POO 2024

En Java

Univ Jijel

A. Brighen

3 Héritage

Bases de POO

Héritage: concepts

L'héritage, est l'un des mécanismes les plus puissants de la POO. Il permet la réutilisation des fonctionnalités d'une classe (appelée ***superclasse*** ou ***classe mère***,), tout en apportant des variations, spécifiques aux classes héritant (appelée ***sous-classe***, ***classe fille***, *classe* ***enfant***, *ou classe* ***dérivée***).

L'objectif principal de l'héritage est de favoriser la **réutilisabilité** et **d'éliminer la redondance** (de code).

une classe enfant obtient les **caractéristiques** de sa superclasse (classe mère)

3

Bases de POO

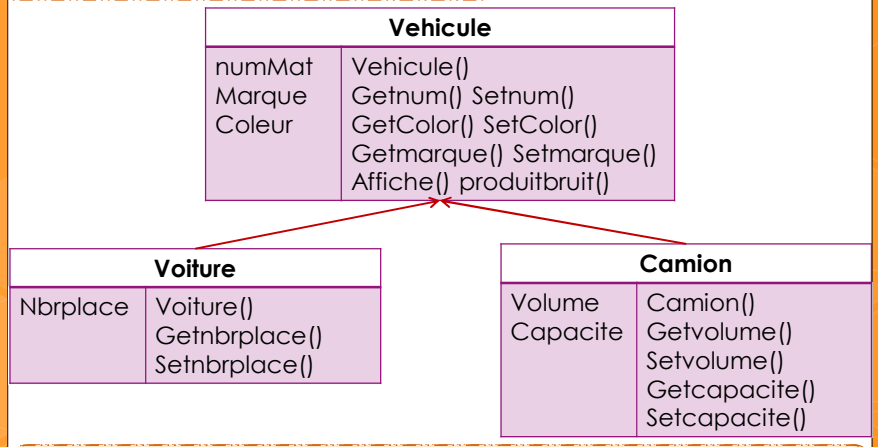
Héritage: concepts

permet de créer une nouvelle classe à partir d'une classe existante en lui proférant ses **propriétés** et ses **méthodes**

4

Bases de POO

Héritage: concepts



Il permet d'établir une relation « **est-un** ».

Voiture « **est un** » Véhicule, Camion « **est un** » vehicule

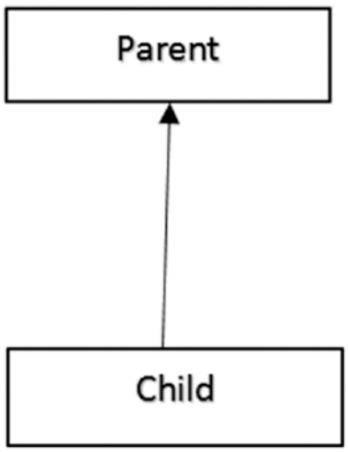
5

Bases de POO

Héritage: concepts

Quatre types d'héritages:

1- Héritage unique (*Single inheritance*): Une classe enfant est dérivée d'une classe de base.



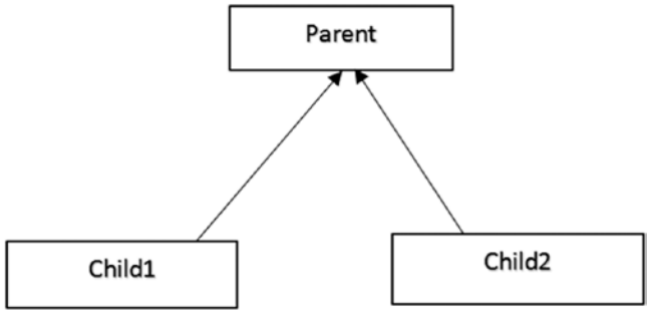
6

Bases de POO

Héritage: concepts

Quatre types d'héritages:

2- Héritage hiérarchique: Plusieurs classes enfants peuvent être dérivées d'une superclasse



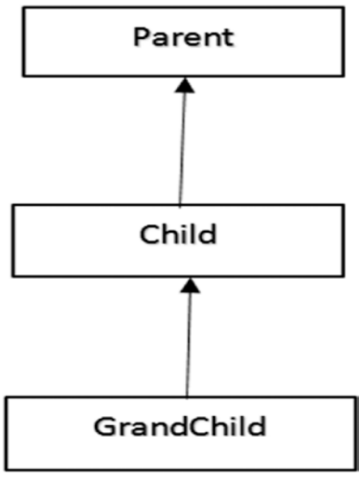
7

Bases de POO

Héritage: concepts

Quatre types d'héritages:

3- Héritage à plusieurs niveaux : la classe parente peut avoir des sous-sous classes.



8

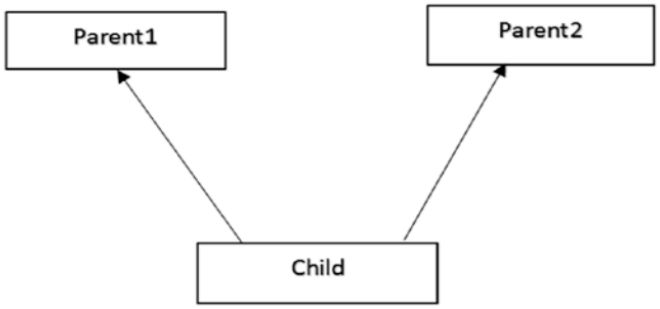
Bases de POO

Héritage: concepts

Quatre types d'héritages:

3- Héritage multiple: la classe dérivé peut dériver de plusieurs superclasses

Ce type d'héritage n'est pas autorisé par Java



9

Bases de POO

Héritage: en Java

Comment une classe hérite-t-elle d'une autre classe en Java?

On utiliser le mot clé **extends** suivi du nom de la **superclasse** dans la déclaration de la classe dérivée.

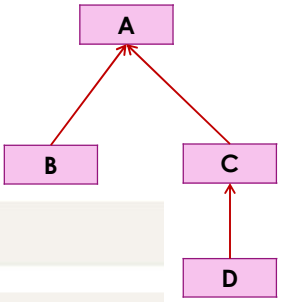
La syntaxe générale est la suivante :

```
1 public class subclass extends superclass{
2
3 }
```

10

Bases de POO

Héritage: en Java




```java
public class A {

}
```

```java
public class B extends A {

}
```

```java
public class C extends A {

}
```

```java
public class D extends C {

}
```

11

Bases de POO

Héritage: droit d'accès

Un **membre** (classes, attributs, méthodes) d'une classe C peut être :

private : accessible seulement à la classe ;

par défaut (package friendly): valeur par défaut, accessible aux classes dans le même paquetage ;

protected : accessible aux classes dans le même paquetage et à toute classe **dérivée** en dehors du paquetage;

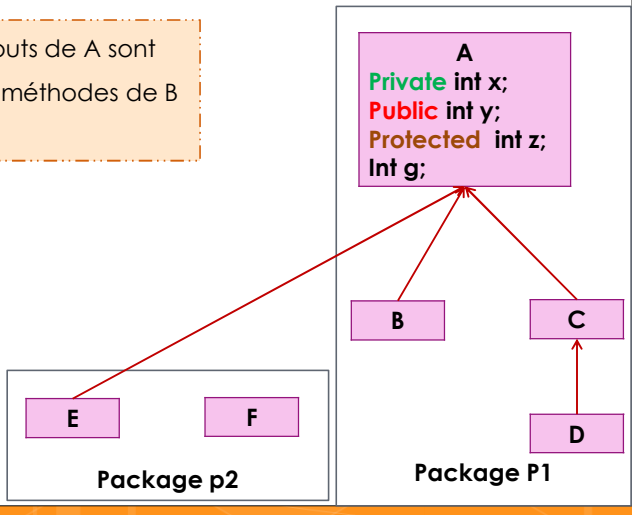
public : accessible à toutes les classes.

12

Bases de POO

Héritage: droit d'accès

Est-ce que les attributs de A sont accessibles par les méthodes de B et C et D??

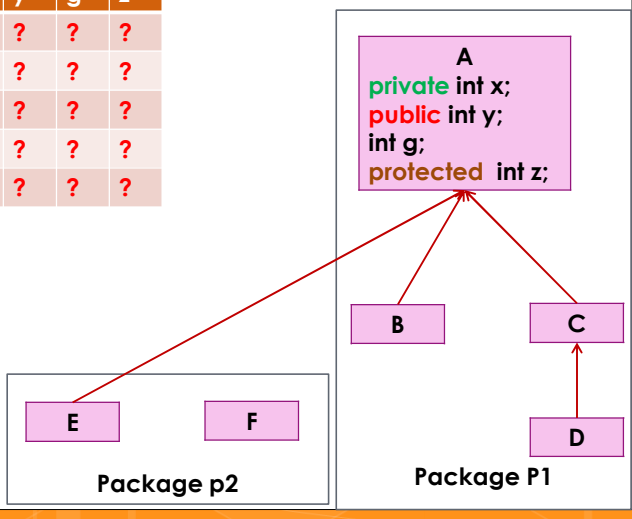


13

Bases de POO

Héritage: droit d'accès

Accessible?	x	y	g	z
B	?	?	?	?
C	?	?	?	?
D	?	?	?	?
E	?	?	?	?
F	?	?	?	?

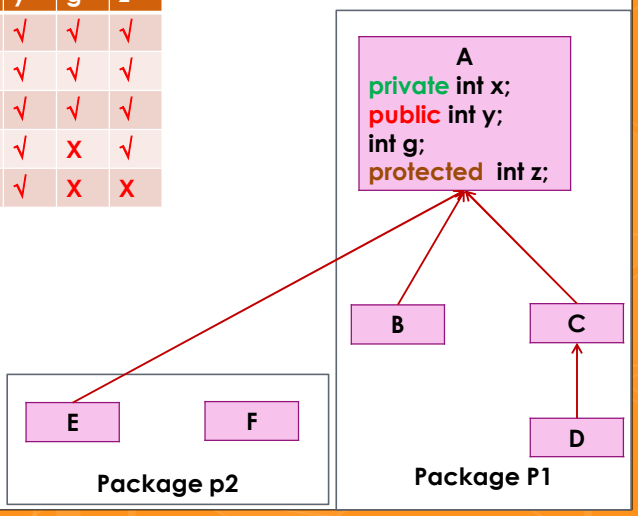


14

Bases de POO

Héritage: droit d'accès

Accessible?	x	y	g	z
B	X	√	√	√
C	X	√	√	√
D	X	√	√	√
E	X	√	X	√
F	X	√	X	X



Bases de POO

Héritage: droit d'accès

```java
package P1
public class A {
private int x;
public int y;
int g;
protected  int z;

}
```

```java
package P1
public class B extends A {
void mB{
x=10; y=12; g=6; z=4;
    }
}
```

```java
package P1
public class C extends A {
void mC{
x=10; y=12; g=6; z=4;
    }
}
```

```java
package P1
public class D extends C {
void mD{
x=10; y=12; g=6; z=4;
    }
}
```

```java
package P2
import P2.A;
public class E extends A {
void mE{
x=10; y=12; g=6; z=4;
    }
}
```

```java
package P2
import P2.A;
public class F {
void mF{
x=10; y=12; g=6; z=4;
    }
}
```

Bases de POO

Héritage: droit d'accès

```java
package P1
public class A {
private int x;
public int y;
int g;
protected  int z;

}
```

```java
package P1
public class B extends A {
void mB{
x=10; y=12; g=6; z=4;
    }
}
```

```java
package P1
public class C extends A {
void mC{
x=10; y=12; g=6; z=4;
    }
}
```

```java
package P1
public class D extends C {
void mD{
x=10; y=12; g=6; z=4;
    }
}
```

```java
package P2
import P2.A;
public class E extends A {
void mE{
x=10; y=12; g=6; z=4;
    }
}
```

```java
package P2
import P2.A;
public class F {
void mF{
x=10; y=12; g=6; z=4;
    }
}
```

17

Bases de POO

Héritage: Spécialisation et Masquage

Lorsqu'une classe enfant désire **modifier l'implémentation** d'une méthode d'une **superclasse**, il lui suffit de redéfinir cette méthode.

La redéfinition d'une méthode est nécessaire si on désire adapter son action à des besoins spécifiques.

La **redéfinition** d'une méthode, aussi appelée « **overriding** » elle consiste à donner une nouvelle implémentation à une méthode héritée **sans changer sa signature** :

Le même nom de méthode

Même type de retour

Même paramètre (nombre, ordre, et type)

18

Bases de POO

Héritage: Spécialisation et Masquage

```java
public class Vehicule {
  int num;
    String marque;
    String color;
    public Vehicule () {
        //code de constructeur
    }

    public void describe () {
System.out.println("Un véhicule: "+num+" "+ color+" "+" "+marque);
    }
}
```

```java
public class Voiture extends  Vehicule{

    public Voiture () {
        //code de constructeur
    }

    public void describe () {
System.out.println("Une voiture: "+num+" "+ color+" "+" "+marque);
    }
}
```

19

Bases de POO

Héritage: Spécialisation et Masquage

```java
class Pet {
private String name;
public String getName( ) {
return name;
}
public void setName(String petName) {
name = petName;
}
public String speak( ) {
return "I'm your cuddly little pet.";
}
}
```

```java
Pet myPet = new Pet( );
System.out.println(myPet.speak());
```

```java
class Cat extends Pet {
public String speak( ) {
return "Don't give me orders.\n" +
"I speak only when I want to.";
}
}
```

```java
Cat myCat = new Cat( );
myCat.setName("mi mi");
System.out.println(myCat.getName( ) + " says: ");
System.out.println(myCat.speak( ));
```

20

Bases de POO

Héritage: Spécialisation et Masquage

```java
class Pet {
private String name;
public String getName( ) {
return name;
}
public void setName(String petName) {
name = petName;
}
public String speak( ) {
return "I'm your cuddly little pet.";
}
}
```

```java
Pet myPet = new Pet( );
System.out.println(myPet.speak());
```

```java
class Dog extends Pet {
public String speak( ) {
return super.speak +
"I speak whenever you want to.";
}
}
```

```java
Dog myDog = new Dog( );
myDog.setName("Li Li");
System.out.println(myDog.getName( ) + " says: ");
System.out.println(myDog.speak( ));
```

21

Bases de POO

Héritage: Spécialisation et Masquage

```java
package b;
public class C extends B{
    int k;
    C(){
        //constructor
    }
    public int f(){
        return super.f()+ 5;
    }
}
```

```java
package b;
public class B {
    int b;
    B(){
    //constructeur
    }
    public int f(){
        return 4;
    }
}
```

```java
class D extends B{
    public int f(){
        return super.f()* 5;
    }
}
```

```java
package b;
public class Main {
    public static void main(String[] args) {
        B a =new B();
        System.out.println(a.f());
        C c=new C();
        System.out.println(c.f());
        D d=new D();
        System.out.println(d.f());

    }
}
```

```
run:
a.f= 4
c.f= 9
d.f= 20
```

22

Bases de POO

Héritage: Spécialisation et Masquage

```
package b;
public class C extends B{
    int b=4;
    C(){
       //constructor
    }
     public int f(){
       return b;
     }
}

class D extends B{
    int b=2;
   public int f(){
    return b;
     }
}

class E extends B{
    int b=2;
   public int f(){
    return super.b+b;
     }
}
```

```
package b;
public class B {
int b=6;
B(){
//constructeur
}
public int f(){
  return b;
  }
}
```

```
run:
a.f= 6
c.f= 4
d.f= 2
d.f= 8
```

```
package b;
public class Main {
    public static void main(String[] args) {
       B a =new B();
      System.out.println("a.f= " +a.f());
       C c=new C();
      System.out.println("c.f= " +c.f());
       D d=new D();
      System.out.println("d.f= " +d.f());
       E e=new E();
      System.out.println("d.f= " +e.f());
    }}
```

23

Bases de POO

Héritage: Spécialisation et Masquage

super.methode_name(): Lorsque la classe parent et la classe dérivées ont le même nom de méthode(redéfinition ou la surcharge), on utilise le mot clé java **super** dans la classe enfant pour accéder à la méthode de la **superclasse**.

super.attribut_name: Lorsque le parent et les classes dérivées ont le même nom de variable, on peut utiliser le mot-clé java **super** pour accéder la variable de la **superclasse**.

Cette écriture permet **d'établir un lien** entre le nom du membre et la classe à laquelle il appartient, et permet **d'éviter la duplication** de code lors la spécialisations de méthodes.

24

Bases de POO

Héritage: Constructeur et super()

Contrairement aux autres membres d'une superclasse, les **constructeurs** d'une superclasse **ne sont pas hérités** par ses sous-classes. Cela signifie qu'on doit définir un constructeur pour une classe enfant ou utiliser le constructeur par défaut ajouté par le compilateur.

Donc, chaque attribut doit être initialisé dans la classe où il est explicitement définit.

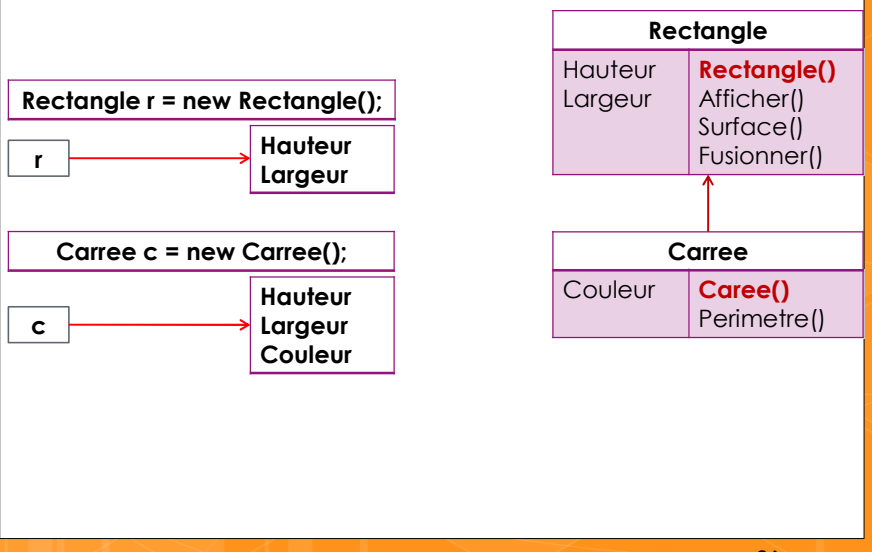
Un constructeur d'une sous-classe peut faire appel au constructeur de la super classe, utilisant l'instruction ***super()*** ou ***super(args)***

Si le constructeur définit d'une sous classe ne contient pas un appel explicite à un constructeur de la superclasse, le compilateur ajoute l'instruction ***super()*** comme **première instruction du constructeur**.

25

Bases de POO

Héritage: Constructeur et super()



26

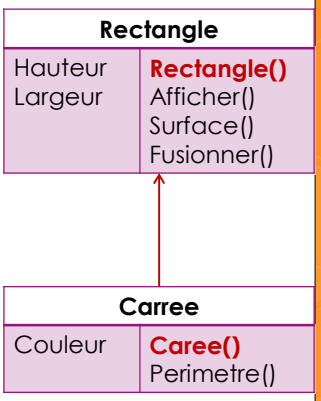
Bases de POO

Héritage: Constructeur et super()

```
public class Rectangle {
    private float h;
    private  float l;
public Rectangle(float a,float b){h=a; l=b;}
public Rectangle(){ h=5; l=10;}

public void afficher(){
System.out.println("h= "+h+" l= "+l);
System.out.println("surface= "+surface());}

float surface(){return h*l;}
```

```
public class Carree extends Rectangle{
    int couleur;
Carree(){super(); couleur=1;}
Carree(float l, int c){super(l,l); couleur=c;}
}
```

```
Carree c=new Carree();
c.afficher(); //h= 5.0 l= 10.0
              //surface= 50.0
        }
}
```



27

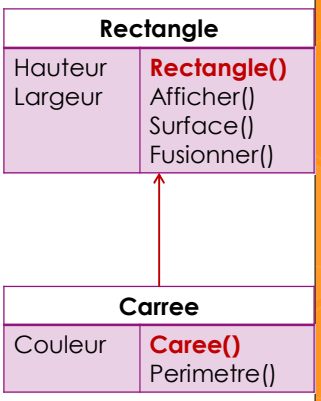
Bases de POO

Héritage: Constructeur et super()

```
public class Rectangle {
    private float h;
    private  float l;
public Rectangle(float a,float b){h=a; l=b;}
public Rectangle(){ h=5; l=10;}

public void afficher(){
System.out.println("h= "+h+" l= "+l);
System.out.println("surface= "+surface());}

float surface(){return h*l;}
```

```
public class Carree extends Rectangle{
    int couleur;
Carree(){ couleur=1;}
Carree(float l, int c){super(l,l); couleur=c;}
}
```

```
public class Carree extends Rectangle{
    int couleur;
Carree(){ couleur=1;}
Carree(float l, int c){ couleur=c;}
}
```




```
Carree c=new Carree();
c.afficher(); //??????
```

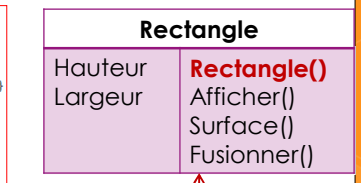
28

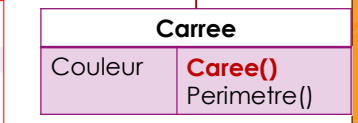
Bases de POO

Héritage: Constructeur et super()

```
public class Rectangle {
    private float h;
    private  float l;
public Rectangle(float a,float b){h=a; l=b;}
public Rectangle(){ h=5; l=10;}

public void afficher(){
System.out.println("h= "+h+" l= "+l);
System.out.println("surface= "+surface());}

float surface(){return h*l;}
```

```
public class Carree extends Rectangle{
    int couleur;
Carree(){ super(5,5);couleur=1;}
Carree(float l, int c){ super(l,l);couleur=c;}
}
```






```
Carree c=new Carree();
c.afficher();
Carree c2=new Carree(10,3);
c2.afficher();
```

```
h= 5.0 l= 5.0
surface= 25.0
h= 10.0 l= 10.0
surface= 100.0
```

29

Bases de POO

Héritage: Constructeur et super()

Si le constructeur de la sous-classe ne contient pas un appel explicite à un constructeur de superclasse, le compilateur ajoute l'instruction **super()**, comme première instruction.

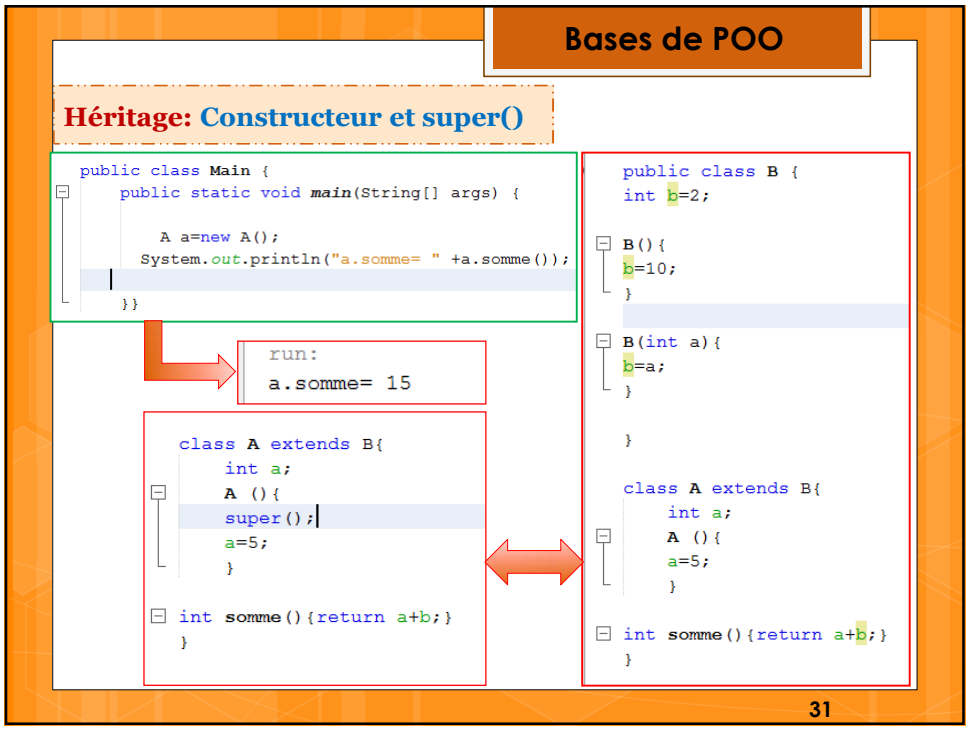
```
class A extends B{

int somme(){return 2*b;}
}
```

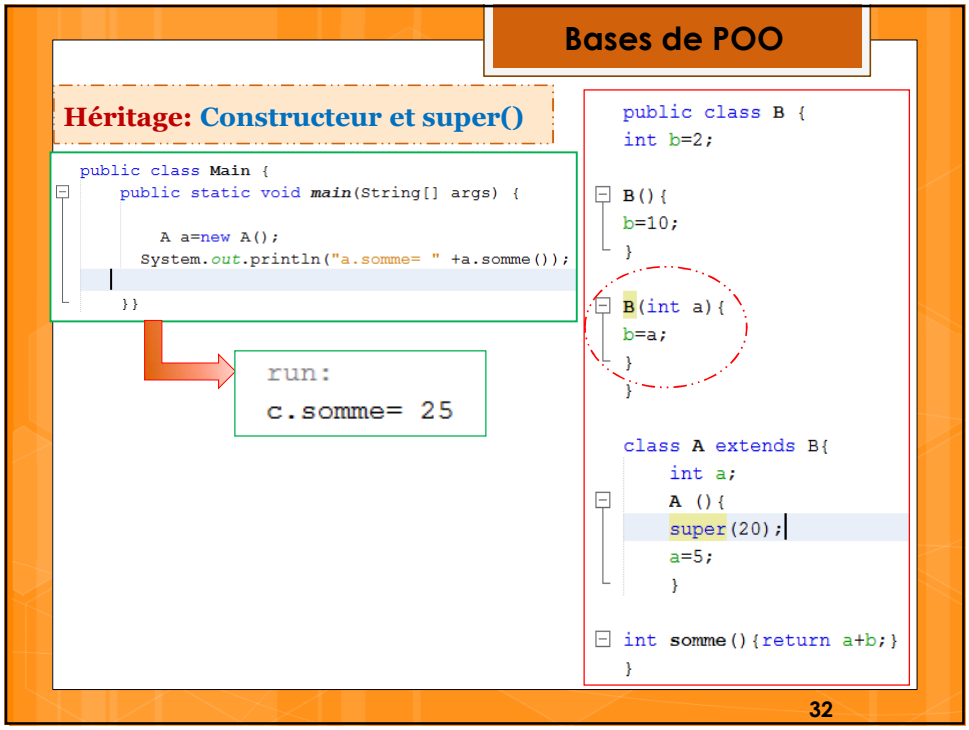
```
public class B {
int b=2;
/*
B(){
b=10;
}*/

B(int a){
b=a;
}

}
```

```
public class Main {
    public static void main(String[] args) {

        A a=new A();
       System.out.println("a.somme= " +a.somme());

    }}
```

```
run:
a.somme= 20
```

30


Bases de POO
Héritage: Constructeur et super()
public class Main {
public static void main(String[] args) {
A a=new A();
System.out.println("a.somme= " +a.somme());
}}
run:
a.somme= 15
class A extends B{
int a;
A (){
super();
a=5;
}
int somme(){return a+b;}
}
public class B {
int b=2;
B(){
b=10;
}
B(int a){
b=a;
}
}
class A extends B{
int a;
A (){
a=5;
}
int somme(){return a+b;}
}
31


Bases de POO
Héritage: Constructeur et super()
public class Main {
public static void main(String[] args) {
A a=new A();
System.out.println("a.somme= " +a.somme());
}}
run:
c.somme= 25
public class B {
int b=2;
B(){
b=10;
}
B(int a){
b=a;
}
}
class A extends B{
int a;
A (){
super(20);
a=5;
}
int somme(){return a+b;}
}
32

Bases de POO

Héritage: Constructeur et super()

```
public class B {
int b=2;

B(){
b=10;
}

/*B(int a){
b=a;
}*/
}

class A extends B{
    int a;
    A (){
    super(20);
    a=5;
    }

int somme(){return a+b;}
}
```

```
public class B {
int b=2;

B(){
b=10;
}

B(int a){
b=a;
}
}

class A extends B{
    int a;
    A (){
    super(20);
    a=5;
    }

int somme(){return a+b;}
}
```

33

Bases de POO

Héritage: Ordre des constructeurs

```
public class B {
int b=2;
B(){b=10; }
B(int a){b=a;}
}
```

```
class A extends B{
    int a;
    A (){
    a=5;
    }

int somme(){return a+b;}
}
```

```
class T extends A{
int t[];
T(){
    t=new int[b+a];
}

int tailleT(){return t.length;}
}
```

Appel au constructeur de B à partir de T?

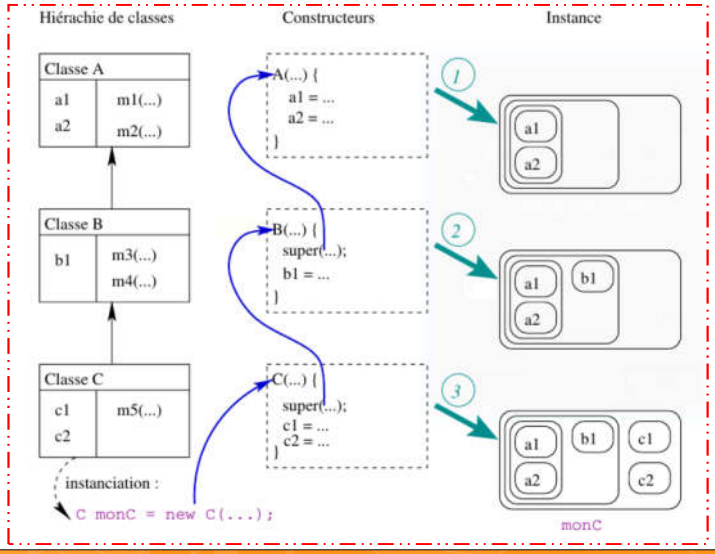
super.super();

Super().super();

34

Bases de POO

Héritage: Ordre des constructeurs



35

Bases de POO

Héritage: Ordre des constructeurs

```
public class Main {
public static void main(String[]args){

        T a=new T();
    }
}
```

```
class A extends B{
    int a;private int e;
    A (){super();
    System.out.println("Bonjour 1");
    }
}
```

```
class T extends A{
 T (){ super();
    System.out.println("Salut 1");
    }
}
```

```
public class B {

B(){
System.out.println("Hallo 1");}
B(int a){
System.out.println("Hallo 2");
}
}
```

```
class A extends B{
    int a;private int e;
    A (){
    System.out.println("Bonjour 1");
    }
}
```

```
class T extends A{
 T (){
    System.out.println("Salut 1");
    }
}
```

36

Bases de POO

Héritage: quiz

```
class TestMain {
   public static void main (String[] args) {
      Car         myCar;
      ElectricCar myElecCar;

      myCar = new Car();
      myCar.make = "Chevy";
      myCar.weight = 1000;
      myCar.color = "Red";

      myElecCar = new ElectricCar();
      myCar.make = "Chevy";
      myCar.weight = 500;
      myCar.color = "Silver";
   }
}
```

```
class Car {
   public    String    make;
   protected int       weight;
   private   String    color;

   ...
}
```

```
class ElectricCar extends Car {
   private   int   rechargeHour;

   public ElectricCar() {
      ...
   }

   //copy constructor
   public ElectricCar (ElectricCar car) {
      this.make         = car.make;
      this.weight       = car.weight;
      this.color        = new String(car.color);
      this.rechargeHour= car.rechargeHour;
   }

   ...
}
```

Lesquelles des instructions suivantes qui sont valides ?

37

ꕥ BON COURAGE ꕥ