Héritage

Mercredi 14 septembre

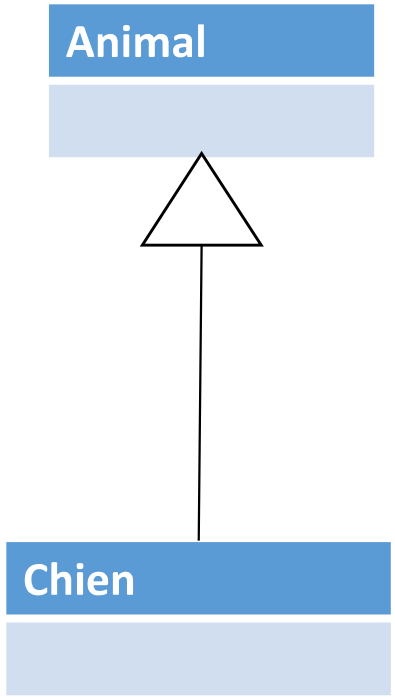
Crée un jar, partager ses projets

- Création du JAR : (concepteur/dev)
  - Creation du projet, avec package et classe (+ tester)
  - Génération du jar et export :
    - Sur eclipse : clic droit sur le projet/export/java/jar
- Création d'un projet qui utilise le JAR : (utilisateur/chef de projet)
  - creation d'un projet avec le main
  - Rajouter le JAR
    - Clic droit sur projet / properties/ java build path/onglet libraries/ addexternal jar
    - Ca crée des « références libraries » dans le projet
  - Import du package : **import monpackage.*;**
  - Utiliser la classe Maclasse.mamethodestatic(); (par exemple)

Héritage : idée

- On veut enrichir une classe avec une autre : une classe A contient de infos et u classe B d'autres types d'info, les deux ensembles pertinents pour une même entité
  - On peut faire avec une association : la classe A possède un B, et a donc accès aux infos contenus dans B
  - On peut utiliser de l'héritage : B est un cas particulier de B et possède les deux ensemble d'info
- Choix du type de lien : en fonction du bon sens
  - Pour l'héritage : si B est une sorte de B (ex : chien/animal)
  - Pour l'association : est composé de (ex : email/piece jointe)
  - Un mauvais choix ne déclenche pas d'erreurs de compil mais donne des non-sen
- Pas d'héritage multiple (existe dans d'autres languages, permet d'hériter de d classes à la fois. pb : ambiguïté possible sur les méthodes)

Symbole : lien d’héritage



C’est quoi l’héritage ?

Cadre : B hérite de A

- Toutes les méthodes de A sont disponible dans B, comme si tout le code de A copié-coller dans B : permet la réutilisation
- Enrichissement : peut ajouter des choses (attributs, méthodes,…) dans la clas qui hérite
- Toutes les classes héritent forcément de la classe Object
- On peut avoir des héritage imbriqués : D hérite de C qui hérite de B qui hérite A (qui hérite de Object). (méthode de cette classe disponible, par ex. le toStri (que l’on écrase par des versions plus lisible quand on le recode)
- Possible de sceller des classes pour empêcher l’héritage (ex : String) par le mo clé **final**
- Héritage successif : on hérite de toutes les classes dont notre classe mère hér (implicitement)

Vocabulaire

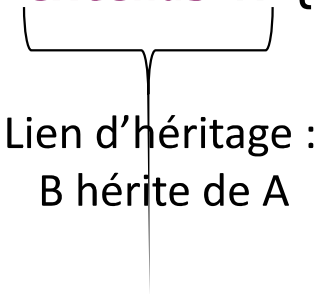
B hérite de A

- B est une spécialisation (un enrichissement) de A
- A est une généralisation de B
- B est la classe dérivée / la classe fille
- A est la classe de base/la classe mère

Comment le coder ?

```
package init;

public class A {

}
```

```
package init;

public class B extends A {

}
```



- Existe dans tous les languages objets, en .NET :
  Class B : A {}

Premier exemple

- Faire un nouveau package avec les classes suivantes :

Classe A :

```
package init;

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

Classe B :

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

Classe Test :

```
package init;

public class Test {

public static void main(String[] args
A monA=new A();
monA.m1();
B monB=new B();
monB.m1();
monB.m2();
}
}
```

De quoi hérite-t-on ?

- De tout ce qui est public : y compris pour appel local comme si le code été présent localement
- Ce qui est privé dans la classe mère ne peut pas être appeler depuis la classe f
- Attribut : privé, ne sont donc pas accessibles depuis les classes filles (mais existent dans l'objet)
- On peut redéfinir (override) des méthodes héritées (ex : toString() que l'on redéfini par rapport à la version existante dans Object)

Protected

- Visibilité protected (s'utilise comme public ou private) : visible pour les classe
  héritent (directement ou transitivement) de la classe où il est défini (et dans l
  package où il est défini)

Classe A :

```
public String toString() {
return "passage par A";
}
```

Classe B :

```
public String toString() {
return super.toString()+" et par B";
}
```

Appel dans le contexte de la classe mère

Dans main :

```
System.out.println(new B());
```

Affiche : passage par A et par B

Constructeurs

Classe X :

```
package init;

public class X {
private int x;

public X(int x) {
super();
this.x = x;
}
public String toString() {
return "x="+x;
}
}
```

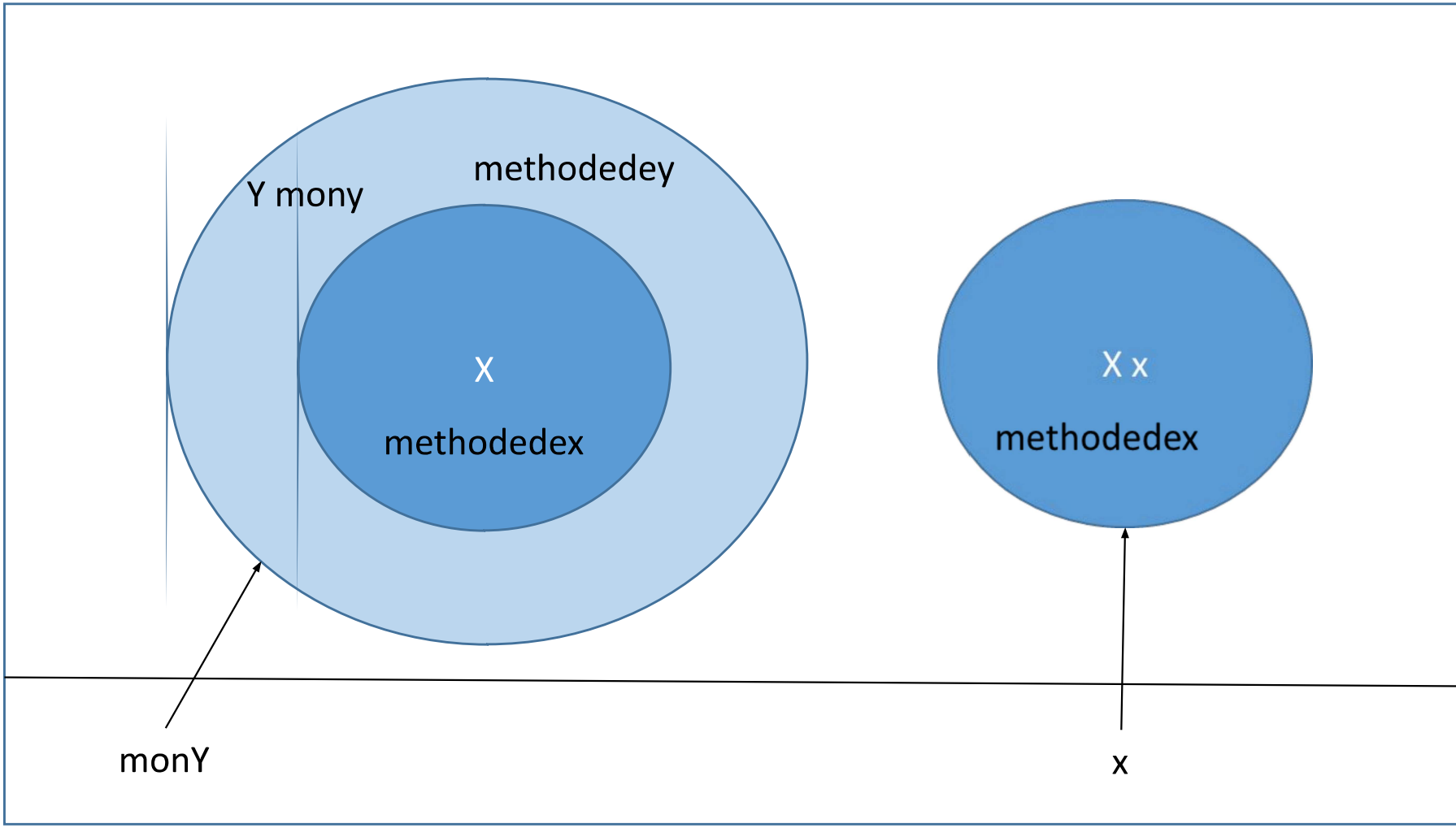
Classe Y

```
package init;

public class Y extends X {
private int y;

public Y(int x, int y) {
super(x);
this.y = y;
}
public String toString() {
return super.toString()+" y="+y;
}
}
```

Il est obligatoire de faire appel au constructeur de X dans Y !

(seul cas différent : le constructeur implicite appelle le constructeur implicte ou par défaut de la sur-classe
marche pas si la surclasse a des constructeurs d'initialisation et pas de constructeur par défaut, ie super()
appeler par défaut au début d'un constructeur)

Schéma mémoire



Déclaration

Y hérite de X

```
X ycache= new Y(14,25);
```

possible de déclarer comme ça : voit alors juste les fonctionalité de X.

Ex : methode m défini uniquement dans Y n’est pas utilisabl via ycache.m(); (erreur de compil)

Polymorphisme :

Animal[] tab = **new Animal[5];**

tab[0]=**new Animal();**

tab[1]=**new Chat();**

tab[2]=**new Chien();**

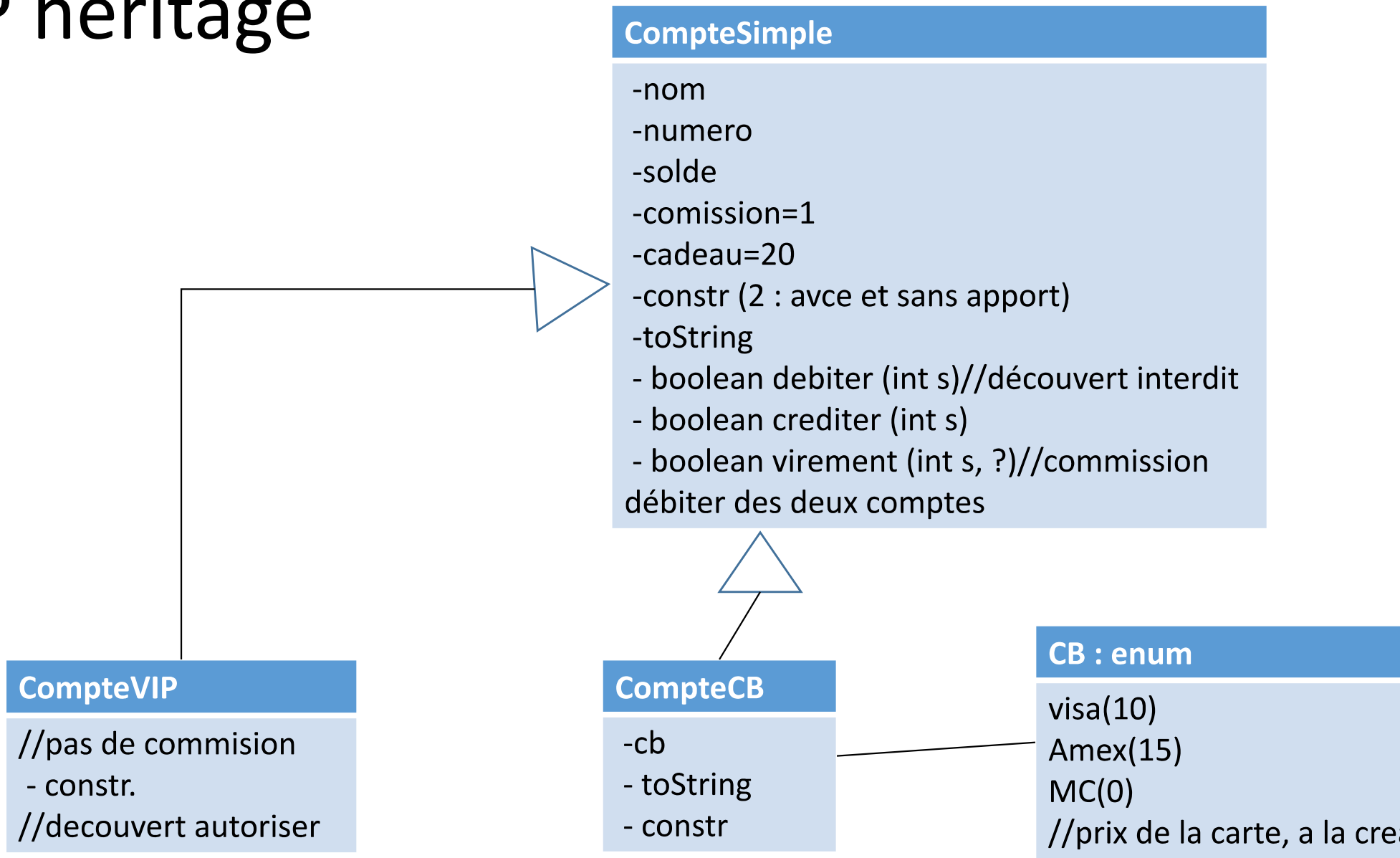
for (Animal A:tab) {

if(A!=null)A.affiche();

}

Une méthode existant sous différentes versions dans chacune des classes liées
héritage permet un traitement spécifique à chaque classe fille.

TP héritage


CompteSimple
-nom
-numero
-solde
-comission=1
-cadeau=20
-constr (2 : avce et sans apport)
-toString
- boolean debiter (int s)//découvert interdit
- boolean crediter (int s)
- boolean virement (int s, ?)//commission
débiter des deux comptes
CompteVIP
//pas de commision
- constr.
//decouvert autoriser
CompteCB
-cb
- toString
- constr
CB : enum
visa(10)
Amex(15)
MC(0)
//prix de la carte, a la cre