Cours 5+6

- Héritage
- Classe abstraite
- Interface

On définit une classe dérivée à partir d'une classe existante comme une extension de celle-ci :
super classe, classe de base, classe mère

la classe dérivée hérite de la classe de base
- toutes les méthodes publiques
- toutes variables (attributs) instance et statiques

le code est réutilisé sans copier

- la classe dérivée peut définir de nouvelles méthodes et variables
- la classe dérivée peut redéfinir les variables ou les méthodes de la classe de base

Réutiliser une classe existante en l'adaptant et factoriser le code

Définition d'une classe dérivée

```
class Point
{private int x,y;
public Point(int a, int b){x=a;y=b;}
public void modif(int coef){x=x*coef;  y=y*coef;}
public void affiche_pt()
{System.out.println("coordonnees du point :"+x+"  "+y);}
}
class Point_Couleur extends Point
{private String couleur;
public Point_Couleur(int a, int b, String c)
{super(a,b); couleur=c;}
public void affiche_pt(){super.affiche_pt();
System.out.println("et sa couleur " +couleur);}
}
public class point_heritage{
public static  void main(String args[])
{ Point pt1=new Point (-44,6);
pt1.affiche_pt();  pt1.modif(2); pt1.affiche_pt();
Point_Couleur ptc=new Point_Couleur (3,4,"blue");
ptc.affiche_pt();  ptc.modif(2);  ptc.affiche_pt();
}}
```

Dans Java 5.0, il est possible de changer le type de la valeur retournée s'il s'agit d'un type classe

```
public class BaseClass
{ . . .
  public Employee getSomeone(int someKey)
  . . .
public class DerivedClass extends BaseClass
{ . . .
  public HourlyEmployee getSomeone(int someKey)
```

classe de base :private void doSomething()
classe dérivée : public void doSomething()

L'inverse n'est pas possible

classe de base:public void doSomething()
classe dérivée :private void doSomething()

•On peut définir plusieurs niveaux d'héritages et l'héritage est transitive

•Il peut y avoir une seule classe de base

•Toute classe est dérivée de la classe racine Objet définie dans java.lang

•On peut interdire qu'une classe soit étendue

```
final class A{
}
```

•Tout constructeur autre que la classe Objet fait appel
- •soit à un constructeur de sa classe de base (super ())
- •soit à un autre constructeur de sa classe (this())
- •si il n'y a pas ceci en première ligne, le compilateur ajoute super(); en première ligne du constructeur

On invoque le constructeur de la classe de base par **super**

```
public derivedClass(int p1, int p2, double p3)
  {
    super(p1, p2);
    instanceVariable = p3; }
```

super

•la première instruction

•s'il n'existe pas dans le constructeur de la classe dérivée, le constructeur de la classe de base est invoquée (erreur s'il n'existe pas)

• il faut invoquer pour initialiser les variables d'instance

this

pour invoquer un autre constructeur dans la même classe

super.methode(); this.methode();

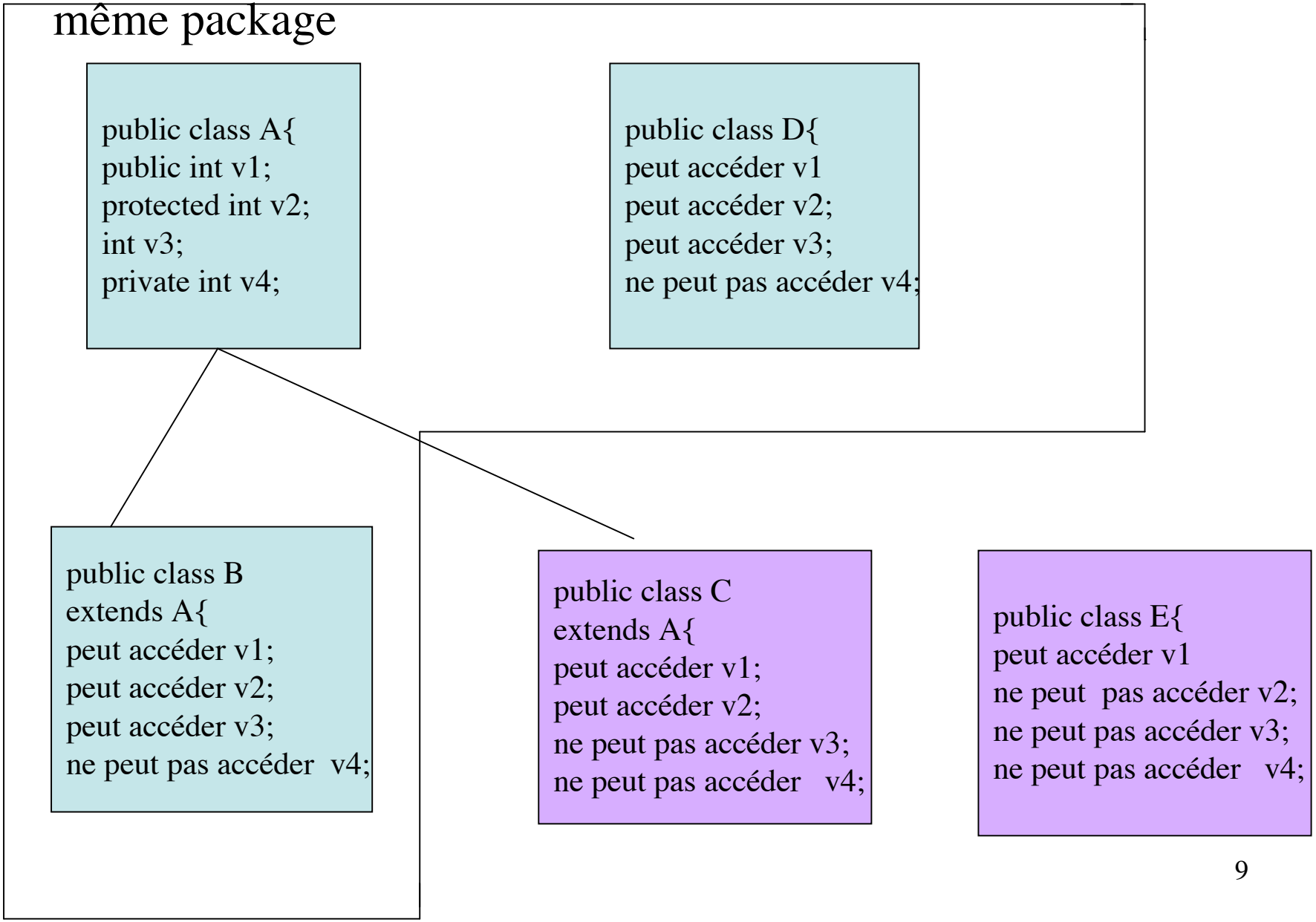
```
import java.io.*;
class A{
public A()
{System.out.println("constructeur de A");}
}
class B extends A {
public B()
{System.out.println("constructeur de B");}
public B(int a)
{this();System.out.println("autre constructeur de B");}
}
class C extends B{
public C()
{super(3);System.out.println("constructeur de C");}
}
public class Tst_const
{public static  void main(String args[])
{ C c= new C();}
}
```

constructeur de A
constructeur de B
autre constructeur de B
constructeur de C

```
import java.io.*;
class A{
public void affiche()
{System.out.println("je suis un A");}
}
class B extends A {}
class C extends A
{public void affiche()
{System.out.println("je suis un C");}
}
class D extends C{
public void affiche()
{System.out.print("je suis un D");super.affiche();}
}
class E extends B{}
class F extends C {}
public class deriv_heritage
{public static  void main(String args[])
{ A a= new A(); a.affiche();
  B b= new B(); b.affiche();
  C c= new C(); c.affiche();
  D d= new D(); d.affiche();
  E e= new E(); e.affiche();
  F f= new F(); f.affiche();}
}
```

```
je suis un A
je suis un A
je suis un C
je suis un D  je suis un C
je suis un A
je suis un C
```

MODES d'accès



Un objet d'une classe dérivée est de type de classe dérivée et aussi de type de classe de base

d'une façon générale, il est de type de toutes ses classes antérieures (ascendantes)

Un objet d'une classe dérivée peut être affecté à une variable de type de chacune de ses classes antérieures

Un objet d'une classe dérivée peut être utilisé à la place d'un objet de chacune de ses classes antérieures.

Un objet d'une classe de base ne peut pas être utilisé à la place d'un objet de ses classes dérivées.

Conversion et Transtypage

•La conversion d'un type ordinaire vers un autre type ordinaire

implicitement vers un type « plus grand »

explicitement vers un type « plus petit »

double pi=3.14; int p; short q=3;

p=q; p=(int) pi;

•transtypage d'une variable de type « classe »

en une autre de type « classe »

les seules conversions possibles concernent les variables qui référencent des objets d'une hiérarchie des classes construites par héritage et dans le sens ascendant. (*conversion implicite*)

conversion explicite vers une classe descendant

(Attention, le compilateur ne vérifie pas la légitimité de cette conversio

```
 A a= new A();
B b= new B();
C c= new C();
D d= new D();

a=c; a=d;

d=a; // impossible


A a1=new A();
b =(B) a1; //compilation possible mais exécution impossible

a1=b; b =(B) a1; //exécution possible car a1 est un B


                                    }
```

Classe Object

Chaque classe est une classe dérivée de la classe **Object**
définie dans le package java.lang importé automatiquement

Tout objet de n'importe quelle classe est de type Object

Un paramètre de type Object peut être remplacé par
un objet de n'importe quelle classe

Il est ainsi possible de définir des méthodes générales dans les bibliothèques (packages)

Les méthodes définies dans la classe Object sont héritées par toute classe

public boolean equals (Object objet_a_comparer)
public String toString()
public int hashcode()
protected Object clone()
protected void finalize

constructeur Object();

```
public boolean equals(Object autreObjet)
{
  if (autreObjet == null)
    return false;
  else if (getClass( ) != autreObjet.getClass( ))
    return false;
  else
  {
    Personne autrePersonne = (Personne)autrePersonne;
    return ((nom.equals(autrePersonne.nom)) &&
      (prenom.equals(autrePersonne.prenom));
  }
}
```

objet instanceof Classname
teste si l'objet appartient à la classe Classname

true si l'objet est de type Classname
(la classe Classname ou une classe dérivée de cette classe

```
if (autrePersonne instanceof Personne)
.....
else ...
```

getclass() est une méthode de la classe Object

teste les classes utilisées pendant la création (new)

```
(objet1.getClass() == objet2.getClass())
```

Polymorphisme est une notion liée à la redéfinition des méthodes et à la liaison dynamique.

Une méthode polymorphe est une méthode déclarée dans une super-classe et redéfinie dans une sous-classe

```
public class deriv_heritage1
{public static  void main(String args[])
{
  A a= new A(); a.affiche();
  B b= new B(); b.affiche();
  a=b; a.affiche();
  C c= new C(); c.affiche();
  a=c; a.affiche();
  D d= new D(); d.affiche();
  a=d; a.affiche();
  c=d; c.affiche(); E e= new E(); e.affiche();
  a=e; a.affiche();
  b=e; b.affiche();
  F f= new F(); f.affiche();
  a=f; a.affiche();
  c=f; c.affiche();
 }}
```

Polymorphisme est une notion liée à la redéfinition des méthodes et à la liaison dynamique.

Une méthode polymorphe est une méthode déclarée dans une super-classe et redéfinie dans une sous-classe

```
public class deriv_heritage1
{public static  void main(String args[])
{
  A a= new A(); a.affiche();
  B b= new B(); b.affiche();
  a=b; a.affiche();
  C c= new C(); c.affiche();
  a=c; a.affiche();
  D d= new D(); d.affiche();
  a=d; a.affiche();
  c=d; c.affiche(); E e= new E(); e.affiche();
  a=e; a.affiche();
  b=e; b.affiche();
  F f= new F(); f.affiche();
  a=f; a.affiche();
  c=f; c.affiche();
 }}
```

```
je suis un A
je suis un A
je suis un A
je suis un C
je suis un C
je suis un D je suis un D je suis un D je suis un A
 je suis un A
 je suis un A
je suis un C
 je suis un C
 je suis un C
```

```
class Orateur{
String action(){return "parle";}
}
class Grenouille  extends Orateur{
String action(){return "coasse";}
}
class Fourmi extends Orateur{
String action(){return "croonde";}
}
public class Tst_Orateur{
public static  void main(String args[]){
Orateur  p = new Orateur();
System.out.println("orateur "+ p.action());
Grenouille q= new Grenouille();
System.out.println("grenouille "+ q.action());
p=q; System.out.println("orateur "+ p.action());
Orateur  x= new Fourmi();
System.out.println("orateur "+ x.action());}
}
```

orateur parle
grenouille coasse
orateur coasse
orateur croonde

```
class Felin {
        boolean a_faim = true;
         void parler() { }
         void appeler() {
                 System.out.println("minou minou,…");
                 if (a_faim) parler();}
}

 class Chat extends Felin {
         String race;
         void parler() { System.out.println("miaou! "); }
}

 class Lion extends Felin {
         void parler() { System.out.println("roar! "); }
          void chasser() {System.out.println("lion est chasseur");}
}
```

```
public class Tst_transtypage {
public static void main( String[] args)
        {  Lion lion_obj = new Lion();
        Felin felin_obj;
        felin_obj  = lion_obj; // OK conversion implicite :
        // les lions sont des félins
        //lion_obj = felin_obj;// ERREUR de Compil :
         //tous les félins ne sont pas des lions
         felin_obj.parler();   // « roar ! »
        //felin_obj.chasser(); // Méthode introuvable dans classe Felin

        lion_obj = (Lion)felin_obj;   // Conversion descendante explicite
        lion_obj.parler();    // « roar ! »
        lion_obj.chasser();   // OK

        Chat chat_obj = new Chat();
        felin_obj = chat_obj;          // Conversion ascendante

        lion_obj = (Lion)felin_obj;   // ERREUR java ClassException

        }

}
```

```
class Point{
protected int x,y;
public Point (int a, int b)    {x=a; y=b;}
public void affiche()
{System.out.println("coord:"+x +" " +y);}
}
class Point_Colore extends Point{
private String couleur;
public Point_Colore(int a, int b, String c)
{super(a,b); couleur=c;}
public void affiche()
{System.out.println("coord:"+x +" " +y+  "couleur :"+couleur);}
}
public class Tstranstypage{
public static  void main(String args[]){
Point pp= new Point(4,7);
Point_Colore ppcol=new Point_Colore(7,9, "vert");
 pp=ppcol; pp.affiche();
 // ppcol=pp interdit  Object o1;   Point p1;
 o1=new Point_Colore(1,8,"rouge");  p1= (Point_Colore)o1;
p1.affiche();
//OK compilation, erreur execution dans la suite
Point p2=new Point(5,5);  Point_Colore pc2= (Point_Colore)p2;
 pc2.affiche();  }}
```

• Différentes exécutions d'une même méthode:

si pp , variable de type Point, a pour contenu la référence à un objet de type Point; pp.affiche() déclenche l'exécution de la méthode de la classe Point.

pp=ppcol; pp a pour contenu la référence d'un objet de type Point_colore; pp.affiche() déclenche l'exécution de la méthode de la classe Point_Colore.

•Un tableau peut contenir des objets appartenant à des classes d'un même arbre d'héritage

•Le type Object peut être utilisé pour référencer n'importe quel type d'objet

Classes Abstraites

•Une méthode abstraite est définie uniquement par son intitulé sans code
•Une classe abstraite est une classe dont au moins une méthode est abstraite

```
abstract class figure
{protected public double long;
abstract void calcul_long();}
```

•Dans les classes dérivées, la méthode abstraite doit être décrite
• La méthode abstraite ne peut pas être private
•On ne peut pas créer un objet d'une classe abstraite
•Une classe abstraite est un type, il est possible de définir des variables ou paramètres de ce type; mais il faut affecter un objet d'une classe concrète, descendante

```
abstract class Figure{
protected double longueur;
abstract public  void  calcul_long();
}
class Segment extends Figure
{private int xA, yA, xB, yB;
public Segment(int xAp,int yAp,int xBp,int yBp)
{xA=xAp; yA=yAp;xB=xBp; yB=yBp;longueur=0;}

public void calcul_long(){
longueur=Math.sqrt((xB-xA)*(xB-xA)+
(yA-yB)*(yA-yB));
System.out.println("long "+longueur);}
}
class Cercle extends Figure{private int rayon;
public Cercle( int r){rayon=r;longueur=0;}
public void calcul_long(){
longueur=2*3.14*rayon;System.out.println("long "+longueur);}
}public class Figabs{
public static  void main(String args[]){
Segment s=new Segment(1,3,4,5);s.calcul_long();
Cercle c=new Cercle(3);c.calcul_long();}}
```

Interface

Une interface est une classe abstraite ayant les caractéristiques suivants:

- toutes les méthodes sont abstraites et public,
  alors qu'une classe abstraite peut avoir des méthodes non abstraites
- toutes les variables sont static et constantes déclarées par le modificateur final, alors qu'une classe abstraite peut avoir des variables ordinaires
- toute classe peut hériter d'une seule classe mais elle
  peut hériter de plusieurs interfaces
- une interface peut hériter d'une ou plusieurs interfaces mais elle ne peut pas hériter d'une classe

- il n'est pas nécessaire d'indiquer abstract pour les méthodes et static final pour les variables
- une interface définie par le mot **interface**
- si une classe hérite d'une interface **implements**

interface I

liste de constantes;

liste de prototypes de méthodes

class A extends B implements I{ }

class A implements I1,I2 { }

interface I extends I1,I2 { }

```
interface Itf_calcul{
double[] valeur={33.2, 13.2, 5.8};
public void calcul_prime();}
class Pers_admin implements Itf_calcul{
private String nom;  private String service;
private int categorie; private int age; private double prime;
public Pers_admin(String n, String s, int c, int a){
nom=n; service=s; age=a;categorie=c;}
public void calcul_prime(){prime=valeur[categorie]*age;}
public void affiche() {
System.out.println("nom: "+nom+" service: "+service+ " prime "+prime);}
}
class Pers_exterieure implements Itf_calcul{
private String nom; private String entreprise;
private int categorie;  private double prime;
public Pers_exterieure(String n, String s, int c){
nom=n; entreprise=s; categorie=c;}
public void calcul_prime(){ prime=valeur[categorie]*20;}
 public void affiche() {
System.out.println("nom: "+nom+" entreprise:"+entreprise+" prime: "+prime)
public class Tstinterface{
public static  void main(String args[]){
 Pers_admin pa= new Pers_admin("Dupont","comptabilite",2,40);
pa.calcul_prime();  pa.affiche();
Pers_exterieure pe=new Pers_exterieure("Bernard", "EDF",1);
 pe.calcul_prime();  pe.affiche();}}
```