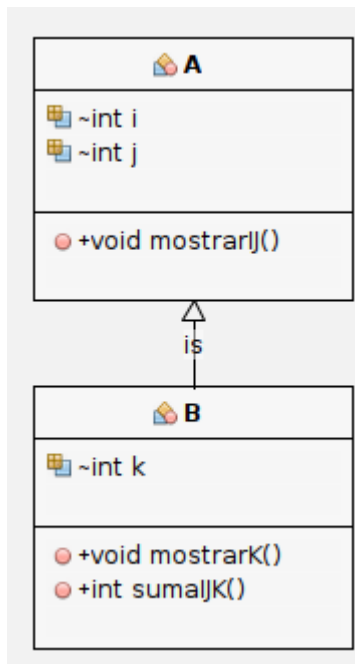


HERENCIA EN EL MISMO PAQUETE



```

public class A {
    int i;
    int j;

    public void mostrarIJ(){
        System.out.println("i="+this.i+" j="+this.j);
    }
}

public class B extends A{
    //Hereda de A las propiedades i , j
    //Añade la propiedad k
    int k;

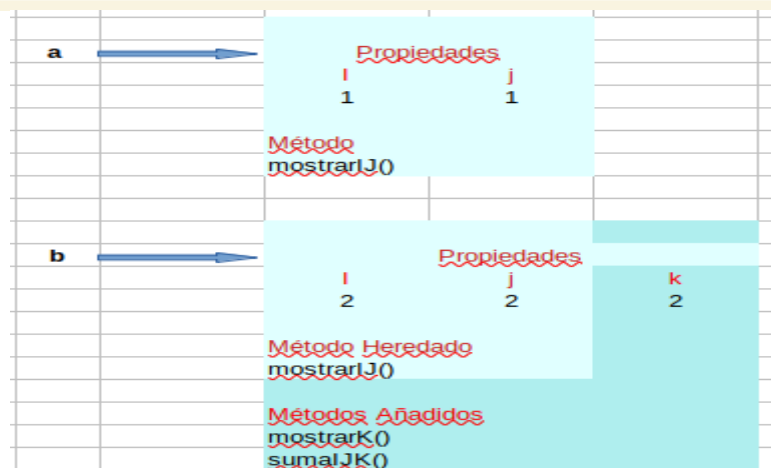
    //Hereda metodo de A: mostrarIJ()
    //Añade los metodos: mostrarK() y sumaIJK()
    public void mostrarK(){
        System.out.println("k: "+k);
    }

    public int sumaIJK(){
        return super.i+super.j+this.k;
    }
}
  
```

```

public class HerenciaAB {
    public static void main(String[] args) {
        A a=new A();
        a.i=1;
        a.j=1;
        System.out.println("Ejecuto a.mostrarIJ()");
        a.mostrarIJ();

        B b=new B();
        b.i=2;
        b.j=2;
        b.k=2;
        System.out.println("Ejecuto b.mostrarIJ()");
        b.mostrarIJ();
        System.out.println("");
        System.out.println("Ejecuto b.mostrarK()");
        b.mostrarK();
        System.out.println("Suma de los campos de b="+b.sumaIJK());
    }
}
  
```



A UNA PROPIEDAD PRIVADA EN UNA SUPERCLASE NO SE PUEDE ACCEDER DESDE UNA SUBCLASE

```
public class A {  
    private int i;  
    int j;  
  
    public void mostrarIJ(){  
        System.out.println("i="+i+" j="+j);  
    }  
}
```

```
public class B extends A{  
    int k;  
  
    public void mostrarK(){  
        System.out.println("k="+k);  
    }  
  
    public int sumaIJK(){  
        return i+j+k;  
    }  
}
```

```
public class Ppat {  
  
    public static void main(String[] args) {  
        A a=new A();  
        a.i=2;  
        a.j=1;  
        System.out.println( x: "Ejecuto a.mostrarIJ()");  
        a.mostrarIJ();  
  
        B b=new B();  
        b.i=2;  
        b.j=2;  
        b.k=2;  
        System.out.println( x: "Ejecuto b.mostrarIJ()");  
        b.mostrarIJ();  
        System.out.println( x: "");  
        System.out.println( x: "Ejecuto b.mostrarK()");  
        b.mostrarK();  
        System.out.println("Suma de los campos de b="+b.sumaIJK());  
    }  
}
```

SOLUCION:

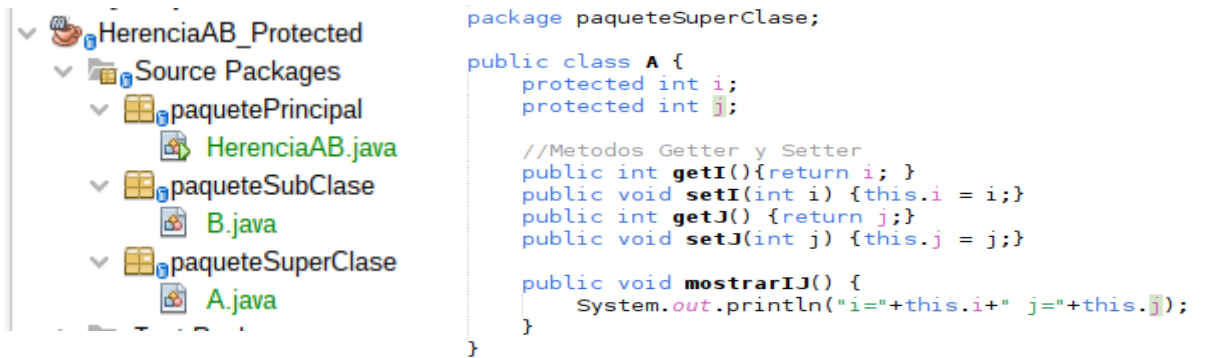
Habría que diseñar los métodos **Getter** y **Setter** para esa propiedad, y de esta forma las subclases podrán acceder a la propiedad. Pero cuando tenemos una jerarquía con muchas subclases es engorroso estar usando Getter y Setter.

La clase que se hereda se llama superclase y la clase que hereda, clase subclase. La superclase se define con las propiedades comunes y las subclases propiedades específicas.

SOLUCIÓN REAL: Uso del modificador **protected**

UNA JERARQUÍA DE CLASES DISTRIBUIDA EN DISTINTOS PAQUETES

* Uso del modificador **protected** en las propiedades como en los métodos



```
package paqueteSuperClase;

public class A {
    protected int i;
    protected int j;

    //Metodos Getter y Setter
    public int getI(){return i; }
    public void setI(int i) {this.i = i;}
    public int getJ() {return j;}
    public void setJ(int j) {this.j = j;}

    public void mostrarIJ() {
        System.out.println("i="+this.i+" j="+this.j);
    }
}
```

```
package paqueteSubClase;

import paqueteSuperClase.A;

public class B extends A{
    //Hereda de A las propiedades i , j
    //Añade la propiedad k
    protected int k;

    //Hereda metodo de A: mostrarIJ() y los Getter y Setter de i j
    //Añade los metodos: mostrarK() y sumaIJK()
    public void mostrarK(){
        System.out.println("x: k");
    }

    public int sumaIJK(){
        return super.i+super.j+this.k;
    }

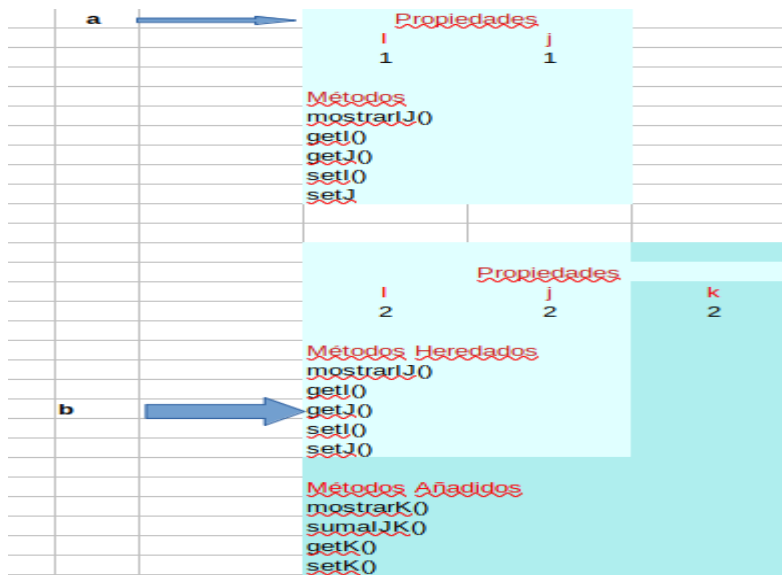
    //Añade los Getter y Setter
    public int getK() {return k;}
    public void setK(int k) {this.k = k;}
}
```

```
package paquetePrincipal;

import paqueteSubClase.B;
import paqueteSuperClase.A;

public class HerenciaAB {
    public static void main(String[] args) {
        A a=new A();
        a.setI( i:1);
        a.setJ( j:1);
        System.out.println( x: "Ejecuto a.mostrarIJ()");
        a.mostrarIJ();

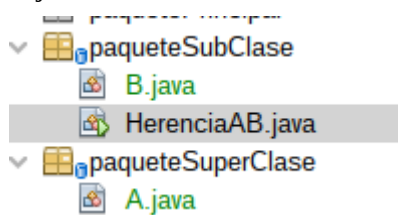
        B b=new B();
        b.setI( i:2);
        b.setJ( j:2);
        b.setK( k:2);
        System.out.println( x: "Ejecuto b.mostrarIJ()");
        b.mostrarIJ();
        System.out.println( x: "");
        System.out.println( x: "Ejecuto b.mostrarK()");
        b.mostrarK();
        System.out.println("Suma de los campos de b="+b.sumaIJK());
    }
}
```



Con **protected**,

- a) Acceso es público en la jerarquía de clases, aunque éstas estén en distintos paquetes.
- b) Si alguna clase no pertenece a la jerarquía y comparte un paquete con alguna clase sí pertenezca, todos los miembros declarados como **protected** de la subclase será accesibles (se considerarán amigables) para la clase no perteneciente a la jerarquía:

Ej,



La línea 22 NO da error

```

5  package paqueteSubClase;
6
7  import paqueteSuperClase.A;
8
9  public class HerenciaAB {
10     public static void main(String[] args) {
11         A a=new A();
12         a.setI( i:1);
13         a.setJ( j:1);
14         System.out.println( x: "Ejecuto a.mostrarIJ()");
15         a.mostrarIJ();
16
17
18         B b=new B();
19         b.setI( i:2);
20         b.setJ( j:2);
21
22         b.k=2;
23
24         System.out.println( x: "Ejecuto b.mostrarIJ()");
25         b.mostrarIJ();
26         System.out.println( x: "");
27         System.out.println( x: "Ejecuto b.mostrarK()");
28         b.mostrarK();
29         System.out.println("Suma de los campos de b="+b.sumaIJK());
30     }
31 }

```

USOS DE super

a) CONSTRUCTORES

```
package paqueteSuperClase;

public class A {
    protected int i;
    protected int j;

    public A(int i, int j) {
        this.i = i;
        this.j = j;
    }

    //Metodos Getter y Setter
    public int getI(){return i;}
    public void setI(int i) {this.i = i;}
    public int getJ() {return j;}
    public void setJ(int j) {this.j = j;}

    public void mostrarIJ() {
        System.out.println("i="+this.i+" j="+this.j);
    }
}

package paqueteSubClase;

import paqueteSuperClase.A;

public class B extends A{
    //Hereda de A las propiedades i , j
    //Añade la propiedad k
    protected int k;

    public B(int k, int i, int j) {
        super(i, j); //Llama al constructor de la clase A
        this.k = k;
    }

    //Hereda metodo de A: mostrarIJ() y los Getter y Setter de i j
    //Añade los metodos: mostrarK() y sumaIJK()
    public void mostrarK(){
        System.out.println("k: "+k);
    }

    public int sumaIJK(){
        return super.i+super.j+this.k;
    }

    //Añade los Getter y Setter
    public int getK() {return k;}
    public void setK(int k) {this.k = k;}
}

package paquetePrincipal;

import paqueteSubClase.B;
import paqueteSuperClase.A;

public class HerenciaAB {
    public static void main(String[] args) {
        A a=new A(1,1);
        System.out.println("Ejecuto a.mostrarIJ()");
        a.mostrarIJ();

        B b=new B(2,2,2);

        System.out.println("Ejecuto b.mostrarIJ()");
        b.mostrarIJ();
        System.out.println("");
        System.out.println("Ejecuto b.mostrarK()");
        b.mostrarK();
        System.out.println("Suma de los campos de b="+b.sumaIJK());
    }
}
```

Ej, Ver Caja, CajaColor y CajaPeso en Proyecto_Herencia

```

package paqueteSuperClase;

import paquetePrincipal.CajaColor;
public class Ppal2 {
    public static void main(String[] args) {
        Caja cajita=new Caja( alto: 10, ancho: 20, profundo: 30);

        System.out.println( x: "----DATOS DE cajita ----");
        System.out.println("cajita =" +cajita);
        System.out.println("Volumen de cajita= "+cajita.volumen());

        System.out.println( x: "----DATOS DE cajonVerde ----");
        CajaColor cajonVerde=new CajaColor( color: "Verde", alto: 23, ancho: 56, profundo: 78);
        System.out.println("cajonVerde =" +cajonVerde);
        System.out.println("Volumen de cajonVerde= "+cajonVerde.volumen());
        System.out.println("Color de cajonVerde =" +cajonVerde.getColor());

        CajaColor cajonRojo=new CajaColor( color: "Rojo");
        System.out.println("cajonRojo =" +cajonRojo);
    }
}

```

----DATOS DE cajita ----

cajita =[Alto=10 Ancho=20 Profundo=30]

Volumen de cajita= 6000

----DATOS DE cajonVerde ----

cajonVerde =[Alto=23 Ancho=56 Profundo=78] Color=Verde]

Volumen de cajonVerde= 100464

Color de cajonVerde =Verde

cajonRojo =[Alto=0 Ancho=0 Profundo=0] Color=Rojo]

b) **super** en MÉTODOS

```

@Override
public String toString() {
    return "[" + super.toString()+" Color=" + color + ']';
}

```

c) **super** en PROPIEDADES, CAMPOS o ATRIBUTOS

```

public class B extends A{
    protected int i;

    public B(int k, int i, int j) {
        super(i, j);
        this.i = k;
    }

    public int suma(){
        return super.i+super.j+this.i;
    }
}

```

ORDEN DE EJECUCIÓN DE LOS CONSTRUCTORES

Los constructores de las subclases llaman al constructor de la clase inmediatamente superior, que a su vez, llamará al constructor de la clase inmediatamente superior, y así sucesivamente, hasta llegar a la superclase, que comenzará a ejecutar los constructores y crear objetos.

REFERENCIA A UNA VARIABLE DE UNA SUPERCLASE A UNA CLASE DERIVADA

Volviendo al caso de las clases A y B, tenemos en la clase Principal:

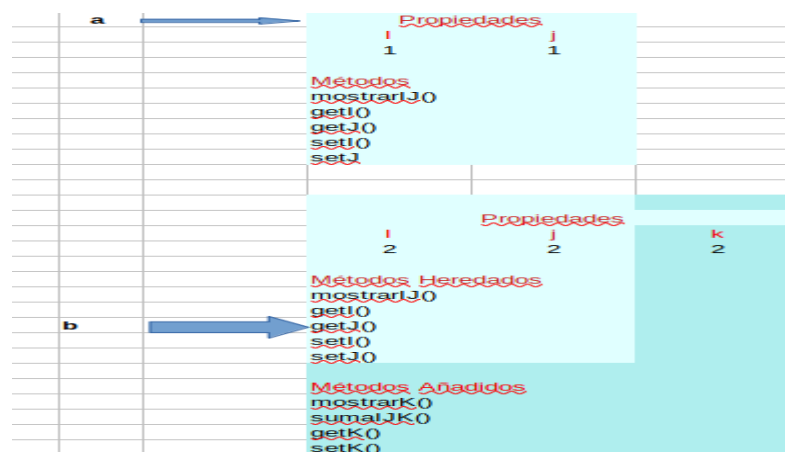
```
package paquetePrincipal;

import paqueteSubClase.B;
import paqueteSuperClase.A;

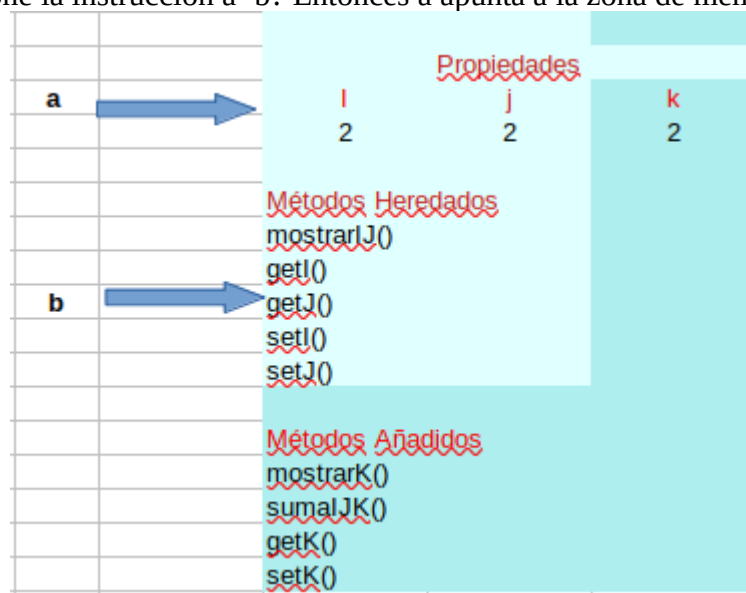
public class HerenciaAB {
    public static void main(String[] args) {
        A a=new A( i:1, j:1);
        System.out.println( x: "Ejecuto a.mostrarIJ()");
        a.mostrarIJ();

        B b=new B( k:2, i:2, j:2);

        System.out.println( x: "Ejecuto b.mostrarIJ()");
        b.mostrarIJ();
        System.out.println( x: "");
        System.out.println( x: "Ejecuto b.mostrark()");
        b.mostrark();
        System.out.println("Suma de los campos de b="+b.sumaIJK());
    }
}
```



¿Qué ocurre si se pone la instrucción a=b? Entonces a apunta a la zona de memoria de b.



El objeto a al ser tipo A tan solo puede acceder a los métodos de A en B : mostrarIJ(), getI(), getJ(), setI() y setJ().

Pero NO puede acceder a los métodos añadidos de B, por lo que, a.mostrarK() daría error, al igual que a.sumaIJK(), getK() y setK().

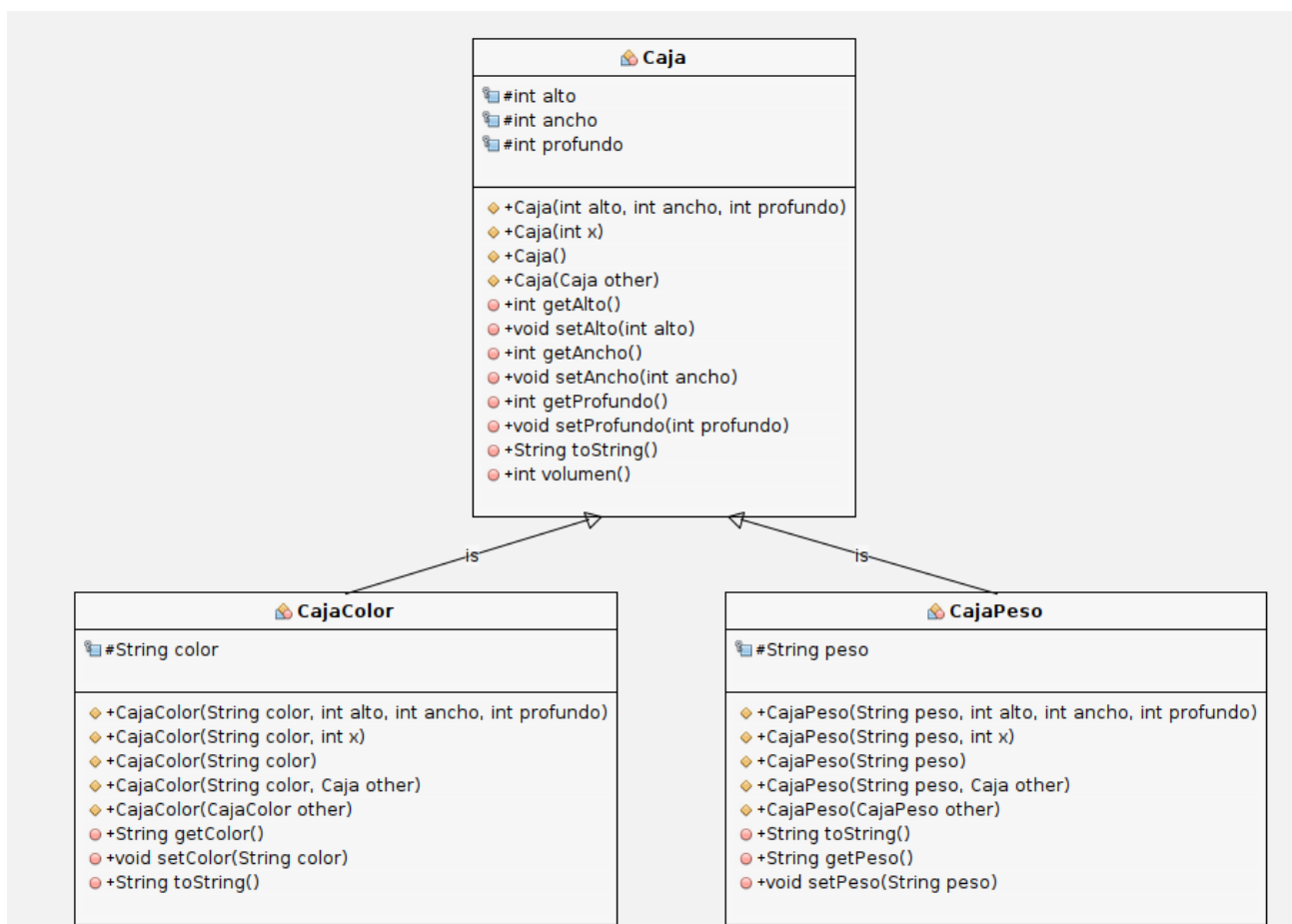
```
12 public class Ppal {
13
14     public static void main(String[] args) {
15         A a=new A(i:1, j:1);
16         B b=new B(k:2, i:2, j:2);
17
18         a=b;
19         a.mostrarIJ();
20
21         System.out.println("Suma de b="+a.sumaIJK());
22     }
23 }
```

Salida comentando la línea 21:

i=2 j=2

En la línea 19, a.mostrarIJ(), aparece que tanto I como j valen 2, porque a apunta a b, tomando sus valores.

Ahora con el ejemplo de las Cajas:




```

package paqueteSuperClase;

import paquetePrincipal.CajaColor;

public class Ppal1 {
    public static void main(String[] args) {
        Caja cajita=new Caja( alto: 10, ancho: 20, profundo: 30);

        System.out.println( x: "----DATOS DE cajita ----");
        System.out.println("Alto de la cajita= "+cajita.getAlto());
        System.out.println("Ancho de la cajita= "+cajita.getAncho());
        System.out.println("Profundo de la cajita= "+cajita.getProfundo());
        System.out.println("Volumen de cajita ="+cajita.volumen());
        System.out.println("cajita ="+cajita);

        System.out.println( x: "----DATOS DE cajonVerde ----");
        CajaColor cajonVerde=new CajaColor( color: "Verde", alto: 23, ancho: 56, profundo: 78);
        System.out.println("Alto de cajonVerde= "+cajonVerde.getAlto());
        System.out.println("Ancho de la cajonVerde= "+cajonVerde.getAncho());
        System.out.println("Profundo de la cajonVerde= "+cajonVerde.getProfundo());
        System.out.println("Volumen de cajonVerde ="+cajonVerde.volumen());
        System.out.println("cajonVerde ="+cajonVerde);

        System.out.println("Color de cajonVerde ="+cajonVerde.getColor());

        //cajita se transforma en cajonVerde
        //cajita ahora apunta a la zona de memoria de cajonVerde
        //Ahora alto, ancho y profundo de cajita seran los mismos de cajonVerde
        //Pero NO tiene color
        //cajita tan solo puede ejecutar metodos definidos en la clase Caja
        //nunca cajita podra ejecutar getColor() - definido en CajaColor

        cajita = cajonVerde;

        System.out.println( x: "----DATOS DE cajita transformada en cajonVerde----");
        System.out.println("Alto de la cajita= "+cajita.getAlto());
        System.out.println("Ancho de la cajita= "+cajita.getAncho());
        System.out.println("Profundo de la cajita= "+cajita.getProfundo());
        System.out.println("Volumen de cajita ="+cajita.volumen());

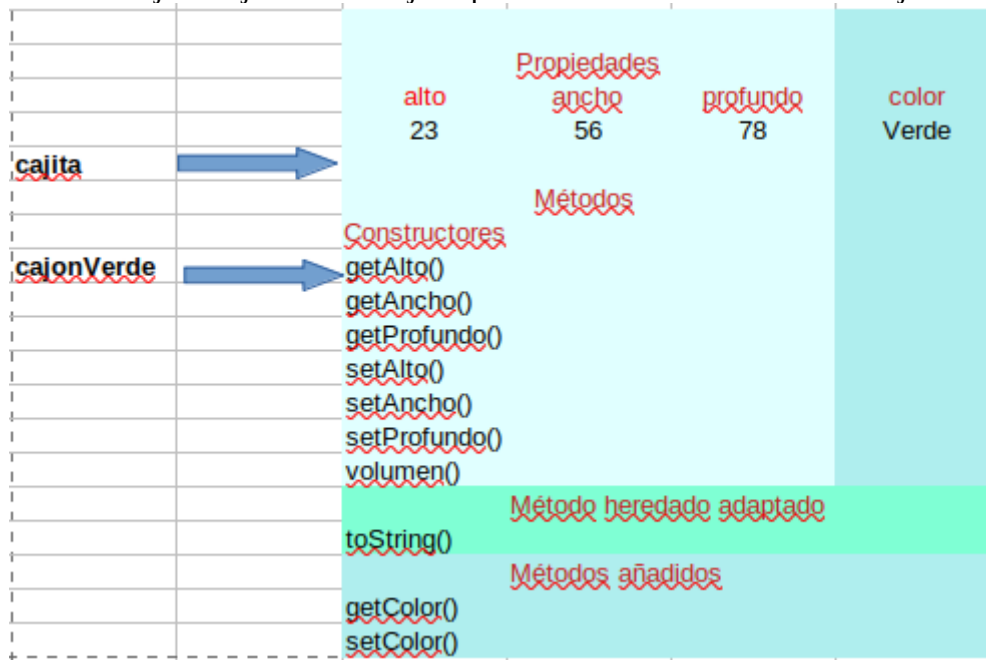
        //System.out.println("Color de cajita ="+cajita.getColor());
        //Error, porque cajita NO tiene acceso al metodo getColor() al ser de tipo Caja
        System.out.println("cajita ="+cajita);
    }
}

```

	cajita		cajonVerde	
		Propiedades alto 10 ancho 20 profundo 30		Propiedades alto 23 ancho 56 profundo 78 color Verde
		Métodos Constructores getAlto() getAncho() getProfundo() setAlto() setAncho() setProfundo() volumen() toString()		Métodos Constructores getAlto() getAncho() getProfundo() setAlto() setAncho() setProfundo() volumen() toString()
				Método heredado adaptado
				Métodos añadidos getColor() setColor()

Un método que es heredado y su código se adapta al interés de la clase se denomina: **sobreescrito**.
El método **toString()** en **cajaColor** está sobreescrito (**@override**).

¿Qué ocurre cuando `cajita=cajonVerde`? `cajita` apunta a la zona de memoria de `cajonVerde`.



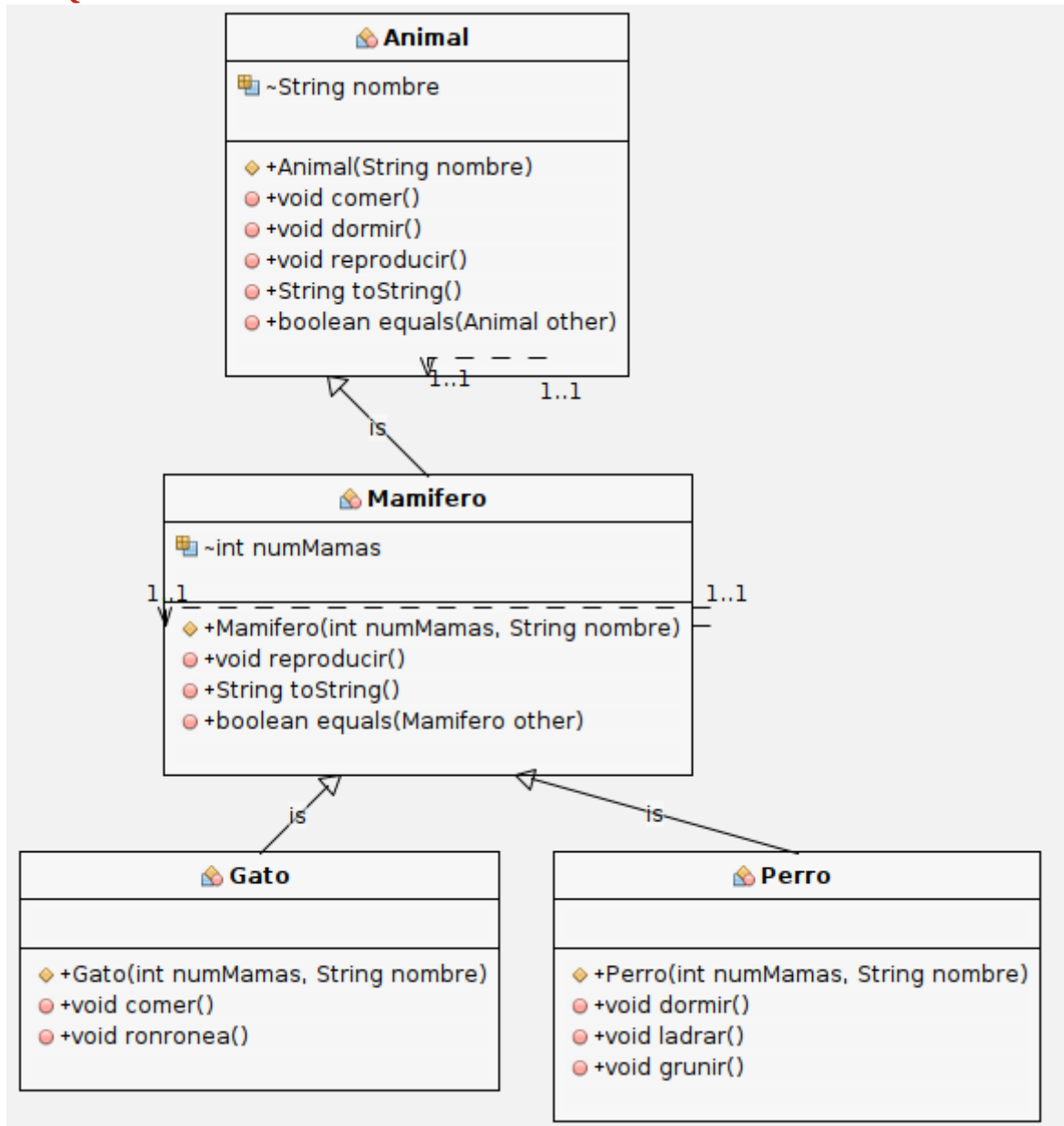
cajita NO puede acceder a los métodos `getColor()` y `setColor()`, pero sí al resto.

Y cuando `cajita` ejecute el método `toString()` ejecutará el de `cajonVerde`, mostrando los valores de `alto`, `ancho`, `profundo` y `color` de **cajonVerde** (ejecuta el código de `toString()` de `CajaColor`).

A este proceso se le llama: Polimorfismo, `cajita` puede tener varias formas, `Caja`, `CajaColor` y `CajaPeso`.

Prueba a crear un objeto de `CajaPeso`: `cajon23`, haz que `cajita` apunte a `cajon23` y comprobarás que al ejecutar el método sobrescrito `toString()` se ejecuta el adaptado en la clase `CajaPeso` para `cajon23`.

JERARQUÍA MULTINIVEL



```

public class Animal{
    String nombre;

    //Constructor
    public Animal(String nombre){this.nombre=nombre;}
    //metodos propios
    public void comer(){ System.out.println( x: "Animal -Necesita comer para sobrevivir");}
    public void dormir(){System.out.println( x: "Animal -Es indispensable para descansar");}
    public void reproducir(){System.out.println( x: "Animal -Origina nuevos seres vivos");}
    //metodo sobreescrito
    @Override
    public String toString(){ return "[Nombre =" +this.nombre+""]";}
}

public class Mamifero extends Animal{
    int numMamas;

    //Constructor
    public Mamifero(int numMamas, String nombre){ super(nombre); this.numMamas=numMamas;}
    //metodos sobreescritos
    @Override
    public void reproducir(){ System.out.println( x: "Mamifero -Reproduccion vivipara");}
    @Override
    public String toString(){ return "[" +super.toString()+" Numero de mamas: "+this.numMamas+""]";}
}

public class Perro extends Mamifero{
    //Constructor
    public Perro(int numMamas,String nombre){super(numMamas, nombre);}
    //metodo sobreescrito
    @Override
    public void dormir(){System.out.println( x: "Perro -Duerme en funcion del ejercicio que realiza");}

    //metodos a agregar
    public void ladrar(){System.out.println( x: "Perro -Ladra que es labor social de guardia");}
    public void grunir(){System.out.println( x: "Perro -Gruñe Grrrr...");}
}

public class Gato extends Mamifero{
    //Constructor
    public Gato(int numMamas, String nombre){ super(numMamas, nombre);}
    //Metodos sobreEscritos
    @Override
    public void comer(){System.out.println( x: "Gato - Su comida preferida: pescado");}

    //Metos propios
    public void ronronea(){System.out.println( x: "Gato - Esta feliz");}
}

```

```

public static void main(String args[]){
    //a.
    System.out.println( x: "---- FUNCIONES DE gamu como Animal-----");
    Animal gamu=new Animal( nombre: "Gamusino");
    System.out.println("Animal: "+gamu);
    gamu.comer();
    gamu.dormir();
    gamu.reproducir();

    //b.
    System.out.println( x: "---- FUNCIONES DE laika como Mamifero-----");
    Mamifero laika=new Mamifero( numMamas: 9, nombre: "Mamifero espacial ruso");
    System.out.println("Mamifero: "+laika); //Heredado de Animal- adaptado en Mamifero
    laika.comer(); //heredado de Animal , sin adaptar
    laika.dormir(); //heredado de Animal, sin adaptar
    laika.reproducir(); //heredado de Animal y adaptado en Mamifero

    //c.
    System.out.println( x: "---- FUNCIONES DE pluto como perro-----");
    Perro pluto=new Perro( numMamas: 10, nombre: "Pluto, amigo de Mickey Mouse");
    System.out.println("Perro: "+pluto); //Heredado de Mamifero
    pluto.comer(); //heredado de Animal , sin adaptar
    pluto.dormir(); //heredado de Animal, adaptado en Perro
    pluto.reproducir(); //heredado de Mamifero

    pluto.ladRAR();
    pluto.grunir();

    //d.
    System.out.println( x: "---- FUNCIONES DE doraemon como gato-----");
    Gato doraemon=new Gato( numMamas: 8, nombre: "Doraemon, el gato cosmico");
    System.out.println("Gato: "+doraemon); //Heredado de Mamifero
    doraemon.comer(); //Heredado de Animal se adapta Gato
    doraemon.dormir(); //Heredado de Animal
    doraemon.reproducir(); //Heredado de Animal y adaptado en Mamifero

    doraemon.ronronear();
}

```

//a.

Propiedades		
	nombre	
	Gamusino	
Métodos		
gamu	Constructores	
	getNombre()	Animal
	setNombre()	Animal
	comer()	Animal
	dormir()	Animal
	reproducir()	Animal
	toString()	Animal

//b.

Propiedades		
	numMamas	nombre
	9	Mamifero espacial ruso
Métodos		
laika	Constructores	
	getNombre()	Animal
	setNombre()	Animal
	dormir()	Animal
	comer()	Animal
	reproducir()	Método heredado adaptado Mamifero
	toString()	Mamifero
Métodos añadidos		
	getNumMamas()	Mamifero
	setNumMamas()	Mamifero

//c.

Propiedades		
	numMamas	nombre
	10	Pluto, amigo de Mickey Mouse
Métodos		
pluto	Constructores	
	getNombre()	Animal
	setNombre()	Animal
	comer()	Animal
	getNumMamas()	Mamifero
	setNumMamas()	Mamifero
	reproducir()	Mamifero
	toString()	Mamifero
Método heredado adaptado		
	dormir()	Perro
Métodos añadidos		
	ladRAR()	Perro
	grunir()	Perro

//d.

Propiedades		
	numMamas	nombre
	8	Doraemon, el gato cósmico
Métodos		
doraemon	Constructores	
	getNombre()	Animal
	setNombre()	Animal
	dormir()	Animal
	getNumMamas()	Mamifero
	setNumMamas()	Mamifero
	reproducir()	Mamifero
	toString()	Mamifero
Método heredado adaptado		
	comer()	Gato
Métodos añadidos		
	ronronear()	Gato

DOWNCASTING (POLIMORFISMO)

```
package paqueteHerenciaAnimal_MultiNivel;
public class HerenciaAnimal_Video2 {
    public static void main(String[] args) {
        System.out.println( x: "---- FUNCIONES DE pluto como perro----");
        Perro pluto=new Perro( numMamas: 10, nombre: "Pluto, amigo de Mickey Mouse");
        System.out.println("Perro: "+pluto); //Heredado de Mamifero
        pluto.comer(); //heredado de Animal , sin adaptar
        pluto.dormir(); //heredado de Animal, adaptado en Perro
        pluto.reproducir(); //heredado de Mamifero

        pluto.ladrrar();
        pluto.grunir();

        System.out.println( x: "----- POLIMORFISMO -----");
        Mamifero laika=new Mamifero( numMamas: 9, nombre: "Mamifero espacial ruso");
        System.out.println("Laika: "+laika);
        laika.comer(); //Heredado de Animal
        laika.dormir(); //Heredado de Animal
        laika.reproducir(); //Heredado de Animal se adapta a Mamifero

        //laika apunta pluto, por lo que el mamifero laika se transforma a perro pluto
        //DownCasting
        laika=pluto;
        System.out.println("laika como pluto (perro): "+laika);
        laika.comer(); //Heredado de Animal
        laika.dormir(); //Heredado de Animal, adaptado a un Perro
        laika.reproducir(); //Heredado de Mamifero

        //laika.ladrrar();

        //Cuando el casting no funciona, se genera una excepcion:
        // ClassCastingException
        laika=new Gato( numMamas: 8, nombre: "Silvestre, lindo gatito");
        System.out.println("Laika como Gato: "+laika);
        laika.comer(); //Heredado Animal y adaptado a Gato
        laika.dormir();
        laika.reproducir();
    }
}
```



```
//laika apunta pluto, por lo que el mamifero laika se transforma a perro pluto
//DownCasting
laika=pluto;
System.out.println("laika como pluto (perro): "+laika);
laika.comer();//Heredado de Animal
laika.dormir();//Heredado de Animal, adaptado a un Perro
laika.reproducir();//Heredado de Mamifero

//laika.ladrrar();
```

		<u>Propiedades</u>	
		<u>numMamas</u>	<u>nombre</u>
		10	Pluto, amigo de Mickey Mouse
<u>pluto</u>	→		
		<u>Métodos</u>	
<u>laika</u>	→	<u>Constructores</u>	
		<u>getNombre()</u>	Animal
		<u>setNombre()</u>	Animal
		<u>getNumMamas()</u>	Mamifero
		<u>setNumMamas()</u>	Mamifero
		<u>comer()</u>	Animal
		<u>reproducir()</u>	Mamifero
		<u>toString()</u>	Mamifero
		<u>Método heredado adaptado</u>	
		<u>dormir()</u>	Perro
		<u>Métodos añadidos</u>	
		<u>ladrrar()</u>	Perro
		<u>grunir()</u>	Perro

```
//Cuando el casting no funciona, se genera una excepcion:
// ClassCastingException
laika=new Gato( numMamas: 8, nombre: "Silvestre, lindo gatito");
System.out.println("Laika como Gato: "+laika);
laika.comer();//Heredado Animal y adaptado a Gato
laika.dormir();
laika.reproducir();
```

		<u>Propiedades</u>	
		<u>numMamas</u>	<u>nombre</u>
		8	Silvestre, lindo gatito
		<u>Métodos</u>	
		<u>Constructores</u>	
<u>laika</u>	→	<u>getNombre()</u>	Animal
		<u>setNombre()</u>	Animal
		<u>dormir()</u>	Animal
		<u>getNumMamas()</u>	Mamifero
		<u>setNumMamas()</u>	Mamifero
		<u>reproducir()</u>	Mamifero
		<u>toString()</u>	Mamifero
		<u>Método heredado adaptado</u>	
		<u>comer()</u>	Gato
		<u>Métodos añadidos</u>	
		<u>ronronear()</u>	Gato

POLIMORFISMO: DOWNCASTING EXPLÍCITO - instanceof

```
public class HerenciaAnimal_Video3 {
    public static void main(String[] args) {
        System.out.println( x: "---- FUNCIONES DE pluto como perro-----");
        Perro pluto=new Perro( rusaMamifero: 10, nombre: "Pluto, amigo de Mickey Mouse");
        System.out.println("Perro: "+pluto); //Heredado de Mamifero

        System.out.println( x: "---- FUNCIONES DE doraemon como gato-----");
        Gato doraemon=new Gato( rusaMamifero: 8, nombre: "Doraemon, el gato cósmico");
        System.out.println("Gato: "+doraemon); //Heredado de Mamifero

        System.out.println( x: "----- POLIMORFISMO -----");
        Mamifero laika=new Mamifero( rusaMamifero: 9, nombre: "Mamifero espacial ruso");
        System.out.println("Laika: "+laika);
        laika.comer(); //Heredado de Animal
        laika.dormir(); //Heredado de Animal
        laika.reproducir(); //Heredado de Animal se adapta a Mamifero

        //laika apunta pluto, por lo que el mamifero laika se transforma a perro pluto
        //DownCasting
        laika=pluto;
        System.out.println("Laika como pluto (perro): "+laika);
        laika.comer(); //Heredado de Animal
        laika.dormir(); //Heredado de Animal, adaptado a un Perro
        laika.reproducir(); //Heredado de Mamifero

        //laika.ladrrar();

        laika=new Gato( rusaMamifero: 8, nombre: "Silvestre, lindo gatito");
        System.out.println("Laika como Gato: "+laika);
        laika.comer(); //Heredado Animal y adaptado a Gato
        laika.dormir();
        laika.reproducir();

        //El DownCasting es directo, se hace de forma implicita
        //Cuando se quiere ejecutar un metodo AÑADIDO de una subclase por
        // un objeto de una superclase -> Downcasting debe ser explicito

        System.out.println( x: "Gremlin ronronea-----");
        Gato gremlin=(Gato)laika; //int x=(int) 12.8;
        gremlin.ronronea(); // x++;

        System.out.println( x: "Transformacion explicita (cast) de laika a Gato");
        //Es lo mismo que decir:
        ((Gato)laika).ronronea(); // ((int)x)++;

        //Cuando el casting no funciona, se genera una excepcion:
        // ClassCastException

        Mamifero et= new Gato( rusaMamifero: 4, nombre: "E.T. el pequeño ExtraTerrestre");
        Gato gremlin1=(Gato)et; //similar a int x=(int) 12.8;

        System.out.println("Gremlin =" +gremlin1);
        gremlin1.ronronea(); // x++;

        //Tambien
        System.out.println( x: "Transformacion explicita (cast) de et a Gato");
        //Es lo mismo que decir:
        ((Gato)et).ronronea(); // ((int)x)++;

        //instanceof --nos dice si el objeto es de una clase
        if (doraemon instanceof Gato){
            System.out.println( x: "Doraemon es un Gato");
            doraemon.ronronea();
        }
    }
}
```



```

//El DownCasting es directo, se hace de forma implicita
//Cuando se quiere ejecutar un metodo AÑADIDO de una subclase por
// un objeto de una superclase -> Downcasting debe ser explicito

System.out.println(x: "Gremlin ronronea-----");
Gato gremlin=(Gato)laika; //int x=(int) 12.8;
gremlin.ronronea();      // x++;

System.out.println(x: "Transformacion explicita (cast) de laika a Gato");
//Es lo mismo que decir:
((Gato)laika).ronronea(); // ((int)x)++;

//Cuando el casting no funciona, se genera una excepcion:
// ClassCastException

Mamifero et= new Gato( raza: "E.T. el pequeño ExtraTerrestre");
Gato gremlin=(Gato)et; //similar a int x=(int) 12.8;

System.out.println("Gremlin =" +gremlin1);
gremlin1.ronronea(); // x++;

//Tambien
System.out.println(x: "Transformacion explicita (cast) de et a Gato");
//Es lo mismo que decir:
((Gato)et).ronronea(); // ((int)x)++;

```

instanceof

```

//instanceof --nos dice si el objeto es de una clase
if (doraemon instanceof Gato){
    System.out.println(x: "Doraemon es un Gato");
    doraemon.ronronea();
}

```