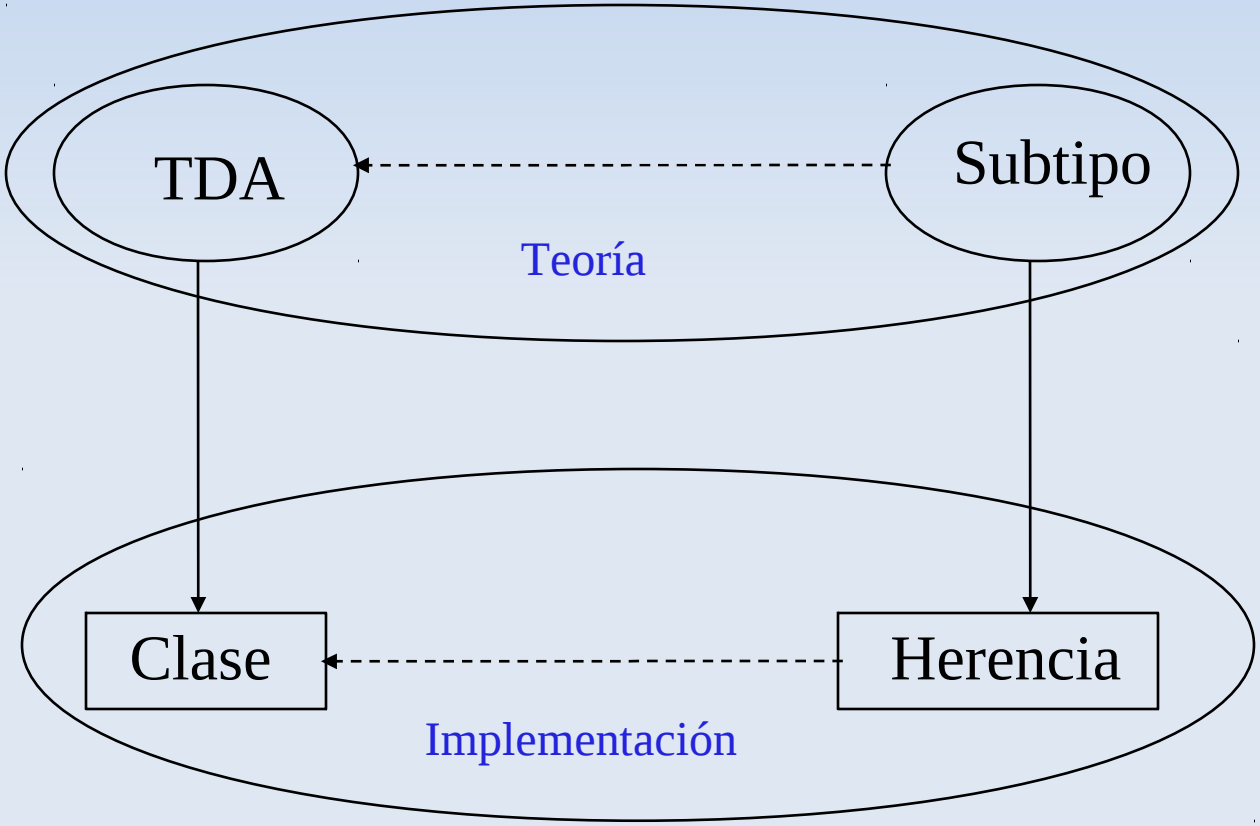
Herencia entre clases



Herencia entre clases

Componentes a heredar dentro de una clase

- Interfaz de la superclase
- Código que implementa los métodos
- Representación (variables de instancia de la superclase)

Con respecto a variables de instancia

- No hay redefinición (c++, java, etc)
- Redefinición arbitraria
- Redefinición restringida
- Definiciones ocultas (private)

Redefinición de variables de instancia

```
Class C{
    a1:T1;
    a2: T2;
    a3: T3;
}
```

```
Class C´subclass C{
        a2: T2';
        a4: T4;
        a5: T5;
}
```

Si T_2' es subclase de T_2 $\Rightarrow$ redefinición restringida

Si T_2' no es subclase de T_2 $\Rightarrow$ redefinición arbitraria

Redefinición de métodos

- No hay redefinición
- Redefinición arbitraria: sólo exige conservación del nombre del método
- Redefinición restringida: relación de subtipos entre firmas de los métodos

¿Cuando un método es subtipo de otro?

- Notación:

nombre: tipos de argumentos $\rightarrow$ tipo del resultado

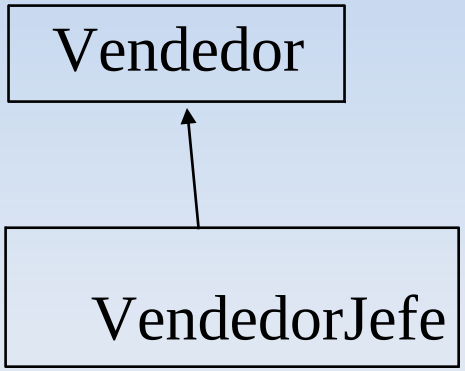
Redefinición de métodos (2)

- ¿Cuando T1 → T2 es subtipo de T3 → T4 ?
- Existen dos enfoques:
  - Regla de la covarianza:
    - $T1 \leq T3$ y $T2 \leq T4$ y
  - Regla de la contravarianza:
    - $T3 \leq T1$ y $T2 \leq T4$

Redefinición de métodos (3)

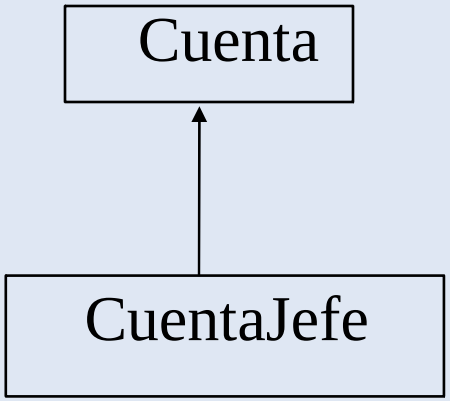
En java:

```
class Vendedor{
   ....
  public  Vendedor agregarCuenta(Cuenta c){ ...}
 ....
}
class VendedorJefe extends Vendedor{
 ...
  public  VendedorJefe agregarCuenta(CuentaJefe cj){...}
...
}
```



Uso: Cuenta cuenta;

```
Vendedor  vendedor_1, vendedor_2;
 ...
vendedor_1 = new VendedorJefe();
cuenta         =  new Cuenta();
vendedor_2  = vendedor_1. agregarCuenta(cuenta);
```



Redefinición de métodos

- Regla de la contravarianza: (Razón intuitiva)
- ¿Cuándo una función puede ser usada en el contexto de otra?
  - Los argumentos de una función representan el dominio de una función
  - El valor retornado representa el rango de la función
  - Una función con dominio D_1 es también función con dominio D_2 si y solo si $D_2 \leq D_1$
  - Una función cuyo rango es R_1 es también función con rango R_2 si y solo si $R_1 \leq R_2$
- Conclusión:
  - D_1 más grande que D_2
  - R_1 más pequeño que R_2

Contratos y Herencia

- **¿Cómo cambia el contrato en las clases heredadas?**

```
class  C
    ... m(...){
       precondición P1
       ......
       postcondición P2
    }
```

```
class  C' extends C
    ... m(...){
       precondición P'1
       ......
       postcondición P'2
    }
```

  - *P'*$_1$ más débil que *P*$_1$
  - *P'*$_2$ más fuerte que *P*$_2$
  - Invariante de *C'* más fuerte que invariante de *C*

Contratos y Herencia (2)

```
class Cuenta{
    protected float _balance;
    protected float _old_balance;
    protected void hold(){_old_balance=_balance;}
    protected boolean invariante(){ return balance() >= 0;}
    public Cuenta(){balance = 0; assert invariante() : “inv. invalido”;}
    public float balance(){return _balance;}
    public  void depositar(float x){
            assert  x>0:  “ x menor o igual a 0”;
            hold(); _balance +=x;
            assert _balance == old_balance+x : “balance mal calculado”;
            assert   invariante() : “invariante inválido”;}
    void girar(float x){
            assert x<=balance() && x > 0: “ x inválido”;
            hold(); _balance -=x;
            assert _balance == old_balance -x : “balance mal calculado”;
            assert  invariante(): “invariante inválido”; }
}
```

Contratos y Herencia (3)

```
class CuentaCredito extends Cuenta{
    protected float _max_credito, _credito;
    protected float _old_credito;
    protected  void hold(){_old_credito=_credito; super.hold();}
    protected  boolean invariante(){return _credito <=_max_credito &&
                                                    super.invariante();}
    public   CuentaCredito(int x){_max_credito = x; _credito= 0 ;
                                assert  invariante() : “inv. inválido de CuentaCredito”;
                                assert _max_credito > 0 : “ max_credito <=0”;
                                assert  _credito ==0 : “credito distinto de 0”;
    public float balance(){return _balance+_credito;}
    public void aumentar(float x){
            assert  x>0 && _credito+x <= _max_credito : “x inválido”;
            hold(); _credito +=x;
            assert  _credito == old_credito+x && _balance == _old_balance : “...”;
            assert  invariante() : “invariante inválido”;
    public void reducir(float x){
            assert x> 0 && x<=_credito  : “x inválido”;
            hold(); _credito -=x;
            assert  _credito == old_credito -x: “credito inválido”;
            assert    invariante() :  “invariante inválido de Cuenta Credito”; } }
```

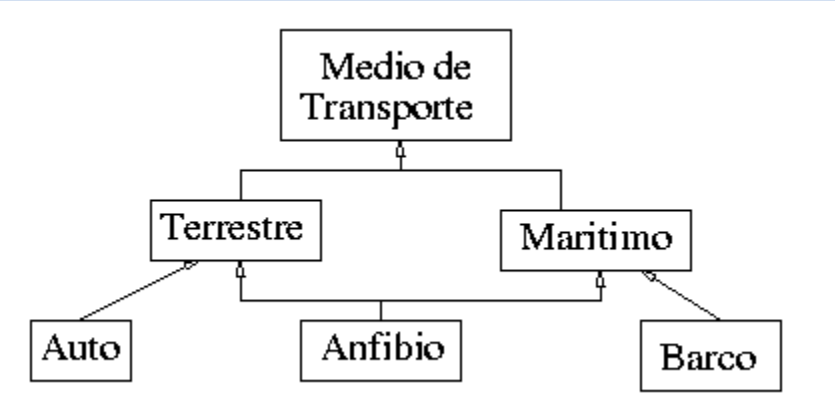
Exclusión de métodos de la superclase

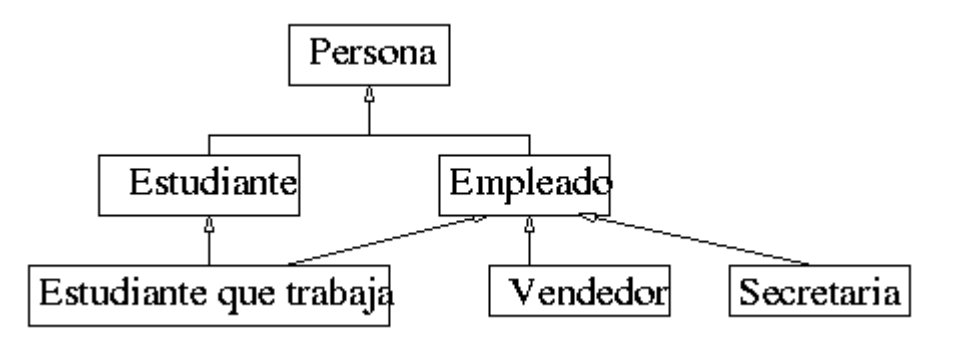
- Existen dos formas básicas de excluir código:
  - Redefinir el método y generar mensaje de error cuando se llame
  - Especificar que el método no debe ser heredado. Ejemplo: exclude

- No usarlo: Contradice el concepto de subtipos

Herencia múltiple

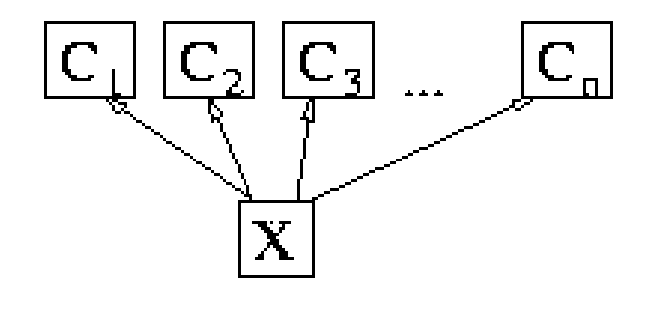
Motivación: objetos pueden heredar propiedades de más de una clase





Herencia múltiple (2)

Forma general :



Propiedades(x) = U1..n Propiedades(Ci) + Propiedades Locales(x)

- ¿Qué conflictos pueden existir?
  - Propiedad de igual nombre viene de un ancestro común
  - Propiedad de igual nombre viene de clases distintas.

Herencia múltiple (3)

- ¿Cómo resolver el conflicto de igual nombre y distintos ancestros?
  - Ejemplo: atributo dirección en Estudiante y Empleado
- Linearización
  - class EstudianteQueTrabaja:public Empleado, Estudiante..
  - Orden: Persona, Estudiante, Empleado, EstudianteQueTrabaja (Flavors)
- Cambio de nombre
  - Estudiante rename dirección as dirección_tmp (Eiffel)
- Especificación de atributos con nombre de clase
  - Estudiante::dirección o Empleado::dirección (C++)