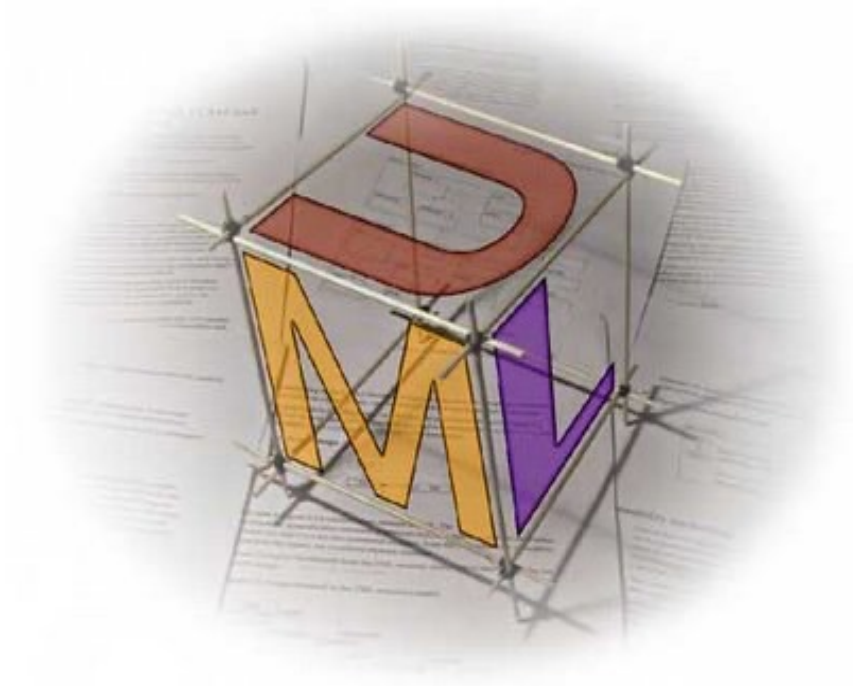


Approche Orientée Objet et UML

Remy.courdier@univ-reunion.fr

Plan du cours

L'objet

- Introduction au Génie Logiciel
- **L'approche Orientée Objet et UML**
- Les diagrammes de modélisation
- Relations entre les différents diagrammes
- De l'analyse à la conception
- Relation entre les notations OMT et UML
- Les Design Pattern

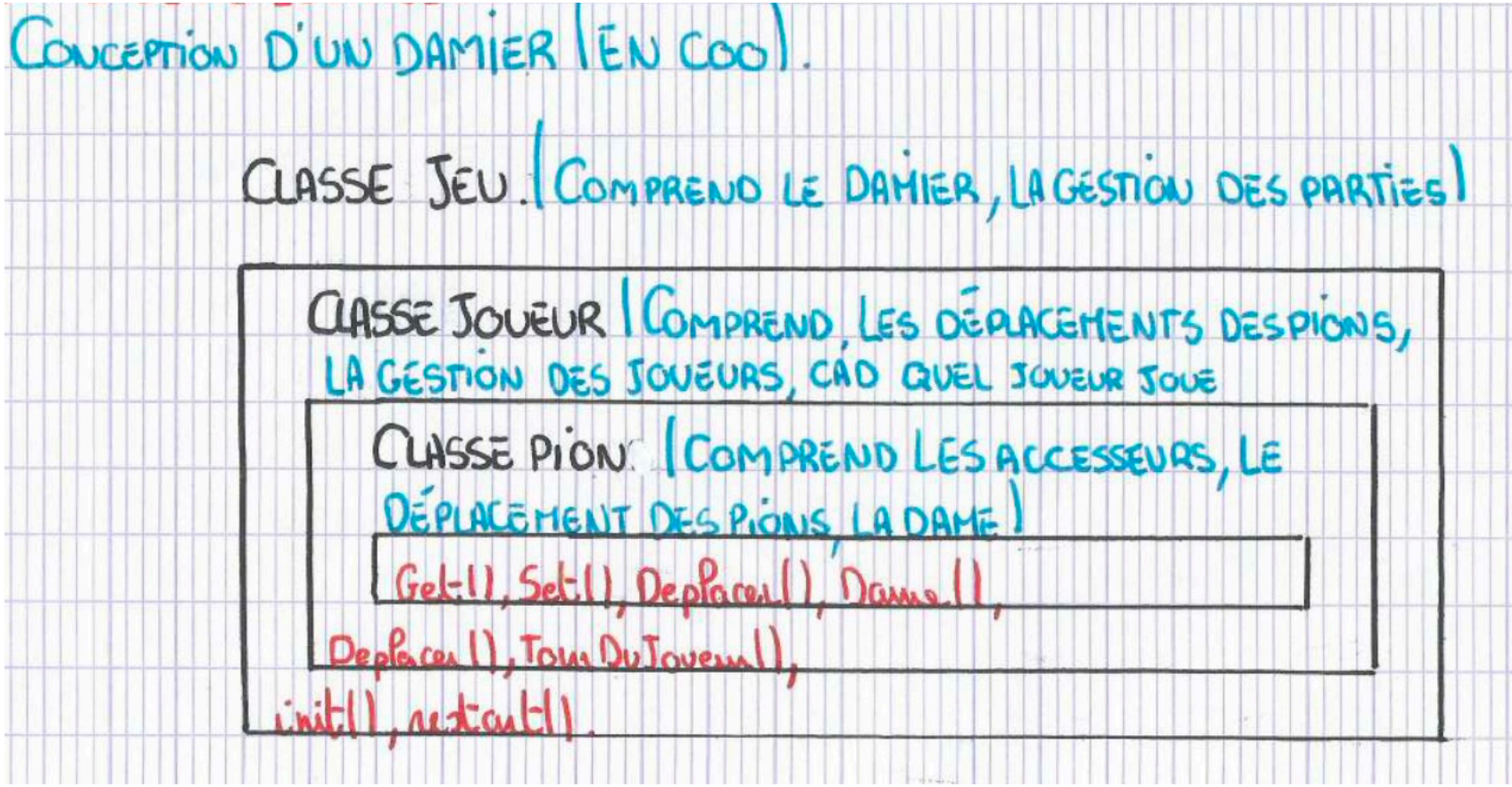
Conception et Objet

L'objet

Comment « desiner » une architecture

L'objet

Architecture objet d'un jeu de Dame…



Est-ce suffisant pour l'équipe de développeur ?

Chapitre 2 : L'approche Orientée Objet

L'objet

- Rappel des principes de l'orienté objet
- Les Objets
- Les messages ou la communication entre objets
- Les classes
- Les relations entre les classes
- L'héritage entre classes
- Les architectures à base d'objets

2.1. Rappels des principes de l'orienté objet

L'objet

- Calquer le découpage de la représentation informatique sur les entités physiques et virtuelles mises en jeu dans les processus réels que l'on cherche à modéliser ou à automatiser
- Privilégier une approche architecturale pour implémenter des solutions à des problèmes plutôt qu'une approche fonctionnelle visant à résoudre un problème posé

- ✓Stabilité de la modélisation par rapport aux entités du monde réel
- ✓Construction itérative facilitée par le couplage faible entre composants
- ✓Possibilités de réutiliser des éléments d'un développement à un Autre
- ✓…

2.2 Les Objets

L'objet

Définition :

- ✓ Un objet définit une représentation d'une entité atomique réelle ou virtuelle, dans le but de le piloter ou de le simuler.
  Les objets encapsulent une partie des connaissances du monde dans lequel ils évoluent.
- ✓ Un objet associe données et traitements en ne laissant visible que l'interface de l'objet, c'est à dire les traitements que l'on peut lui appliquer.

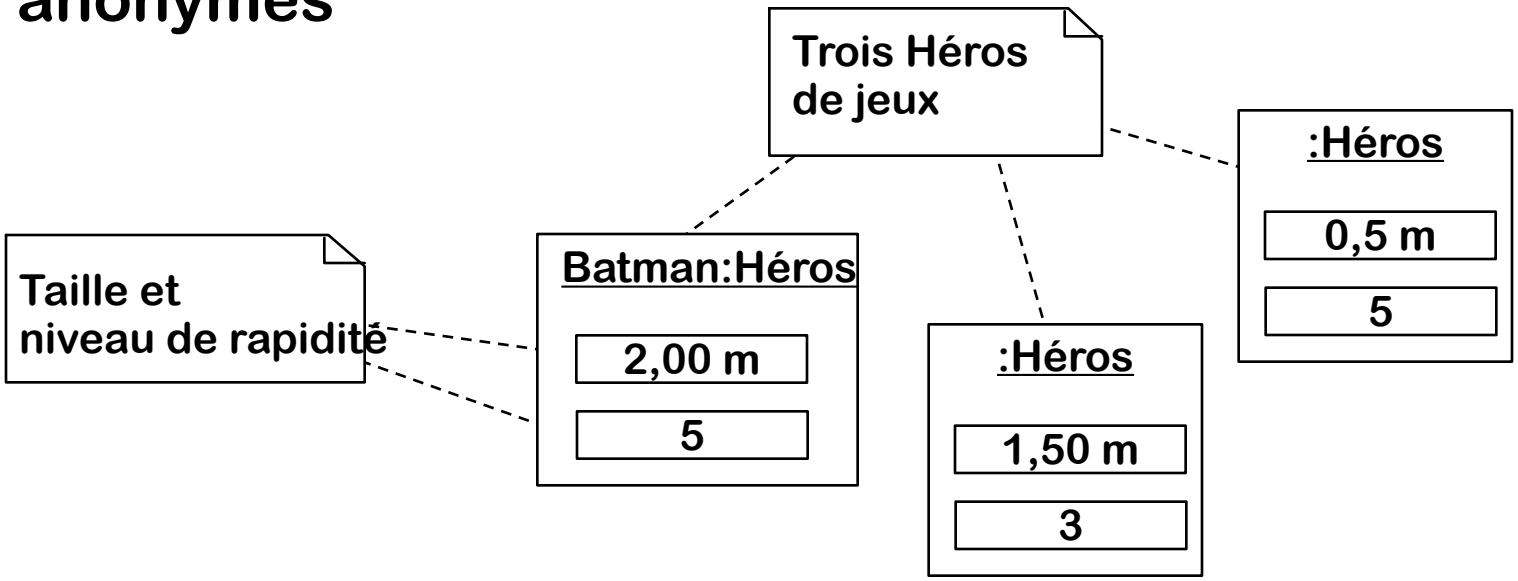
Objet = Identité + Etat + Comportement

- ✓ **L'Identité :** En plus de son état un objet possède une identité qui permet de le distinguer de manière non ambiguë indépendamment de son Etat.
- ✓ **L'état** : regroupement des valeurs instantanées de tous les attributs d'un objet
- ✓ **Le comportement** : regroupe toutes les compétences d'un objet et décrit les actions et les réactions de cet objet

Notations

L'objet

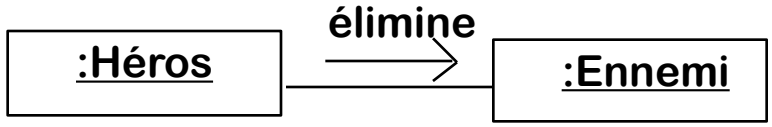
- En UML un objet se représente sous la forme d'un rectangle; le nom de l'objet est souligné
- Le rectangle dont le coin en haut est replié représente une information de clarification optionnelle appelée note
- Les deux points précisent qu'il s'agit d'objets anonymes



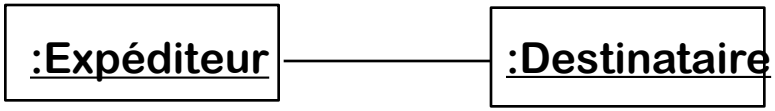
Notations

L'objet

- Les messages sont représentés par des flèches placées le long des liens qui unissent les objets



- Notion de synchronisation



envoi simple (un seul objet à la fois est actif)

envoi synchrone (expéditeur bloqué jusqu'à acceptation du destinataire)

envoi dérobant (destinataire bloqué jusqu'à réception du message)

envoi minuté (bloque l'expéditeur pendant un temps donné)

envoi asynchrone (n'interrompt pas l'exécution de l'expéditeur)

2.4. La classe

L'objet

- Entité de modélisation qui décrit un ensemble d'objets qui partage les mêmes :
  - ✓ caractéristiques,
  - ✓ contraintes,
  - ✓ sémantiques (sens).

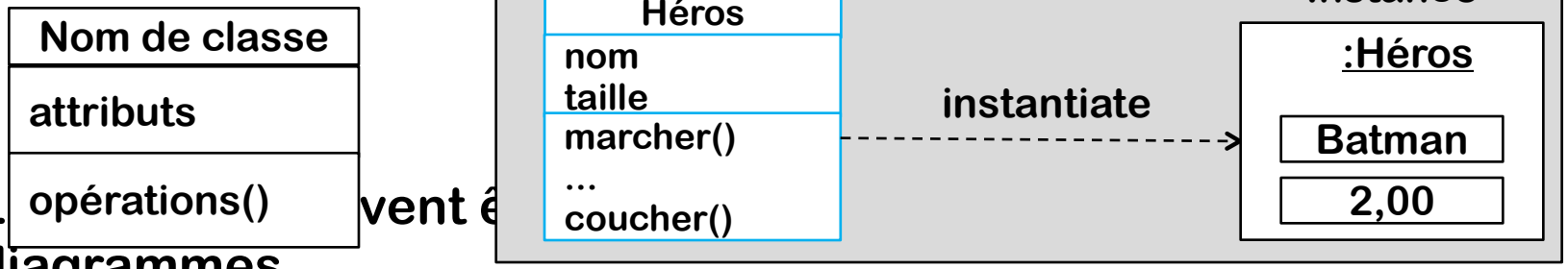
- Description
  - ✓ La classe correspond à une abstraction extraite d'une réalité
  - ✓ La classe décrit le domaine de définition d'un ensemble d'objets : c'est un modèle qui définit les données et les traitements communs à une collection d'objets similaires
  - ✓ La classe peut être vue comme un « moule » à objet
  - ✓ Chaque objet est créer par une seule classe
  - ✓ Les généralités sont contenues dans la classe et les particularités dans les objets

Notation

L'objet

- **Terminologie orientée objet**
  - ✓Les traitements sont appelés **méthodes ou opérations** de l'objet
  - ✓Les données sont appelées **variables**, **attributs ou propriétés**
  - ✓Les objets sont construits à partir de la classe par un processus appelé **instanciation**
  - ✓Tout objet est **instance** d'une Classe

- **Chaque classe est représentée sous la forme d'un rectangle divisé en 3 parties**



- **L… vent é… diagrammes**

Nom de classe

Héros

L'encapsulation

L'objet

L'occultation des détails de réalisation est appelée encapsulation

- ✓ **Par défaut les valeurs des attributs d'un objets sont encapsulées dans l'objet et ne peuvent pas être manipulées directement par un autre objet**

Un accès « libre » a des attributs peut conduire rapidement à des données mal gérees

L'encapsulation

L'objet

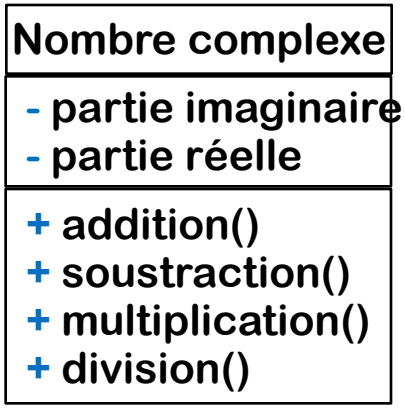
L'occultation des détails de réalisation est appelée encapsulation

✓ Des règles de visibilité précisent la notion d'encapsulation
- assouplissement du degré d'encapsulation et de protection au profit de certaines classes bien particulières
- intérêt : réduire le temps d'accès aux attributs

✓ Trois niveaux d'encapsulation :
- privé (-): attribut non vu de l'extérieur de l'objet

```
en C++ les classes amies peuvent
encore accéder aux attributs
```

- public (+): attribut visible pour toutes les classes
- protégé (#) (voir plus loin)



Variable de classe vs Variable d'instance

L'objet

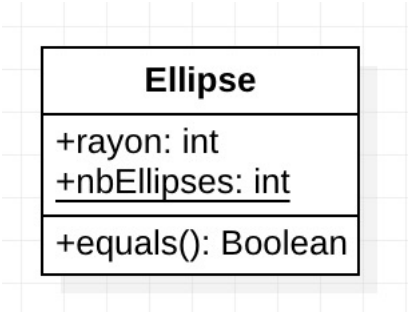
Une variable de classe est propre à la classe

Une variable d'instance est propre à l'objet

- En d'autres termes si on instancie plusieurs objets à partir d'une classe X :
  - ✓ les valeurs des variables d'instances de cette classe seront différenciés d'un objet à l'autre.
  - ✓ les valeurs des variables de classe de la classe X seront les mêmes pour tous les objets de la classe X.

Une variable de classe peut être utilisée dans un code sans qu'il existe une instance de classe mais en utilisant directement la classe.
•Par exemple si on considérer la variable de classe nbX, on pourra écrire X.nbX = 0
•Si il existe un instance unX de la classe X, cette instance pourra également manipuler cette variable comme toute autre variable, on pourra écrire : unX.nbX = 0

- Notation UML
  - ✓ Les variables et opérations de classes sont soulignées



Dans cet exemple, on suppose que nbEllipse est une variable de classe qui comptabilise le nombre d'instances d'Ellipses

Variable de classe et d'instance par l'exemple

L'objet

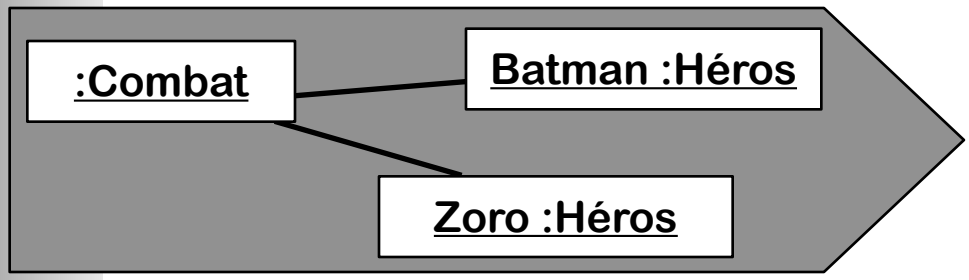
```
/* Classe Personnage */
public class Personnage {
   // variable de Classe: sa valeur est la même pour tous les objets de la classe en java utilisation du mot clé « static »
   private static int nbPersonnages = 0;   // nombre de personnages créés (initialement nul donc)
   // variable d'Instance: sa valeur est propre à chaque instance (donc peut varier d'un objet à un autre)
   private int nbVies; // nombres de vies restantes au personnage créé
   private String nom; // nom du personnage

   public Personnage(String pNom, int nbViesDepart) {
      nbPersonnages++;    // un nouveau personnage est créé, on incrémente donc le nombre total de personnages créés
      this.nom = pNom;
      this.nbVies = nbViesDepart; // on initialise le nombre de vies dont bénéficie le personnage lors de sa création
      System.out.println(nom + " créé avec " + nbVies + " vie(s)");
      System.out.println("Nombre total de personnages créés: " + nbPersonnages);
   }
}
/* Classe Jeu */
public class Jeu {
   public static void main(Strind[] args) {
      Personnage neo   = new Personnage("Neo", 1);
      Personnage smith = new Personnage("Smith", 1000);
   }
}
```

2.5 Les relations entre les classes
La relation d'association

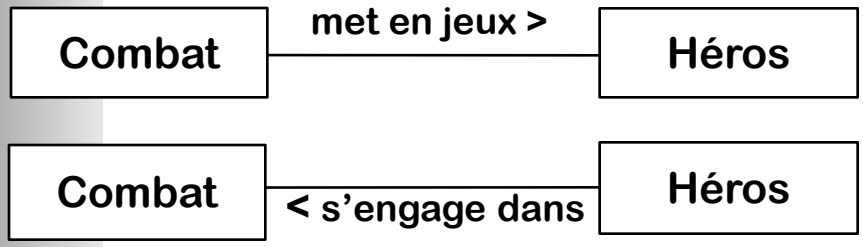
L'objet

Une association représente une relation sémantique entre les objets, c'est une abstraction des liens qui existent entre les objets instances



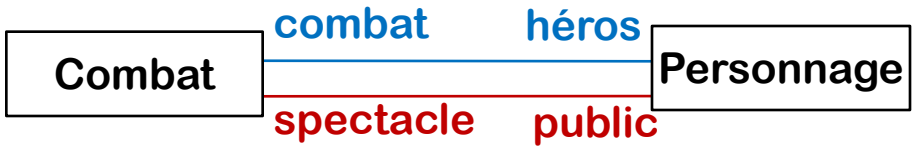
une association se représente par une ligne continue tracée entre les classes

Nommage d'une association



le nommage des associations facilite la compréhension des modèles

Rôle d'une association



Le rôle décrit comment une classe voit une autre classe au travers d'une association

Le rôle prend tout son intérêt lorsque plusieurs associations existent entre 2 classes

Multiplicité et Contraintes

L'objet

- La multiplicité précise le nombre d'instances (et non de liens comme dans Merise) qui participent à la relation

Trop d'instances peuvent conduire à une catastrophe !

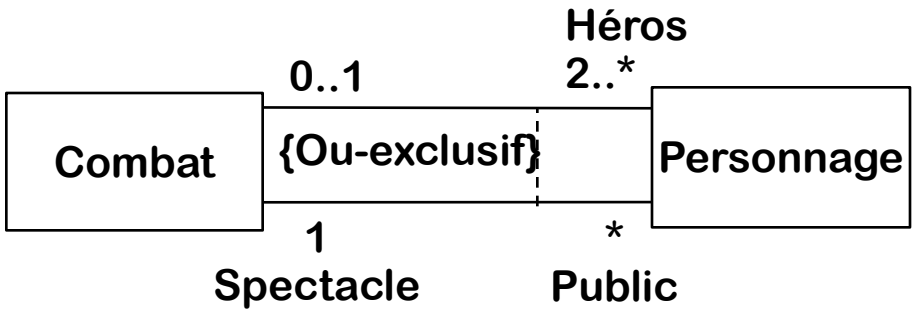
Imaginez le même type de problème avec un A380…

Multiplicité et Contraintes (2)

L'objet

- Toutes sortes de contraintes peuvent être définies sur une relation ou un groupe de relations

```
exemple : {ordonnés} ; {Sous-ensemble} ; (Ou-exclusif}
```

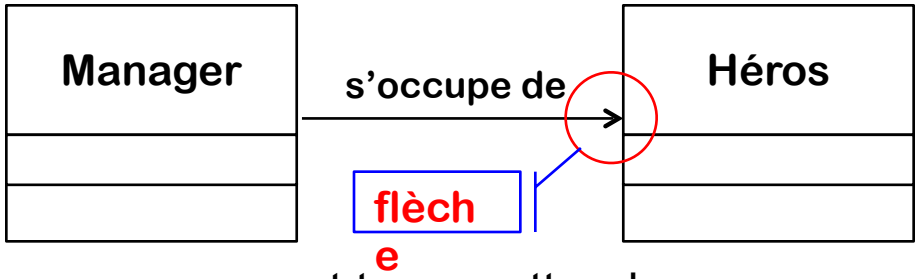


Un combat met en jeu différents personnages ; certains joueront le rôle du public et d'autres le rôle de Héros de combat. Un Héros donné peut s'engager dans un combat ou non. Un personnage du public est toujours spectateur d'un combat.

Navigabilité et association récursive

L'objet

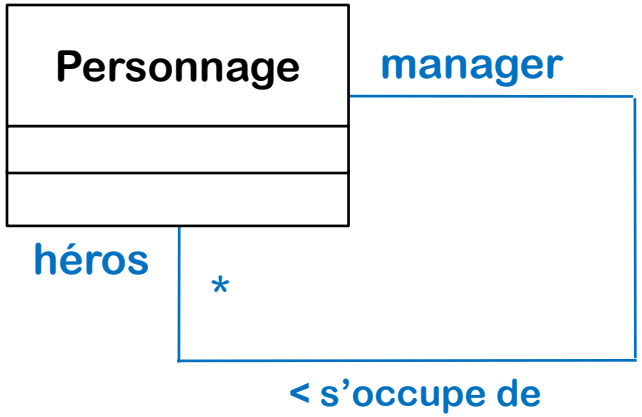
Association directionnelle / Directed association



Restriction de la navigabilité d'une association à un seul sens à l'exécution.

Le manager peut transmettre des messages à un Héros mais un Héros ne peut pas prendre l'initiative de communication avec le Manager

Association récursive

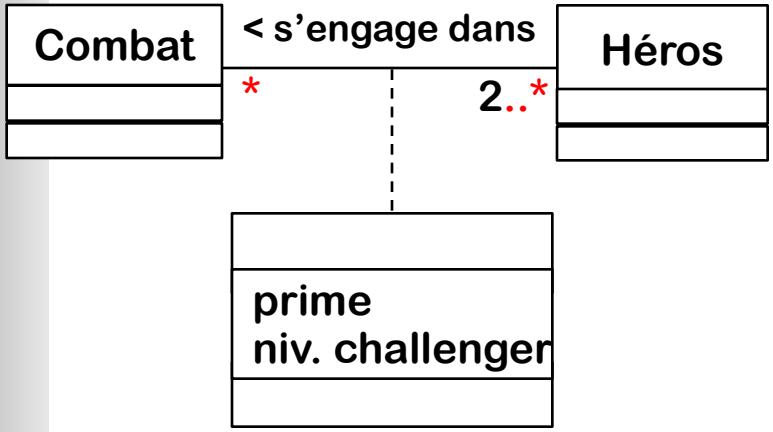


Un personnage qui joue le rôle de manager peut s'occuper de plusieurs personnage jouant le rôle d'héros. Un personnage héros est pris en charge par un personnage manager et un seul.

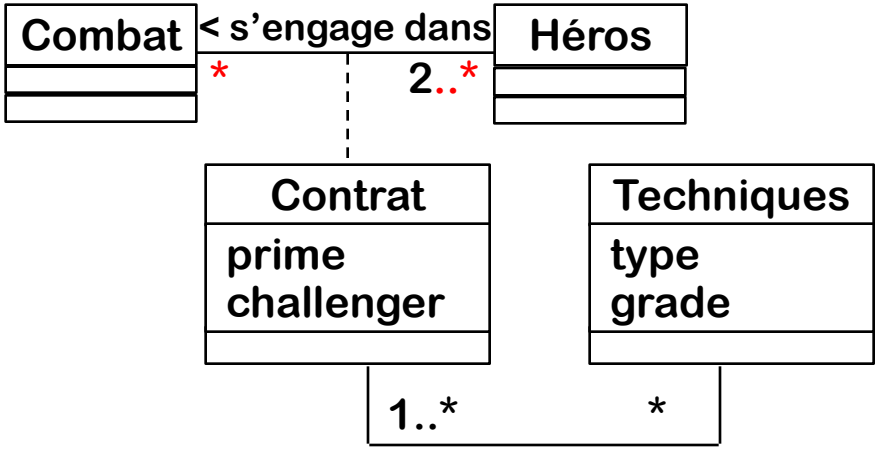
La classe-association

L'objet

- Une association peut être représentée par une classe pour ajouter des attributs et des opérations
  - ✓Une telle classe peut participer à plusieurs relations
  - ✓Une telle classe qui ne participe pas à d'autres relations est appelée **association attribuée** et ne porte pas de nom
- La notation UML utilise une ligne pointillée pour attacher une classe à une association



association attribuée

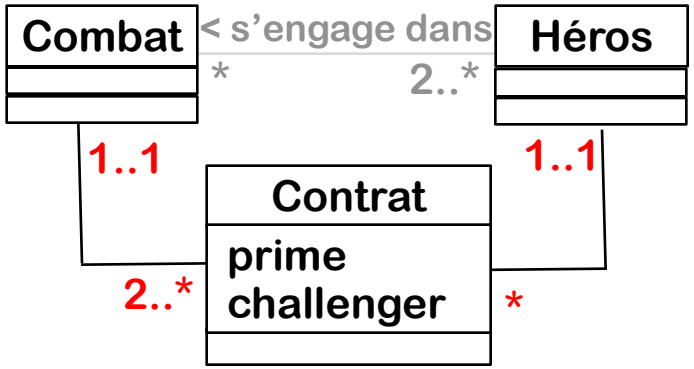


classe association

La classe-association

L'objet

- Une association peut être représentée par une classe pour ajouter des attributs et des opérations
  - ✓Une telle classe peut participer à plusieurs relations
  - ✓Une telle classe qui ne participe pas à d'autres relations est appelée **association attribuée** et ne porte pas de nom
- La notation UML utilise une ligne pointillée pour attacher une classe à une association



classe association

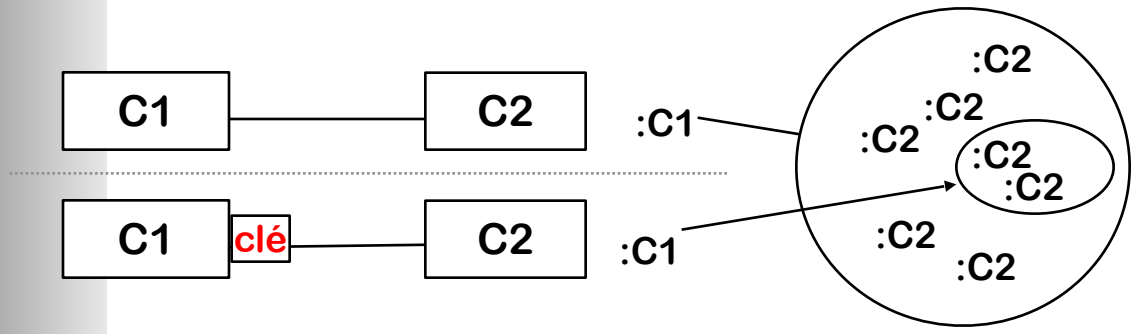
Si on enlève ici l'association directe entre Combat et Héros, définissez les modalités manquante sur ce diagramme

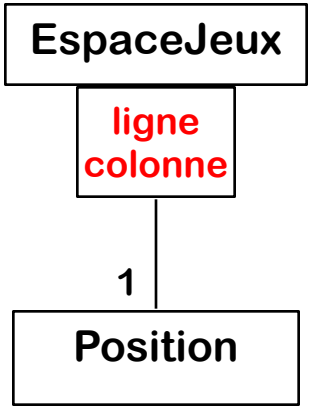
Les restrictions ou qualification

L'objet

Une qualification consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une relation

- ✓ réduction du nombre d'instances qui participent à une association (multiplicité ou cardinalité)
- ✓ Une restriction est réalisée au moyen d'un attribut particulier appelé "Clé"
- ✓ La clé est représentée au niveau de la classe de départ dans un compartiment rectangulaire



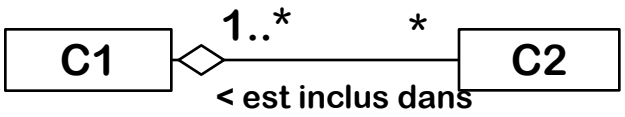


La relation d'agrégation

L'objet

- Il s'agit d'une variante de l'association portant la sémantique suivante :
  « Est inclus dans »
- Propriétés
  - ✓une classe fait partie d'une autre classe (composition)
  - ✓les valeurs d'attributs d'une classe se propagent dans les valeurs d'attributs d'une autre classe
  - ✓une action sur une classe implique une action sur une autre classe
  - ✓les objets d'1 classe sont subordonnés à ceux d'1 autre classe

  Note : L'agrégation n'implique par contre pas nécessairement ces critères !
- Notation UML :
  - ✓Une agrégation se représente avec un losange placé du coté de l'agrégat



La relation de composition
ou d'agrégation forte

L'objet

- **Représente une contenance structurelle entre instances**
  Décrit la notion de « tout et parties » dans laquelle les parties sont indissociables du tout.

- **Association portant la sémantique suivante :**
  « est composé de »

- **Propriétés**
  La cardinalité placé à coté de l'agrégat et facultative et correspond par défaut à **1..1**, elle peut être **0..1** mais jamais **x..*.**

  Une partie d'un tout n'appartient qu'à un seul tout.

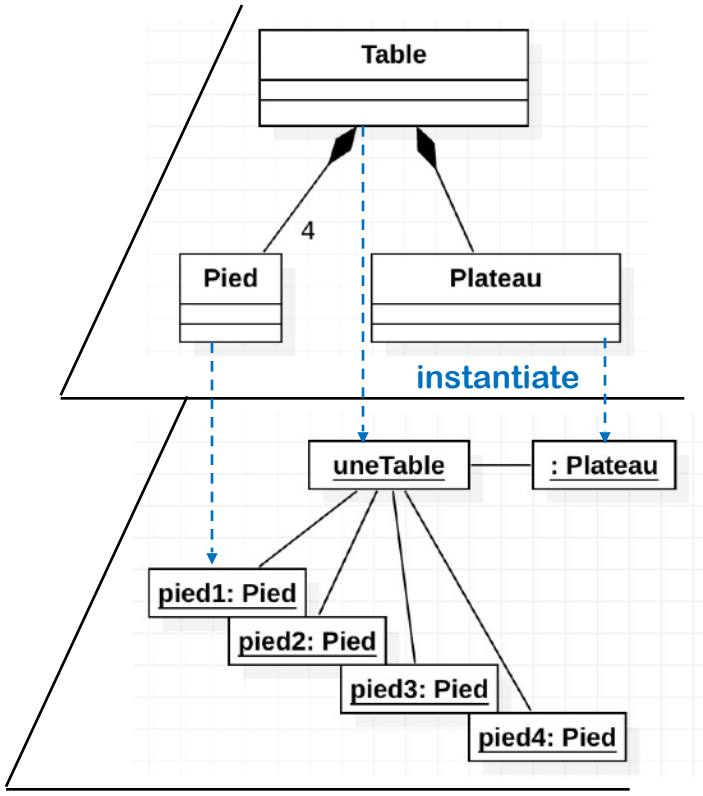
- **Notation UML :**
  Une composition se représente avec un losange **plein** placé du coté de l'agrégat.

Agrégation et Composition

On utilise une composition et non une simple agrégation si :

- La destruction de l'instance représentant « le tout »
implique obligatoirement la destruction des instances représentant les parties (les composants)

- La copie de l'instance représentant « le tout »
implique nécessairement la copie de ses composants car ceux-ci ne peuvent faire partie que de ce « tout » et ne peuvent pas être associés à d'autres « tout ».



L'utilisation de la composition indique au développeur par exemple que la suppression de l'instance uneTable implique la suppression de l'instance de Plateau et des 4 instances pied1, pied2; pied3, pied4.

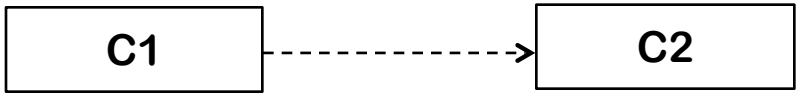
2.7 La relation de Dépendance

L'objet

- **Relation exprimant une dépendance sémantique entre les éléments du modèle Cette relation est nécessairement orientée.**

- **Principaux cadres d'utilisation :**
  - ✓ **Dépendance d'interface** : entre une classe et une interface que celle-ci s'impose de respecter.
    Sémantique « *réalise* » ou « *implemente* »
  - ✓ **Dépendance de stéréotype** : Pour définir une classe générique paramétrable.
    Sémantique « *bind* »
  - ✓ **Dépendance d'instanciation** : Pour décrire une relation entre une classe et ses instances
    sémantique « *instanceof* » (ou « instantiate » dans le sens inverse)

- **Notation UML :**
  Représentée par une ligne pointillée orientée



2.6 La relation de Généralisation / héritage

L'objet

Les hiérarchies de classes ordonnent les objets au sein d'arborescences d'abstraction croissante

Généralisation

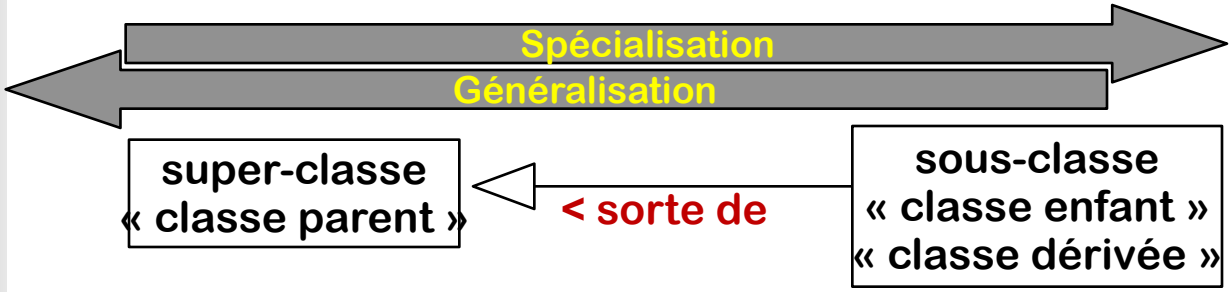
- ✓ Factorisation des éléments communs d'un ensembles de classes dans une classe plus générale appelée super-classe

Spécialisation

- ✓ Capture des particularités d'un ensemble d'objets non discriminés par les classes déjà identifiées

Notation UML

- ✓ La généralisation est symbolisée par une ligne entre deux classe terminée par une flèche qui pointe vers la classe la plus générale



Une classe enfant hérite des attributs (variables et opérations) de la classe parent.

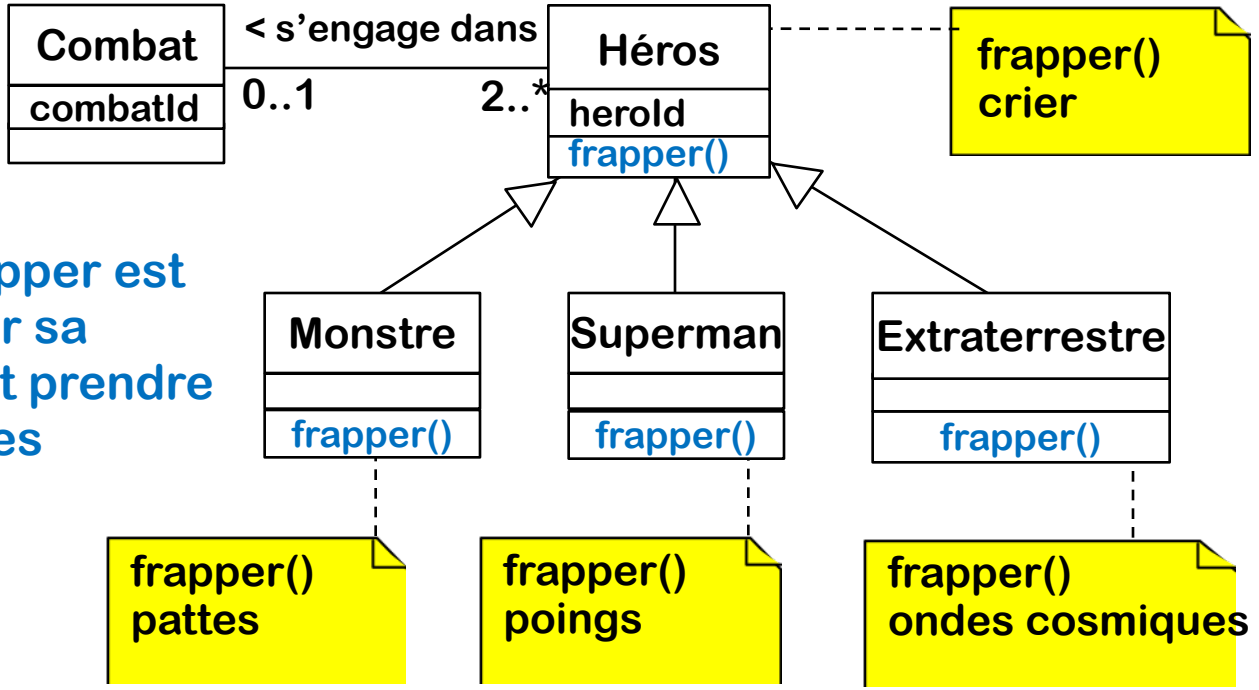
Le polymorphisme

L'objet

"caractéristique d'un élément qui peut prendre plusieurs formes"

- Chaque sous classe peut modifier localement le comportement de ses opérations pour considérer le particularisme de son niveau d'abstraction

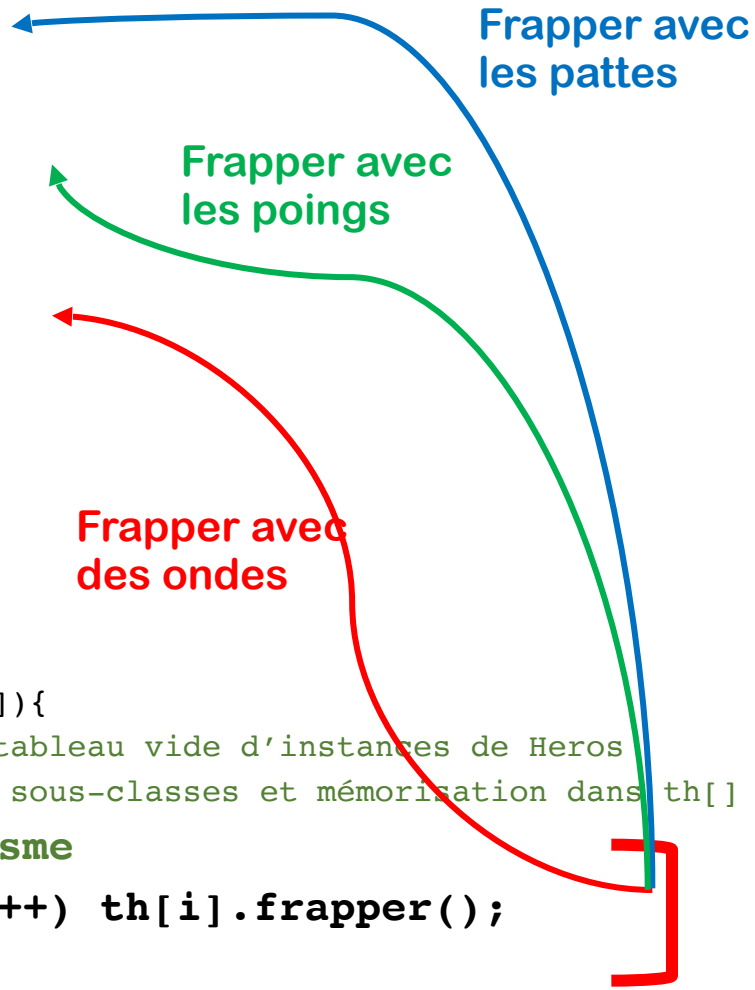
L'opération frapper est polymorphe car sa réalisation peut prendre plusieurs formes



Le polymorphisme par l'exemple (Java)

L'objet

```
class Monstre extends Heros {
public void frapper() {//TODO CODE}
}
class Superman extends Heros {
public void frapper() {//TODO CODE}
}
class Extraterrestre extends Heros {
public void frapper() {//TODO CODE}
}
abstract class Heros {
public int heroId;
public void frapper() {//TODO CODE}
}
public class Combat {
public int combatId;
private Heros th [];
public static void main (String args[]){
th[] = new Heros[3]; //création d'un tableau vide d'instances de Heros
//… creation de heros des différentes sous-classes et mémorisation dans th[]
// application du polymorphisme
for (int i=0; i<tf.length; i++) th[i].frapper();
}
}
```

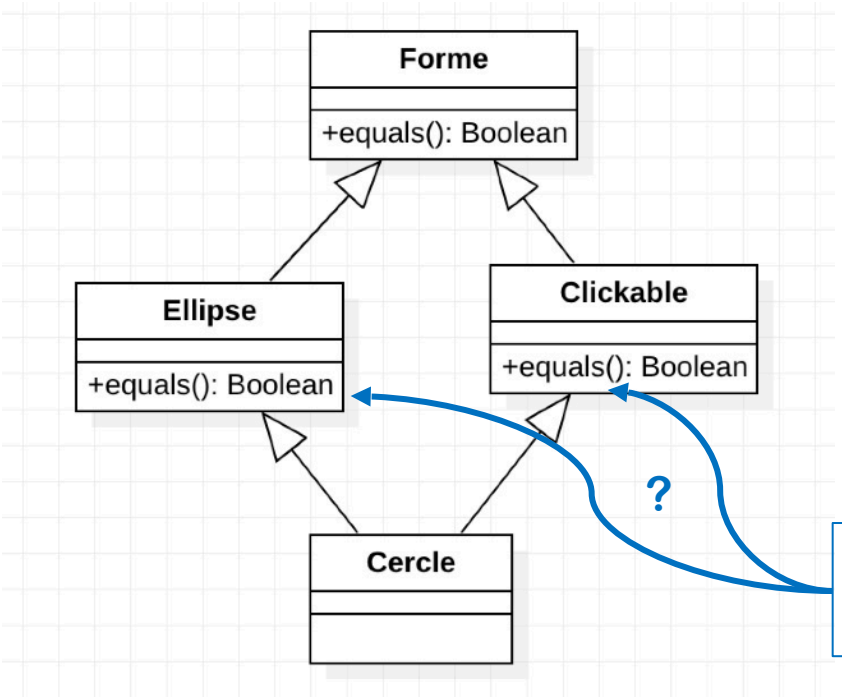


Le Multihéritage

L'objet

UML autorise implicitement l'héritage multiple

=> Problème du diamant



- Ici la classe Cercle hérite de deux façons de réaliser l'opération equals().
- UML ne fournit aucune résolution ou recommandation explicite pour ces problèmes et ambiguïtés
- C'est lors de l'implémentation avec le choix du langage qu'il faut retravailler ce diagramme.

```
private Cercle unCercle;
unCercle = new Cercle;
unCercle.equals();
```

Combinaison de généralisation (Generalization Set)

L'objet

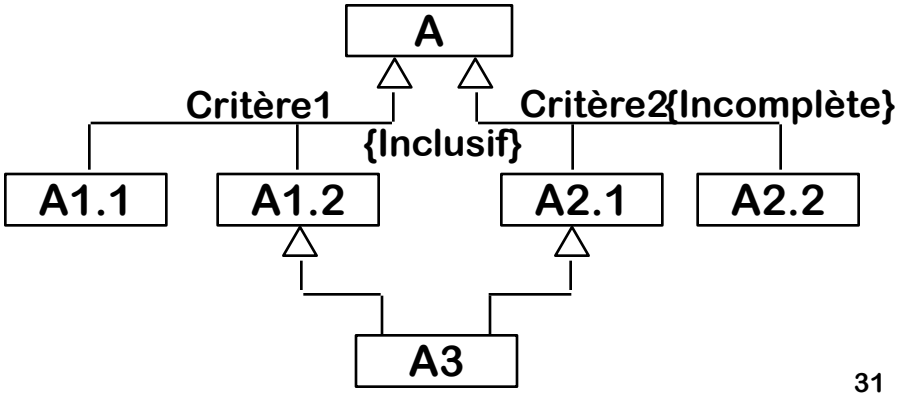
Critères de généralisation

- ✓Spécialisation selon plusieurs critères simultanément
- ✓On associe des discriminants à la relation de généralisation

Contraintes

- ✓généralisation exclusives /inclusives
  - {Exclusif} : par défaut une généralisation est exclusive ; une classe descendante d'une classe A peut être descendante d'une seule sous-classe de A
  - {Inclusif} : Une classe descendante de la classe A appartient au produit cartésien des sous-classes de la classes A

- ✓Généralisation complète / Incomplète
  - {Complète} : *existence de toutes les sous-classes*
  - {Incomplète} : généralisation extensible



Les classes abstraites

L'objet

Classe de spécification générale

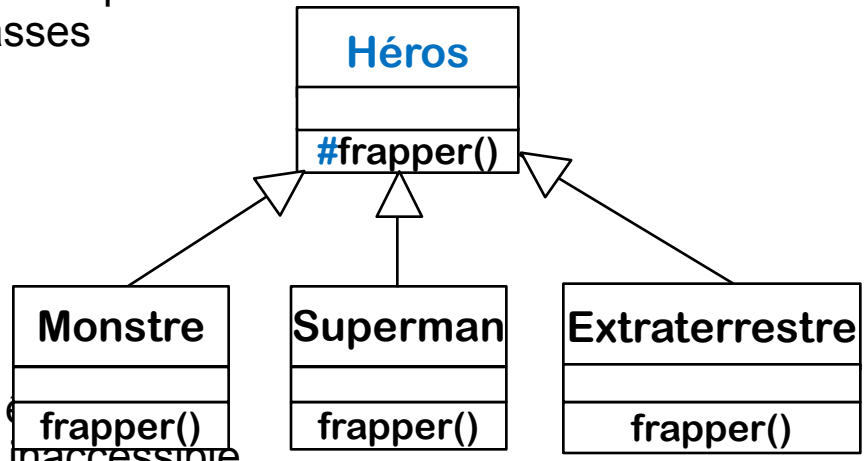
- ✓ Classe qui ne permet pas l'instanciation d'objets
- ✓ Il s'agit de sorte de type de classe

Base pour les logiciels extensibles

- ✓ L'ensemble des mécanismes généraux est décrit dans les classes abstraites sans tenir compte des spécificités rassemblées dans les classes "concrètes"

Notation UML :

- ✓ **Le nom des classes abstraites est en italique**
- ✓ **Encapsulation :** Un attribut pouvant être accessible par les classes enfants et inaccessible aux autres classes est qualifié de : « protected » - notation UML « # »



Opération de classe : Terminologie

- La signature d'une opération est constituée de son nom et de ses paramètres.

- La surcharge d'une opération héritée consiste en la définition d'une opération de même nom que celle de la classe parent mais de signature différente.

- La redéfinition d'une opération héritée consiste en la définition d'une opération de la classe parent avec la même signature.
  - ✓ On parle de redéfinition d'opération avec spécialisation lorsque l'opération fait appel à l'opération de la classe parent dans son implémentation.
  - ✓ On parle de redéfinition d'opération avec surcharge lorsque l'opération masque l'opération de la classe parent.

Le polymorphisme par l'exemple (Java)

L'objet

```
class Monstre extends Heros {
public void frapper() {//TODO CODE}
}
class Superman extends Heros {
public void frapper() {//TODO CODE}
}
class Extraterrestre extends Heros {
public void frapper() { super.frapper() //TODO CODE}
}
abstract class Héros {
public int heroId;
public void frapper() {//TODO CODE}
}
public class Combat {
public int combatId;
private heros th [];
public static void main (String args[]){
th[] = new Heros[3]; //création d'un tableau vide d'instances de Heros
//… creation de heros des différentes sous-classes et mémorisation dans th[]
// application du polymorphisme
for (int i=0; i<tf.length; i++) th[i].frapper();
}
}
```

redéfinition avec surcharge

Frapper avec les pattes

Frapper avec les poings

redéfinition avec spécialisation

Frapper avec des ondes

2.8 Les architectures à base d'objets

L'objet

Les mécanismes objets se retrouvent :

- ✓ dans les langages de programmation, on parle alors de POO
- ✓ au cœur de certaines Bases de Données
- ✓ dans les environnements d'interface Homme-Machine
- ✓ dans l'archi. des documents du World Wide Web et de l'Internet
- ✓ comme base des modèles et méthodologies de développement

Langages de programmation orientés objets :

- ✓ Simula (1967) - Tony Hoare : classe, polymorphisme, héritage
- ✓ **Smalltalk (1972) - Alan Kay :** inspiré par Simula & Lisp : messages, encapsulation, typage et polymorphisme (via la sous-classification)
- ✓ **C++** (1983), **Objective-C** (1984), **Common Lisp Object** (1988),…
- ✓ **Java, C#, Python, JavaScript, …**

Les bus d'objets distribués :

- ✓ Corba et les ORB - (Common Object Request Broker Architecture)
- ✓ DCOM,
- ✓ Java RMI (Remote Méthode Invocation),…

Fin du Chapitre

L'approche Orientée Objet et UML