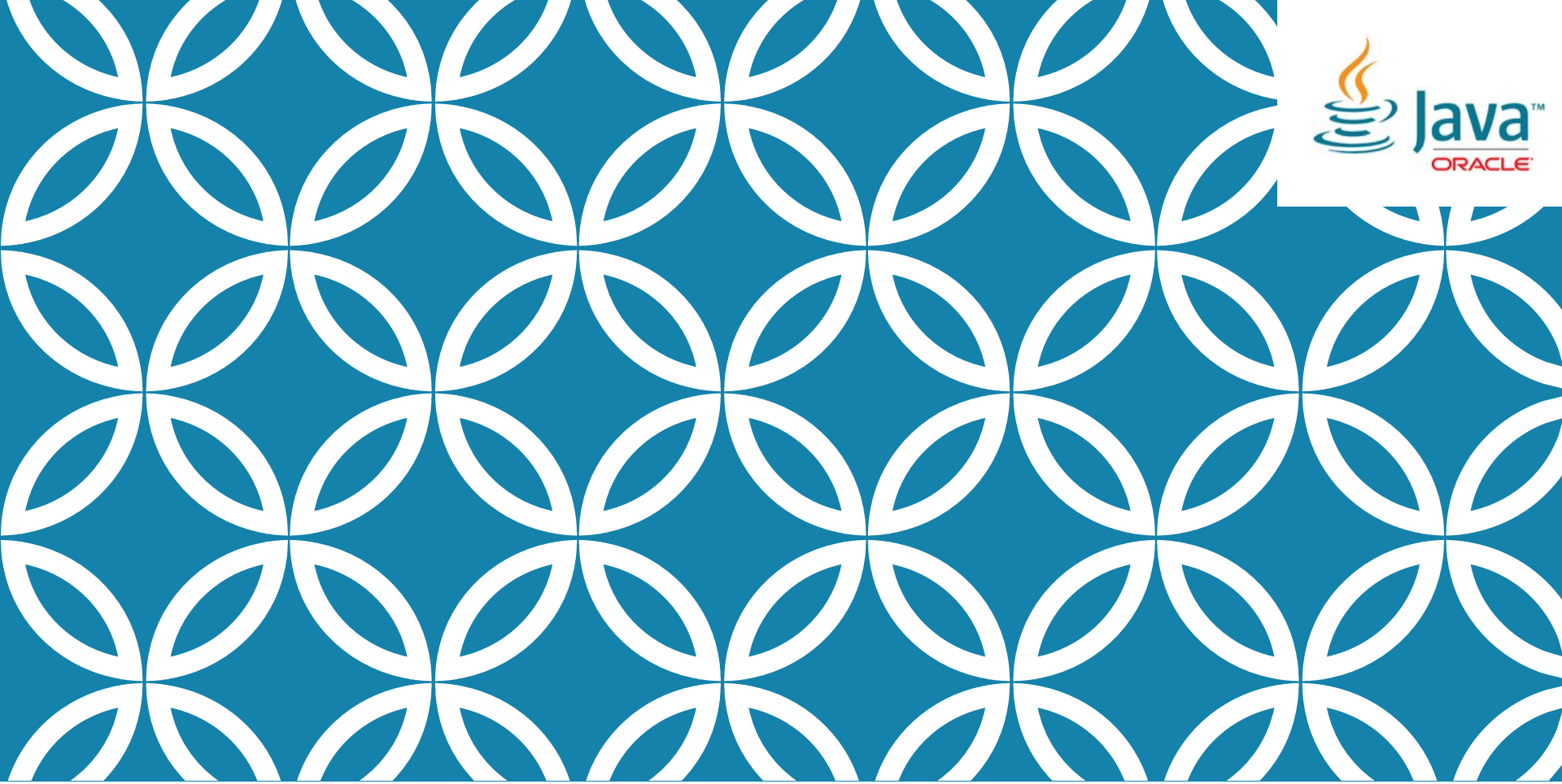


COURS 6 : HÉRITAGE ET POLYMORPHISME (UPCASTING/DOWNCASTING)

Par Aïcha El Golli

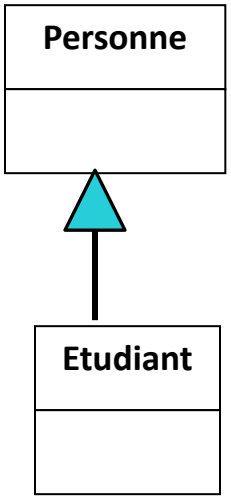
aicha.elgolli@essai.ucar.tn

SURCLASSEMENT OU UPCASTING

La réutilisation du code est un aspect important de l'héritage, mais ce n'est peut-être pas le plus important

Le deuxième point fondamental est la relation qui relie une classe à sa superclasse : ***Une classe B qui hérite de la classe A peut être vue comme un sous-type (sous ensemble) du type défini par la classe A.***

SURCLASSEMENT OU UPCASTING



un étudiant est une personne

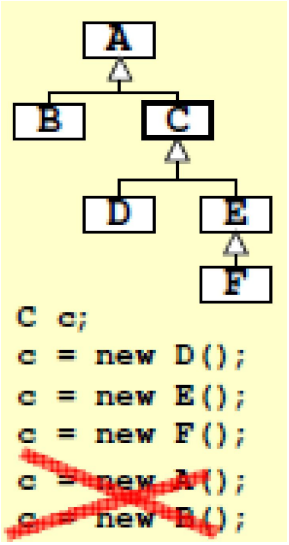
l'ensemble des étudiants est inclus dans l'ensemble des personnes

tout objet instance de la classe B peut être aussi vu comme une instance de la classe A.

cette relation est directement supportée par le langage JAVA : ***à une référence déclarée de type A il est possible d'affecter une valeur qui est une référence vers un objet de type B (surclassement ou upcasting)***

```
Personne p ;
p= new Etudiant(...) ;
```

SURCLASSEMENT OU UPCASTING

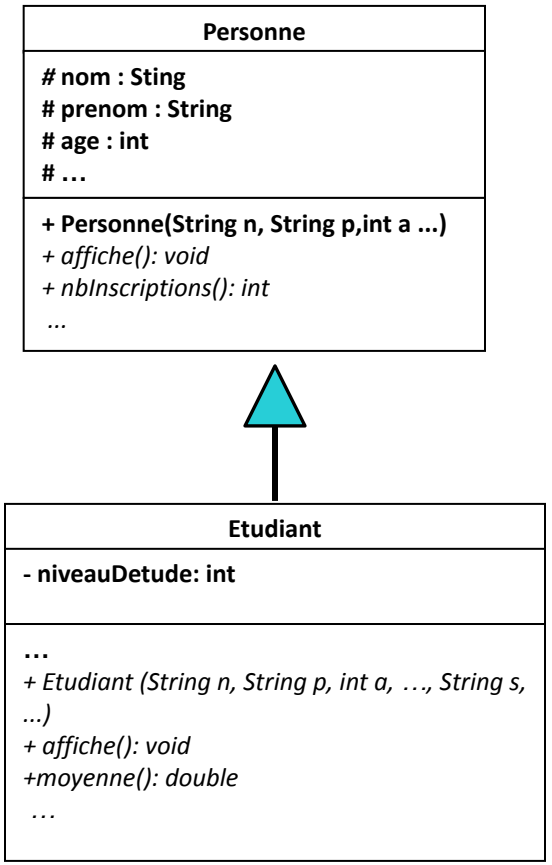


plus généralement à une référence d'un type donné, il est possible d'affecter une valeur qui correspond à une référence vers un objet dont le type effectif est n'importe quelle sous-classe directe ou indirecte du type de la référence

Lorsqu'un objet est "sur-classé" il est vu par le compilateur comme un objet du type de la référence utilisée pour le désigner

Ses fonctionnalités sont alors restreintes à celles proposées par la classe du type de la référence

SURCLASSEMENT OU UPCASTING




```
Etudiant  es;
es = new Etudiant ("DUPONT","Jean",25,..,"troisieme",..);
Personne e;
e = es; //upcasting
e.affiche();
es.affiche();
e.nbInscriptions();
es.nbInscriptions();
es.moyenne ();
```

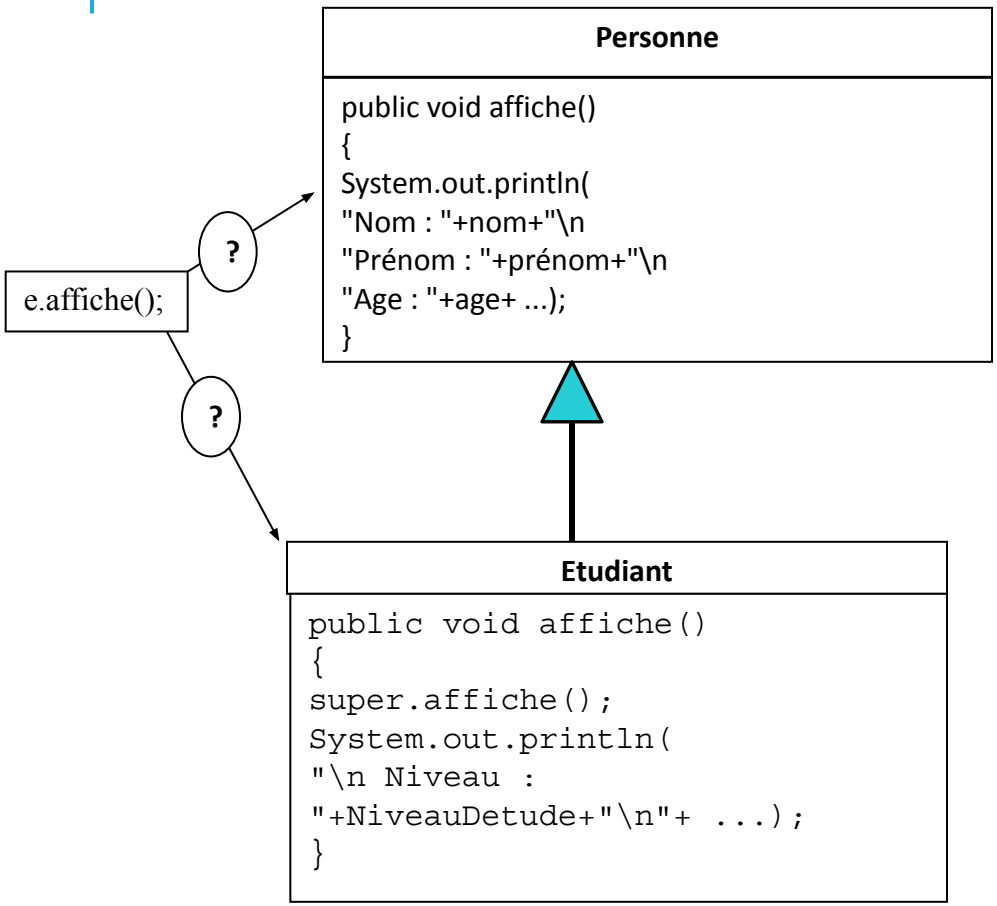
```
e.moyenne();
```

Le compilateur refuse ce message : pas de méthode moyenne() définie dans la classe Personne

…

Résolution des messages : Que va donner e.affiche() ?

SURCLASSEMENT OU UPCASTING



Lorsqu'une méthode d'un objet est accédée au travers d'une référence "surclassée", c'est la méthode telle qu'elle est définie au niveau de la classe effective de l'objet qui est en fait invoquée et exécutée

Nom : DUPONT
Prénom : Jean
Age : 25
...
Niveau : troisieme

LIEN DYNAMIQUE (MÉCANISME DE RÉSOLUTION DES MESSAGES)

Les messages sont résolus à l'exécution

- *la méthode exécutée est déterminée à l'exécution (run-time) et non pas à la compilation*
- *à cet instant le type exact de l'objet qui reçoit le message est connu*
- *la méthode définie pour le type réel de l'objet recevant le message est appelée (et non pas celle définie pour son type déclaré).*

A la compilation : seules des vérifications statiques qui se basent sur le type déclaré de l'objet (de la référence) sont effectuées

- *la classe déclarée de l'objet recevant le message doit posséder une méthode dont la signature correspond à la méthode appelée.*

LIEN DYNAMIQUE (VÉRIFICATIONS STATIQUES)

- à la compilation il n'est pas possible de déterminer le type exact de l'objet récepteur d'un message

```
public class A {
public void m1() {
  System.out.println(" m1 de A "); }
 }
```

```
public class B extends A {
public void m1() {
  System.out.println(" m1 de B "); }
public void m2() {
  System.out.println(" m2 de B "); }
 }
```

```
A obj;
for(int i=0; i<10; i++)
{
   hasard=Math.random();
   if(hasard<0.5)
     obj= new A();
   else
     obj= new B();
   obj.m1();
}
```

LIEN DYNAMIQUE (VÉRIFICATIONS STATIQUES)

vérification statique : garantit dès la compilation que les messages pourront être résolus au moment de l'exécution

```
public class A {
public void m1() {
  System.out.println(" m1 de A "); }
public void m1(int x) {
  System.out.println(" m1(x) de A "); }
 }
```

```
public class B extends A {
public void m1() {
  System.out.println(" m1 de B "); }
public void m2() {
  System.out.println(" m2 de B "); }
 }
```

```
A refA = new A();
refA.m1();
refA.m1(10);
refA = new B();
refA.m1();
```

Le choix de la méthode à exécuter est effectué **statiquement à la compilation** en fonction du type des paramètres

La sélection du code à exécuter est effectué **dynamiquement à l'exécution** en fonction du type effectif du récepteur du message

A QUOI SERVENT L'UPCASTING ET LE LIEN DYNAMIQUE ?

A la mise en œuvre du polymorphisme

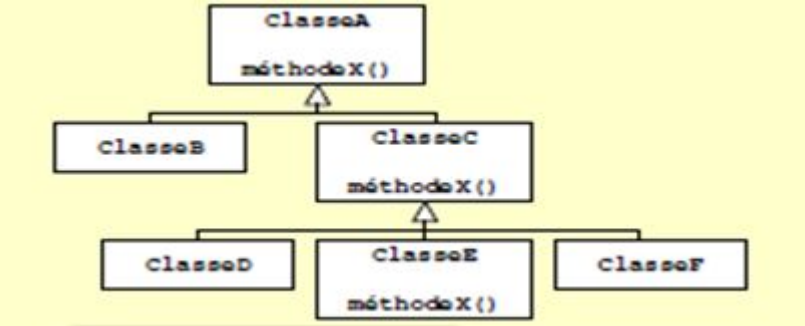
Le terme polymorphisme décrit la caractéristique d'un élément qui peut se présenter sous différentes formes.

En programmation Objet, on appelle polymorphisme

- *le fait qu'un objet d'une classe puisse être manipulé comme s'il appartenait à une autre classe.*
- *le fait que la même opération puisse se comporter différemment sur différentes classes de la hiérarchie.*

Le **polymorphisme** constitue la troisième caractéristique essentielle d'un langage orienté objet après l'abstraction des données (encapsulation) et l'héritage

A QUOI SERVENT L'UPCASTING ET LE LIEN DYNAMIQUE ?




```
ClasseA objA ;
objA =...
objA.methodesX();
```

Surclassement
La référence peut désigner des objets de classe différente (n'importe quelle sous classe de ClasseA

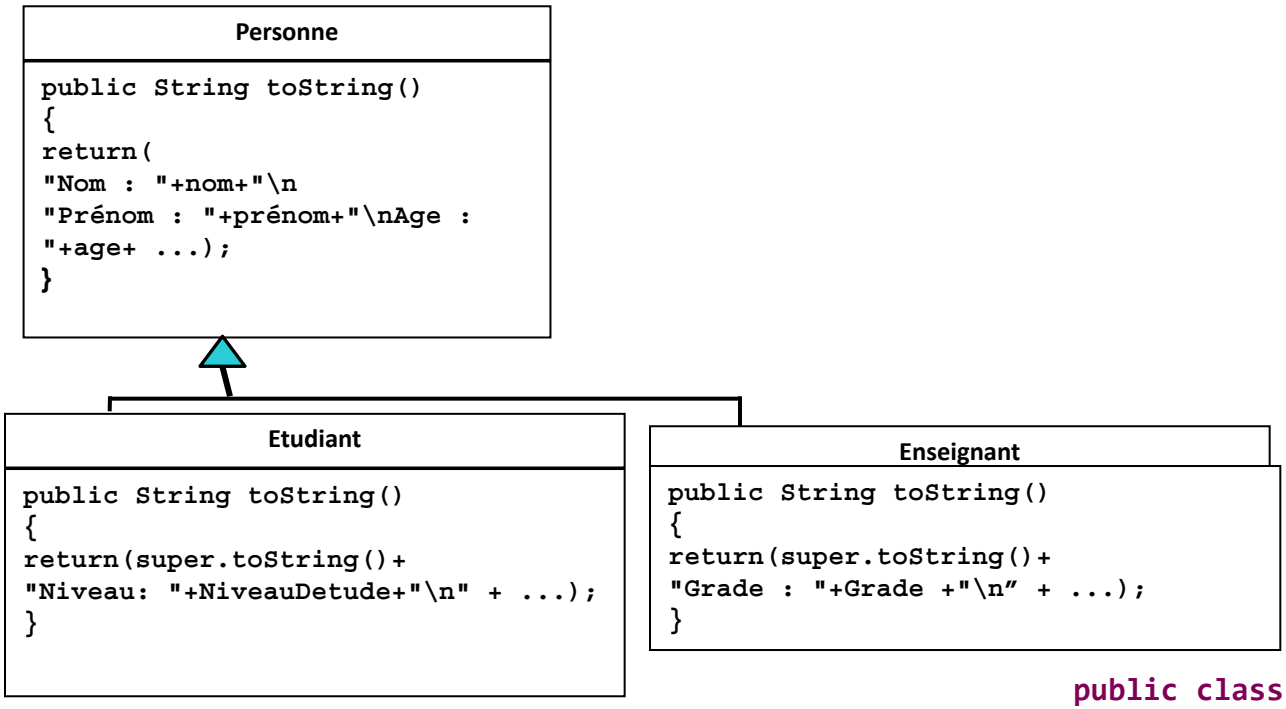
✚

Lien dynamique
Le comportement est différent selon la classe effective de l'objet

Un cas particulier de Polymorphisme

Manipulation **uniforme** des objets de plusieurs classes par l'intermédiaire
d'une classe de base commune

A QUOI SERVENT L'UPCASTING ET LE LIEN DYNAMIQUE



liste peut contenir des personnes de n'importe quel type

```
Groupe td1 = new Groupe ();
td1.ajouter(new Personne("DUPONT", …));
td1.ajouter(new Etudiant("BIDULE", "Louis", … ,
"Deuxieme");
```

Si un nouveau type de personne est défini, le code de Groupe reste inchangé

```
public class Groupe{
Personne [ ] liste = new Personne[30];
int nbEtudiants = 0;

public void ajouter(Personne p)
{ if (nbEtudiants < liste.length)
liste[nbEtudiants++] =p;
}
public void afficherListe()
{for (int i=0;i<nbEtudiants; i++)
System.out.println(liste[i]);
}}
```

A QUOI SERVENT L'UPCASTING ET LE LIEN DYNAMIQUE ?

En utilisant le polymorphisme en association à la liaison dynamique

- *plus besoin de distinguer différents cas en fonction de la classe des objets*
- *possible de définir de nouvelles fonctionnalités en héritant de nouveaux types de données à partir d'une classe de base commune sans avoir besoin de modifier le code qui manipule l'interface de la classe de base*

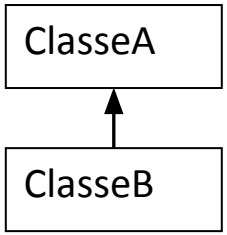
Développement plus rapide

Plus grande simplicité et meilleure organisation du code

Programmes plus facilement extensibles

Maintenance du code plus aisée

SURCHARGE ET POLYMORPHISME




```
public class ClasseC {
public static void methodeX(ClasseA a){
System.out.println("param typeA");  }
public static void methodeX(ClasseB b){
System.out.println("param typeB");  }
}

ClasseA refA=new ClasseA();
ClasseC.methodeX(refA);
ClasseB refB= new ClasseB();
ClasseC.methodeX(refB);
refA=refB;//upcasting
ClasseC.methodeX(refA);
```

choix de la méthode à exécuter est effectué à la compilation en fonction des types déclarés : Sélection Statique

DOWNCASTING

ClasseX obj = …

ClasseA a = (ClasseA) obj;

Le downcasting (ou transtypage) permet de « forcer un type » à la compilation

C'est une « promesse » que l'on fait au moment de la compilation.

Pour que le transtypage soit valide, il faut qu'à l'exécution le type effectif de ***obj*** soit « compatible » avec le type ***ClasseA***

Compatible : la même classe ou n'importe quelle sous classe de ***ClasseA*** *(****obj instanceof ClasseA****)*

Si la promesse n'est pas tenue une erreur d'exécution se produit.

ClassCastException est levée et arrêt de l'exécution