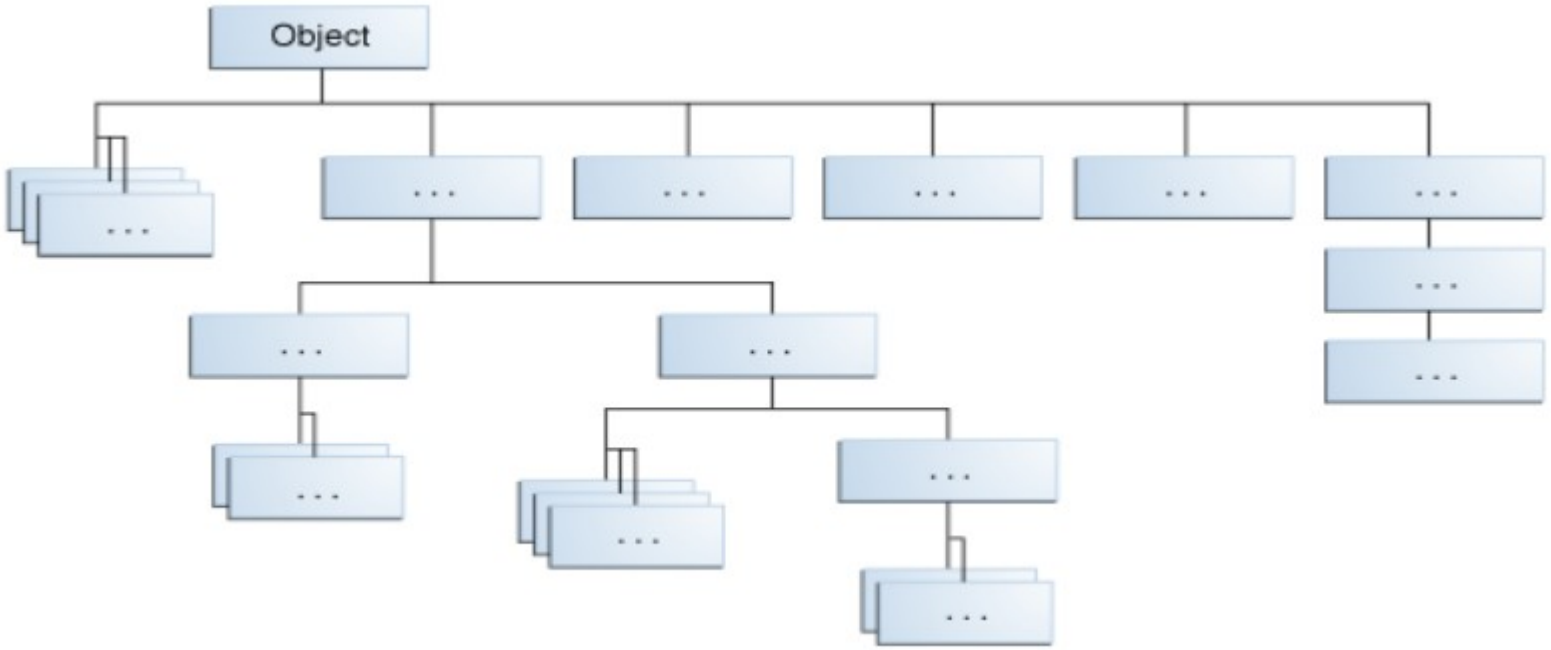
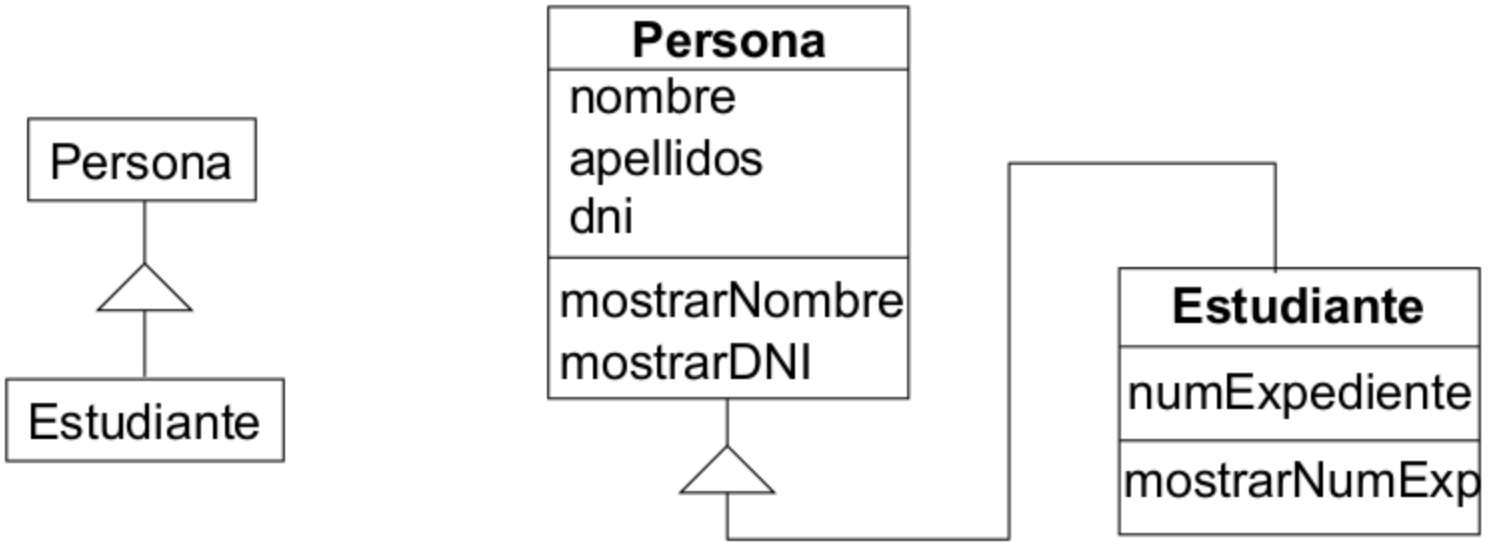
Herència i polimorfisme

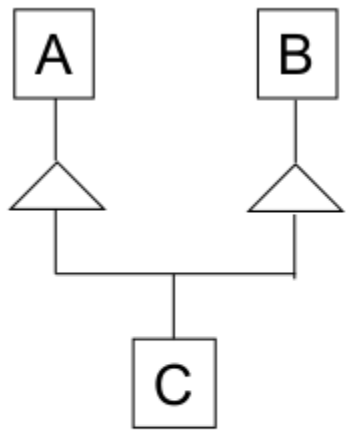


- L'herència permet definir classes (subclasses) a partir d'una altra classe més genèrica (superclasse).
- La subclasse reuneix totes la propietats de la superclasse, a més de les seves pròpies.
  L'herència potencia la reutilització de codi, genera codi més fiable i robust i redueix el cost de manteniment.

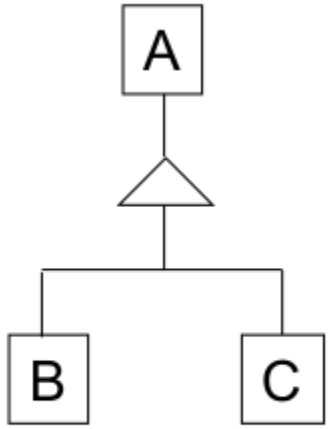



```
class Estudiante extends Persona { . . .
```

- En Java no es permet l'herència múltiple.



Error



Correcto

Una subclasse hereta tots els mètodes i atributs de la superclasse
EXCEPTE:
– Atributs i mètodes privats

– Constructors (no s'hereten però sí s'executen)

- Quan es crea un objecte d'una subclasse, el constructor de la classe pare TAMBÉ s'executa

```
class A(){
   A() { System.out.println("En A"); }
}
class B extends A {
   B() { System.out.println("En B"); }
}
class Demo {
   public static void main(String args[]) {
      B b = new B();
   }
}
```

A la pantalla sortirà :
"En A"
"En B"
Primer s'executa el constructor de la superclasse i després el

de la subclasse
Per defecte el constructor que s'executarà en la superclasse serà el constructor sense paràmetres

```
class A(){
   int i;
   A() { i = 0; }
   A( int i ){ this.i = i; }
}
class B extends A {
   int j;
   B() { j = 0; }
   B( int j ){ this.j = j; }
}
class Demo {
   public static void main(String args[]) {
      B b1 = new B(); System.out.println("i=" + b1.i + "j=" + b1.j);
      B b2 = new B(5);System.out.println("i=" + b2.i + "j=" + b2.j);
   }
}
```

A la pantalla sortirà :
i=0 j=0

i=0 j=5

Per exectuar l'execució d'un determinat constructor de la classe pare (superclasse) cal utilizar **super**

```
class B extends A {
   int j;
   B() { j = 0; }
   B( int j ){
      super(j); //
                //
      this.j = j;
   }
}
```

A la pantalla sortirà :

i=0 j=0

i=5 j=5

Si utilitzem **super**, aquesta ha de ser la primera instrucció del constructor. D'aquesta manera es respecta l'ordre

d'execució dels constructors.
Un altre exemple d'ús de **super** :

```
class Esfera {
  Esfera ( double r ) {
    radio = r;
  }
}
class Planeta extends Esfera {
  int numSatelites;
  Planeta( double r, int ns ) {
    super(r);
    numSatelites = ns;
  }
}
```

Modificadors d'accés

	private	protected	public
Mateixa classe	SI	SI	SI
Altre classe mateix paquet	NO	SI	SI
Subclasse de diferent paquet	NO	SI	SI
No subclasse de diferent paquet	NO	NO	SI

La classe **Object**

- **Object** és la classe base (superclasse) de totes les altres classes.
- Si una classe no especifica **extends**, llavors s'entén que deriva de **Object** . Per tant totes les classes deriven directa o indirectament de **Object**.
- Alguns mètodes de la classe **Object**:

  – boolean equals (Object o)
  – String toString()
  – int hashCode()

Herència Vs Composició

- No s'ha de confondre l'herència amb la composició.
- Composició: mecanisme pel qual es defineix una nova classe afegint components d'altres classes.

```
class Punto {
   int x, y;
   . . .
}
class Figura {
   Punto origen;
   . . .
```

```
class Punto {
   int x, y;
   . . .
}
class Figura extends Punto{
. . .
```

Donades dues classes A i B:

– A és un B? → Herència

– A té un B? → Composició

```
class A extends B {
```

```
class A {
   B b;
```

- Una figura no és un punt. Una Figura té un punt d'origen.

Classes abstractes

- Una classe abstracta inclou mètodes no implementats (sense codi) obliganta les subclasses directes que ho implementin
- Una classe abstracta no pot ser instanciada (new).
- Si una subclasse que deriva d'una classe abstracta no implementa alguns dels mètodes abstractes declarats en la superclasse, llavors ha de ser declarada també com a abstracta.
- Una classe abstracta pot tenir mètodes no abstractes.

```
abstract class Figura {
      . . .
   abstract double area();
}
class Rectangulo extends Figura {
      . . .
   double area() { return alto * ancho; }
}
class Circulo extends Figura {
      . . .
   double area() { return Mat.PI * radio * radio; }
}
```

El mètode area és abstracte. S'inclou la capçalera del mètode (tipus, nom i paràmetres) però no la implementació (el codi).

Com la classe Figura té un mètode abstracte, també ha de ser abstracta.

Les subclasses de Figura hauran d'implementar el mètode area.

Conversió

Podem definir :

```
Figura f;
f = new Circulo(...);
```

Però no podem fer : `f.setRadio(5);`

Ja que **f** podria ser un rectangle,quadrat, etc ..

Podem solucionar el problema fent un canvi del tipus de la referència a un subtipus (a un tipus que jeràrquicament està a un nivell inferior)

```
Figura f = new Circulo(...);
   . . .
Circulo c;
c = (Circulo)f;
c.setRadio(5);
```

Un cercle és una figura. Un cercle no té una figura.

```
if ( f instanceof Circulo )
   ((Circulo)f).setRadio(5);
else if ( f instanceof Rectangulo )
   ((Rectangulo)f).setDim(5,5);
```