Programación III

I.T.Informática de Sistemas

La estructura dinámica: Objetos.

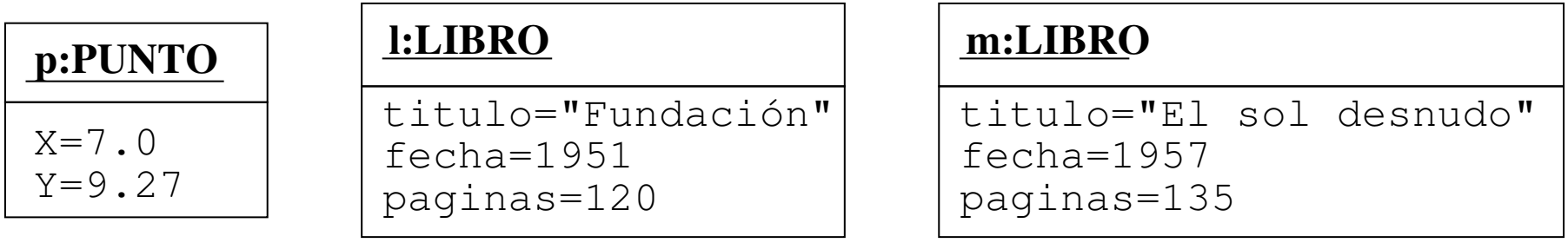
Prof. Félix Prieto Arambillet
Departamento de Informática
Universidad de Valladolid
Curso 2003/2004

El concepto de objeto

Definición 1 *Llamaremos objeto a la instancia de una clase en tiempo de ejecución.*

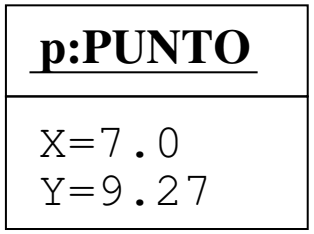
Si un objeto o *es instancia de una clase* C *diremos que* C *es la clase generatriz de* o*, o simplemente su generatriz.*

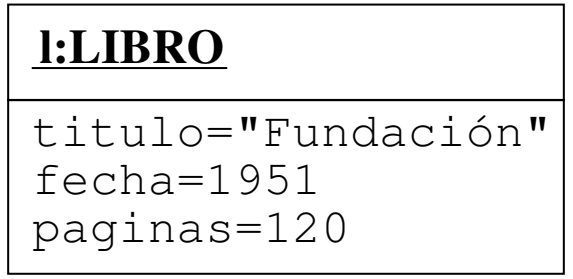
Los objetos pueden ser representados en UML

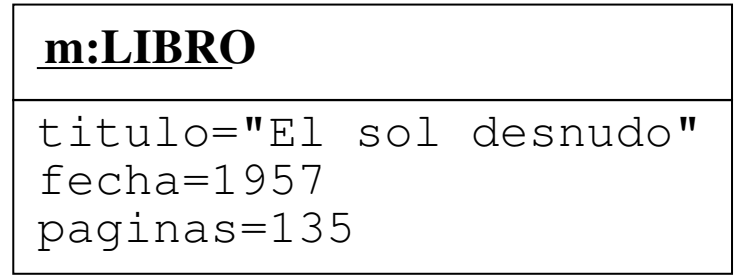


Pero los objetos pueden recibir mensajes y ejecutar métodos

Representación de los objetos

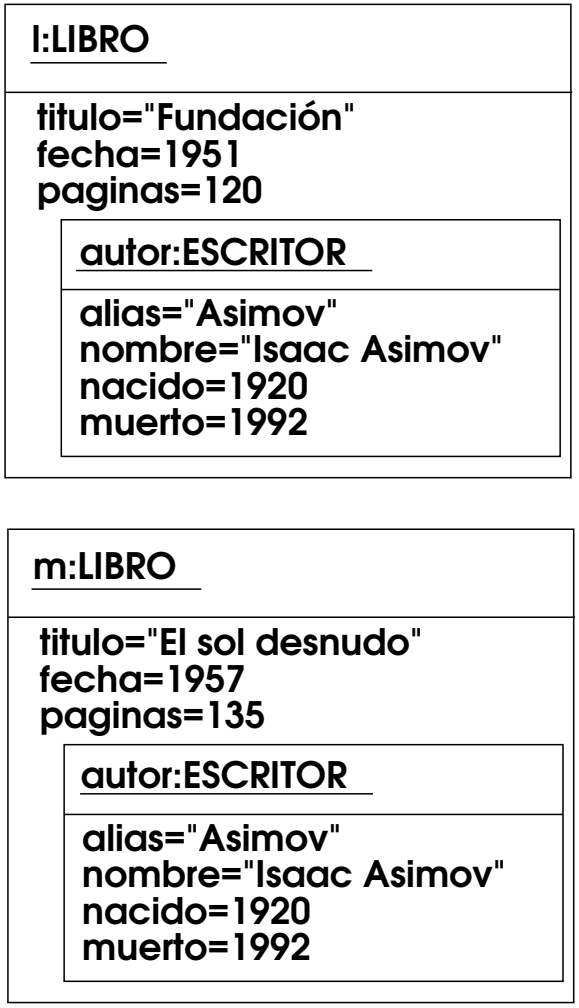


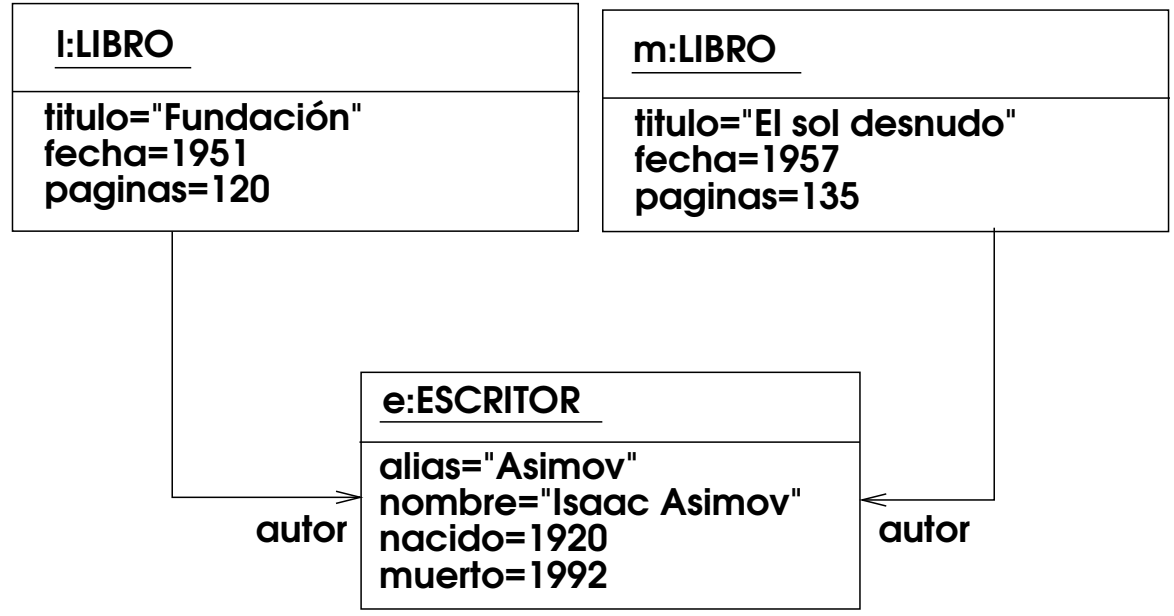




Dos opciones para almacenar el autor

Un autor por libro





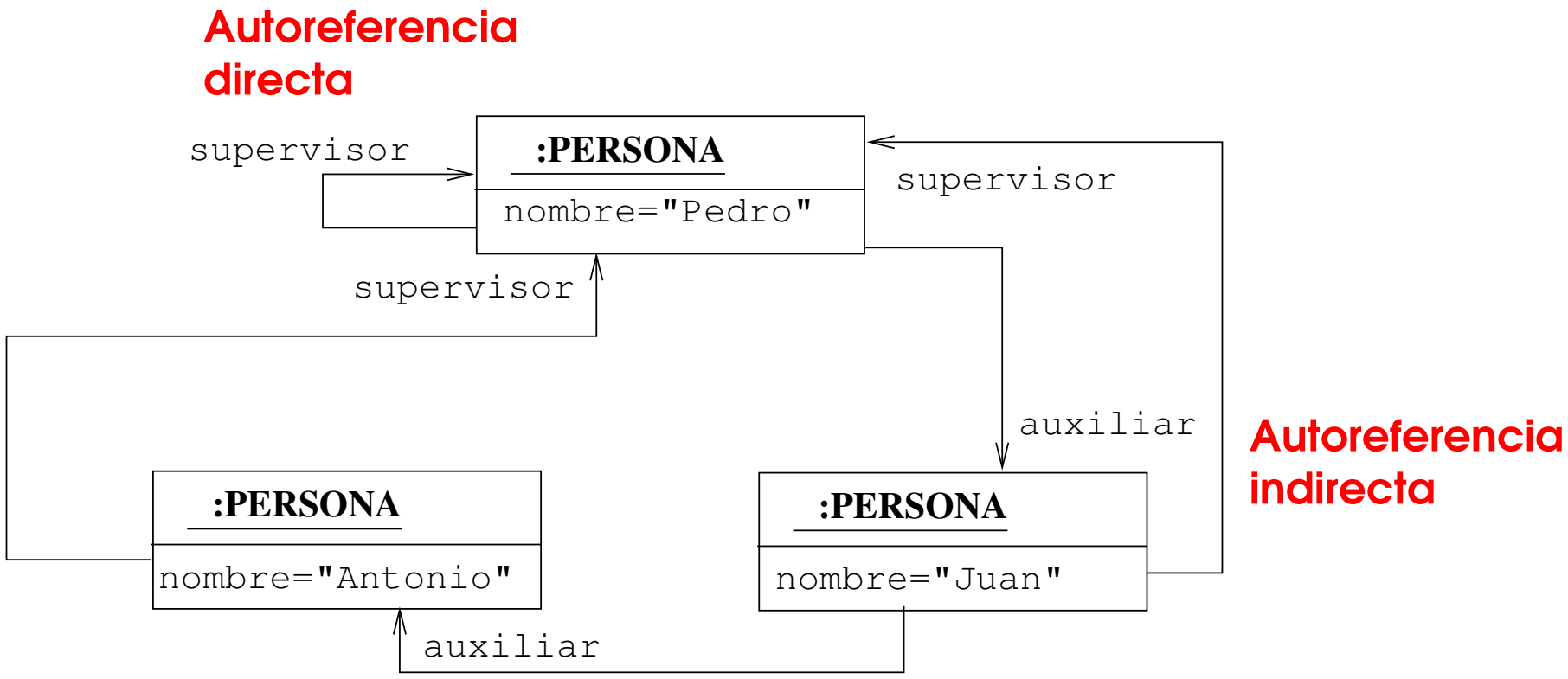
Un autor para todos los libros (Por defecto en Eiffel)

El concepto de referencia

Definición 2 *Una referencia es un valor en tiempo de ejecución que puede estar vacío (*`void`*) o conectado. Cuando una referencia está conectada apunta a un objeto único, y decimos que está conectada al objeto.*

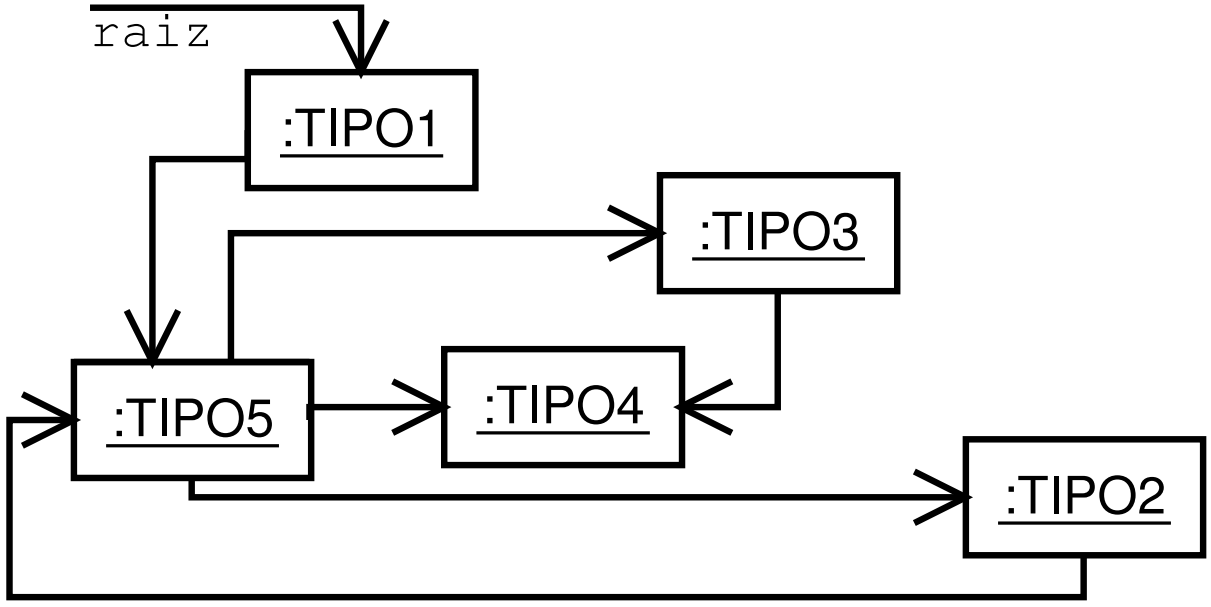
- Cada objeto tiene su propia identidad
- Solo podemos acceder a los objetos mediante referencias
- Eiffel no implementa una aritmética de punteros
- Eiffel no define el modo en que se implementan los punteros

Formas de autoreferencia



La autoreferencia está permitida, tanto entre objetos como entre clases. La autoreferencia entre objetos require autoreferencia entre clases, pero no viceversa.

Los sistemas en ejecución



El objetivo no es que el sistema de objetos sea simple, sino que lo sea el sistema de clases

Creación de una instancia de libro

```
class CITA
feature
   fuente: LIBRO;
   pagina: INTEGER;
   crea_libro is
      do
         create fuente
      end;
end -- class CITA
```

Creación de un objeto

Regla 1 *El efecto de una instrucción de creación* **`create`** x *(o* `!!`x*, en la sintáxis original), donde la referencia* x *está basada en una clase* C *es la ejecución de los siguientes pasos:*

- *Creación de una nueva instancia de* C*, a la que denominaremos* oC
- *Inicialización de cada atributo de* oC *de acuerdo con los valores por defecto.*
- *Conexión de la referencia* x *al nuevo objeto* oC

Inicializaciones por defecto.

- Para las referencias `Void`
- Para los datos numéricos cero.
- Para los datos lógicos `False`
- Para los caracteres el carácter nulo.
- Los atributos de tipo **`STRING`** se inicializan a `Void`, como caso particular de referencias
- Las clases definidas por el programador pueden dar lugar a tipos con inicialización por defecto definida por éste.

Clase con procedimientos de creación

```
cass PUNTO
-- Representa un punto del plano
create {ANY}
  crea_polares, crea_cartesianas
feature {NONE}
  crea_polares(a, b: DOUBLE) is
    -- Inicializa un punto partiendo de sus coordenadas polares
    do
      x := b.cos * a; y := b.sin * a;
    end;
  crea_cartesianas(a, b: DOUBLE) is
    -- Inicializa un punto partiendo de sus coordenadas cartesianas
    do
      x := a; y := b;
    end;
```

Creación de un objeto(II)

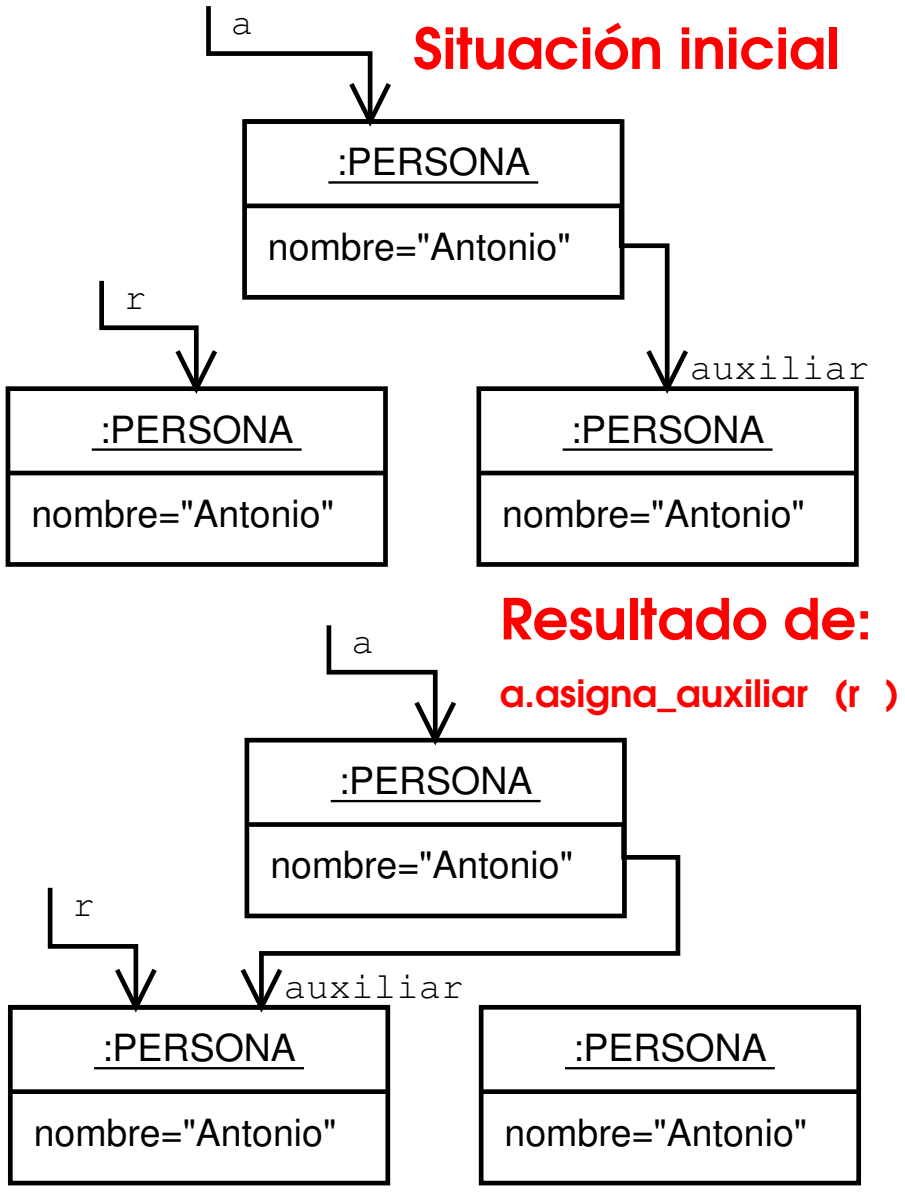
Regla 2 *Una instrucción de creación* **`create`** `x.metodo` *(en la sintáxis original* `!!x.metodo`*) donde la referencia* `x` *está basada en una clase* `C` *tiene por efecto la ejecución de los siguientes pasos:*

- *Creación de una nueva instancia de* `C`*, a la que denominaremos* `OC`
- *Inicialización de cada atributo de* `OC` *de acuerdo con los valores por defecto.*
- *Conexión de la referencia* `x` *al nuevo objeto* `OC`
- *Paso del mensaje* `metodo` *al objeto creado.*

Creación de un objeto (III)

- Si una clase dispone de métodos de creación es obligatorio utilizar alguno de ellos
- Si deseamos que nuestra clase disponga de un método de inicialización que no hace nada podemos declarar como tal el método `do_nothing` disponible en todas las clases
- Los niveles de exportación de un método pueden ser distintos como método de creación y como método convencional

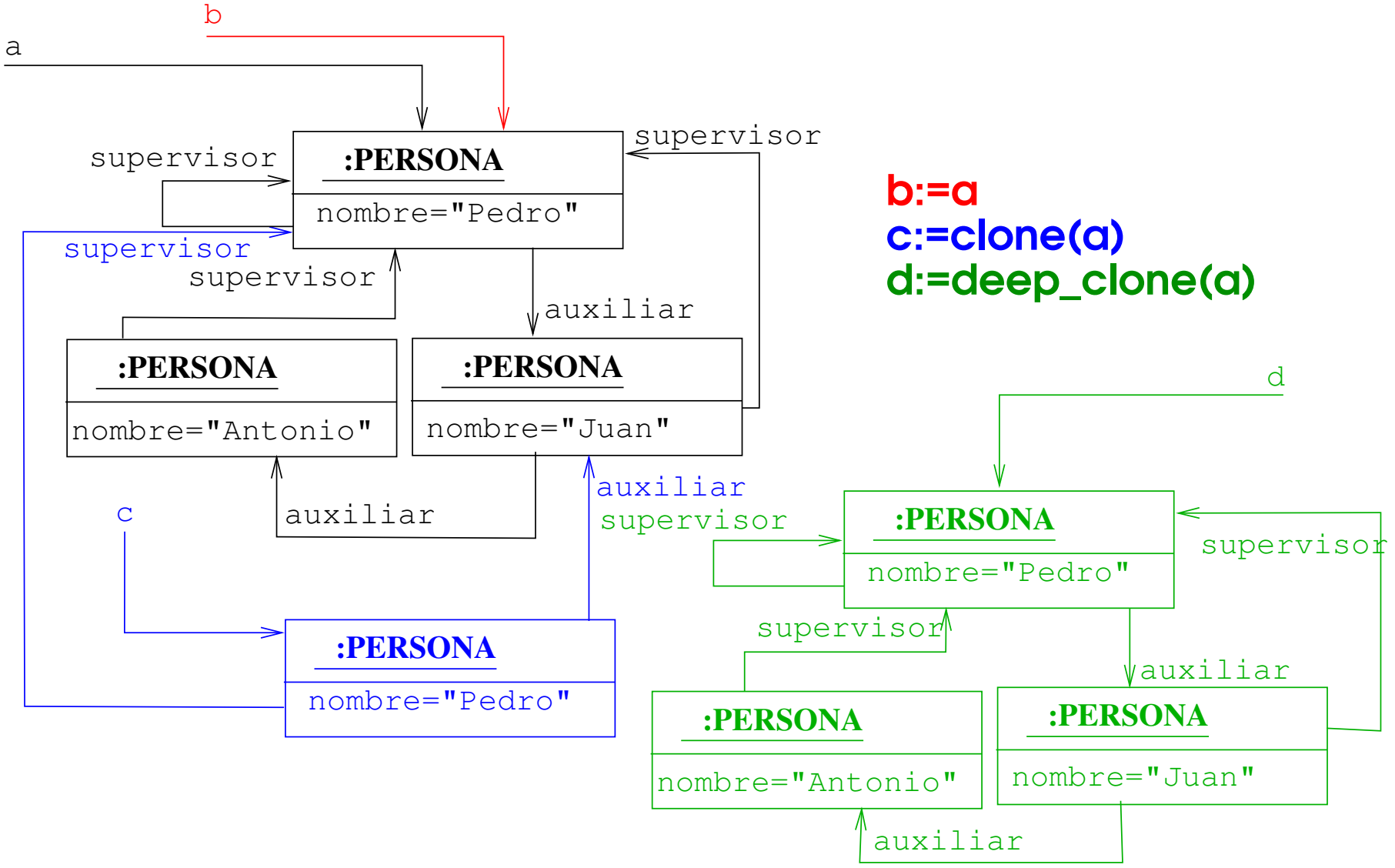
Operaciones con referencias




```
class EMPLEADO
feature
   supervisor, auxiliar : EMPLEADO;
   asigna_auxiliar(p:EMPLEADO) is
   do
      auxiliar := p;
   end
end -- class EMPLEADO
```

Para comparar referencias disponemos de = y /=

Operaciones sobre objetos



Dependencia entre objetos

Definición 3 *Diremos que un objeto depende directamente de otro si uno de sus campos es una referencia conectada a él.*

Un objeto depende de otro si:

- *Depende directamente de él.*
- *Un objeto de que depende directamente depende de él.*

Persistencia y dependencia

Regla 3 *Cuando el sistema almacena un objeto deberá almacenar todos los objetos de que éste depende.*

Regla 4 *Cuando el sistema recupera un objeto deberá recuperar todos los objetos de que éste depende y no han sido previamente recuperados.*

Restricciones de las clases expandidas

Regla 5 *Diremos que* **C** *es cliente expandido de* **D** *si en su definición aparece una referencia con tipo* **expanded** D *o si* **C** *es cliente de* **D** *y* **D** *es una clase expandida.*

Regla 6 *La relación de cliente expandido no puede contener ningún ciclo.*

Regla 7 *Para que una clase pueda ser expandida no debe tener método de creación, o, en su defecto, un sólo método de creación sin parámetros.*

Regla 8 *Si bien un subobjeto puede contener una referencia a un objeto no está permitida las referencias que apuntan a subobjetos.*

El concepto de conexión

Definición 4 *La conexión de* y *a* x *es una de las dos operaciones:*

- *Una asignación de la forma* x `:=` y
- *La substitución del parámetro formal* x *de un método por su correspondiente parámetro actual* y.

En ambos casos decimos que x *es el objetivo de la conexión mientras* y *es la fuente de la misma.*

Las conexiones pueden afectar a objetos expandidos y no expandidos, luego necesitamos establecer claramente su resultado en todos los casos

Efecto de x:=y

Tipo del objetivo x \ **Tipo de la fuente** y	*Referencia*	*Expandido*
Referencia	Conexión por referencia	Como `x:=clone(y)`
Expandido	Como `x.copy(y)` (Falla si `y=Void`)	Como `x.copy(y)`

Efecto de x=y

Tipo de x \ Tipo de y	*Referencia*	*Expandido*
Referencia	Compara referencias	Como `equal` `False` si `x=Void`
Expandido	Como `equal` `False` si `y=Void`	Como `equal`