Programación II

Unidad 4 – Asociación y dependencia

Tecnicatura Universitaria en Desarrollo Web
Facultad de Ciencias de la Administración
Universidad Nacional de Entre Ríos

Objetivos y contenidos

Unidad 4: Asociación y Dependencias

Objetivos:

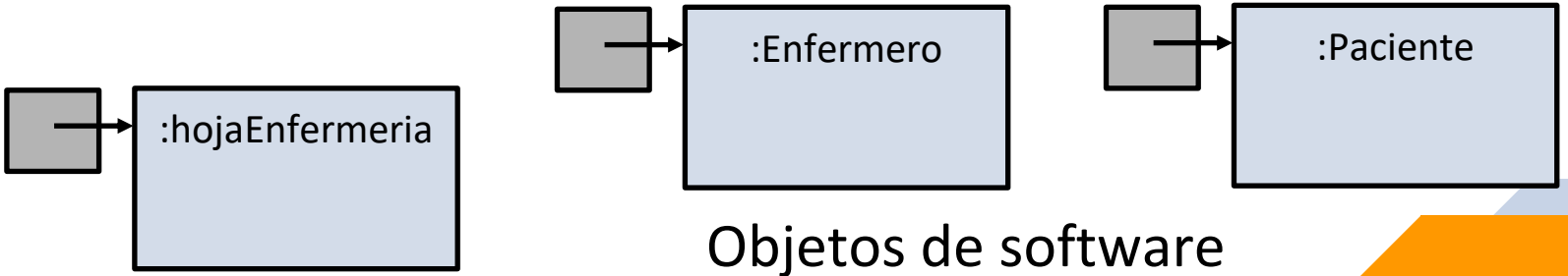
- Identificar cómo hacer uso de asociaciones y dependencias entre clases.
- Comprender la diferencia entre igualdad por valor superficial y en profundidad, y su representación en memoria.

Contenidos:

- Asociación y dependencia entre clases. La relación tieneUn y usaUn. Representación de datos en memoria. Igualdad por valor superficial. Igualdad en profundidad. Copiar y Clonar.

Los objetos

Objetos del problema



Objetos de software

Los objetos

El diagrama de clases

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min: entero max: entero
<<Constructor>> PresionArterial(min,max:entero) **<<Consultas>>** obtenerMin():entero obtenerMax():entero obtenerPulso():entero alarmaHipertension(): boolean equals(pres:PresionArterial):boolean __str__():String

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real **presion: presionArterial**

La clase `SignosVitales` **tiene** un atributo de instancia de clase `PresionArterial`

El diagrama de clases

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min: entero max: entero
<<Constructor>> PresionArterial(min,max:entero) **<<Consultas>>** obtenerMin():entero obtenerMax():entero obtenerPulso():entero alarmaHipertension(): boolean equals(pres:PresionArterial):boolean __str__():String

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real **presion: presionArterial**
<<Constructor>> SignosVitale(temp:real,**pres:PresionArterial**)

El diagrama de clases

PresionArterial

<<Atributo de clase>>
umbralMin=75
umbralMax=140
<<Atributo de instancia>>
min: entero
max: entero

<<Constructor>>
PresionArterial(min,max:entero)
<<Consultas>>
obtenerMin():entero
obtenerMax():entero
obtenerPulso():entero
alarmaHipertension(): boolean
equals(pres:PresionArterial):boolean
__str__():String

SignosVitales

<<Atributo de clase>>
umbralTemp=37.5
<<Atributo de instancia>>
temperatura: real
presion: presionArterial

<<Constructor>>
SignosVitale(temp:real,**pres:PresionArterial**)
<<Consultas>>
obtenerTemperatura():real
obtenerPresion():**PresionArterial**
alarma():boolean
__str__():String
equals(sigVital:SignosVitales):boolean

Clases clientes y proveedoras

PresionArterial

PROVEEDORA

<<Atributo de clase>>
umbralMin=75
umbralMax=140
<<Atributo de instancia>>
min: entero
max: entero

<<Constructor>>
PresionArterial(min,max:entero)
<<Consultas>>
obtenerMin():entero
obtenerMax():entero
obtenerPulso():entero
alarmaHipertension(): boolean
equals(pres:PresionArterial):boolean
__str__():String

SignosVitales

CLIENTE

<<Atributo de clase>>
umbralTemp=37.5
<<Atributo de instancia>>
temperatura: real
presion: presionArterial

<<Constructor>>
SignosVitale(temp:real,**pres:PresionArterial**)
<<Consultas>>
obtenerTemperatura():real
obtenerPresion():**PresionArterial**
alarma():boolean
__str__():String
equals(sigVital:SignosVitales):boolean

```
registro.__str__()
```

La especificación del problema

La temperatura está expresada en grados centígrados

SignosVitales(temp:real,pres:PresionArterial)
Requiere pres ligado y temp > 30

alarma(): boolean
Retorna true si
temperatura > umbralTemp o hay
Alarma de Hipertension

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real **presion: presionArterial**
<<Constructor>> SignosVitale(temp:real,**pres:PresionArterial**) **<<Consultas>>** obtenerTemperatura():real obtenerPresion():**PresionArterial** alarma():boolean __str__():String equals(sigVital:SignosVitales):boolean

Asociación entre clases

```
class SignosVitales():
      #Atributos de classe
      umbralTemp=37.5
      def __init__(self,temp,pres):
          #Atributos de instancia
          self.temperatura = temp
          self.presion = pres
```

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real **presion: presionArterial**
<<Constructor>> SignosVitale(temp:real,**pres:PresionArterial**)

Las clase **SignosVitales** y **PresionArterial** están asociadas

La clase **SignosVitales** es cliente de la clase **PresionArterial**

Asociación entre clases

```
def obtenerTemperatura(self):
    return self.temperatura

def obtenerPresion(self):
    return self.presion
```

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real **presion: presionArterial**
<<Constructor>> SignosVitale(temp:real,**pres:PresionArterial**) **<<Consultas>>** obtenerTemperatura():real obtenerPresion():**PresionArterial**

Asociación entre clases

```
def alarma(self):
    '''Retorna true si
    temperatura > umbralTemp o hay
    Alarma de Hipertension'''
    return self.temperatura > self.umbralTemp or
           self.presion.alarmaHipertension()
```

alarma(): boolean
Retorna true si
temperatura > umbralTemp o hay
Alarma de Hipertension

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real **presion: presionArterial**
<<Constructor>> SignosVitale(temp:real,**pres:PresionArterial**) **<<Consultas>>** obtenerTemperatura():real obtenerPresion():**PresionArterial** **alarma():boolean**

La clase **`SignosVitales`** necesita saber si hay alarma de hipertensión, pero no necesita saber CÓMO se implementa

La relación de asociación entre clases

Cuando una clase **tiene un** atributo de instancia de otra clase, decimos que las clases están **asociadas** y la relación es de tipo **tieneUn**.

El diagrama de las clases asociadas

PresionArterial

<<Atributo de clase>>
umbralMin=75
umbralMax=140
<<Atributo de instancia>>
min: entero
max: entero

<<Constructor>>
PresionArterial(min,max:entero)
<<Consultas>>
obtenerMin():entero
obtenerMax():entero
obtenerPulso():entero
alarmaHipertension(): boolean
equals(pres:PresionArterial):boolean
__str__():String

SignosVitales

<<Atributo de clase>>
umbralTemp=37.5
<<Atributo de instancia>>
temperatura: real
presion: presionArterial

<<Constructor>>
SignosVitale(temp:real,**pres:PresionArterial**)
<<Consultas>>
obtenerTemperatura():real
obtenerPresion():**PresionArterial**
alarma():boolean

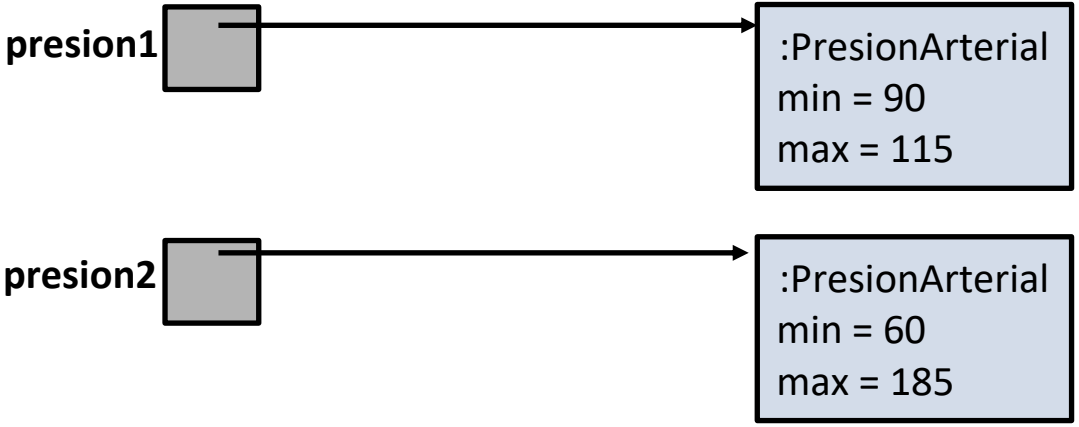
alarma(): boolean
Retorna true si
temperatura > umbralTemp o
alarmaHipertension retorna verdadero

La clase tester

PresionArterial

```
def __init__(self,min,max):
        # requiere max > min > 0
        self.min=min
        self.max=max
```

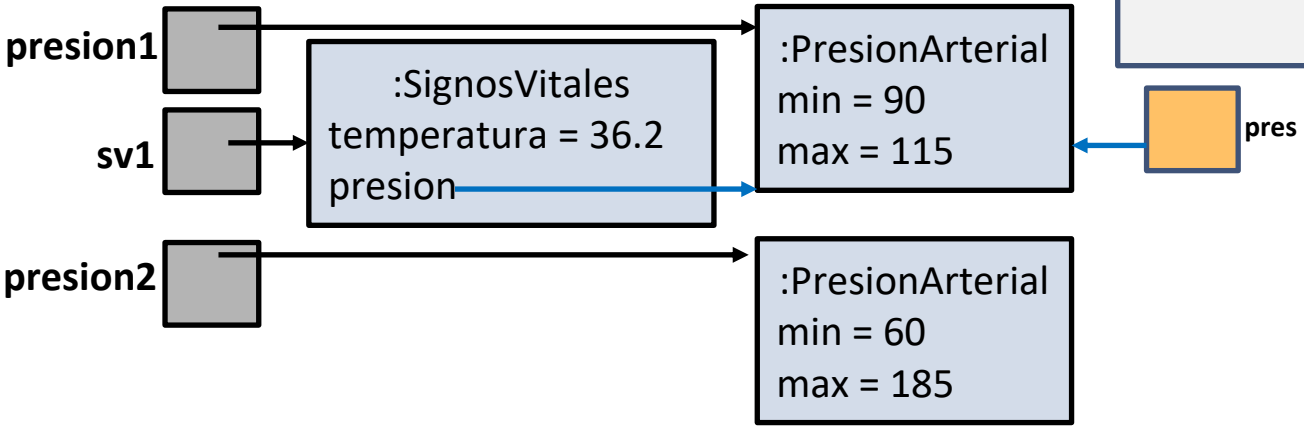
```
presion1 = PresionArterial(90,115)
presion2 = PresionArterial(60,185)
```



Asociación entre clases

```
def __init__(self,temp,pres):
        #Atributos de instancia
        self.temperatura = temp
        self.presion = pres
```

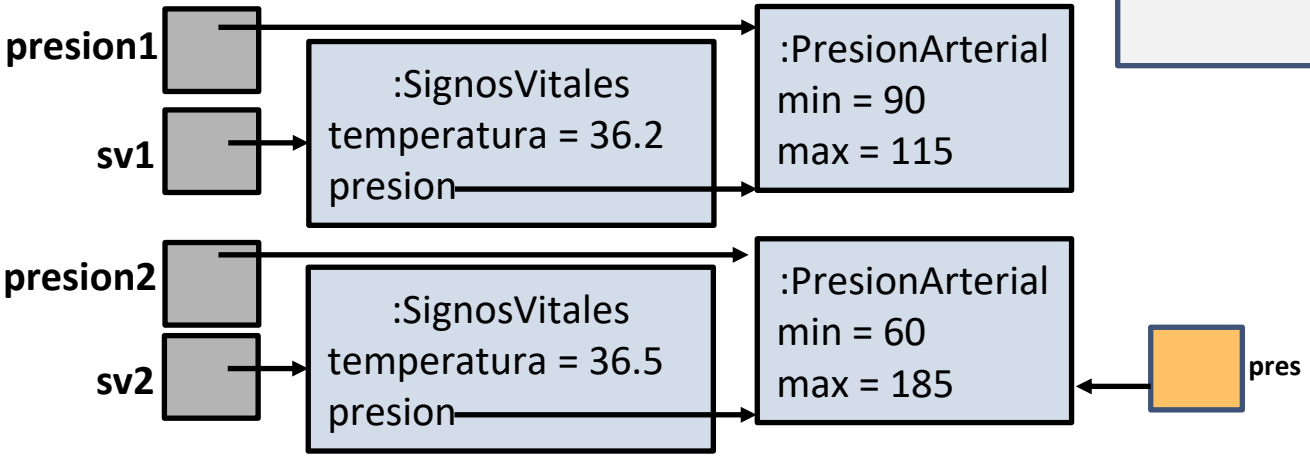
```
presion1 = PresionArterial(90,115)
presion2 = PresionArterial(60,185)
sv1 = SignosVitales(36.2,presion1)
```



Asociación entre clases

```
def __init__(self,temp,pres):
        #Atributos de instancia
        self.temperatura = temp
        self.presion = pres
```

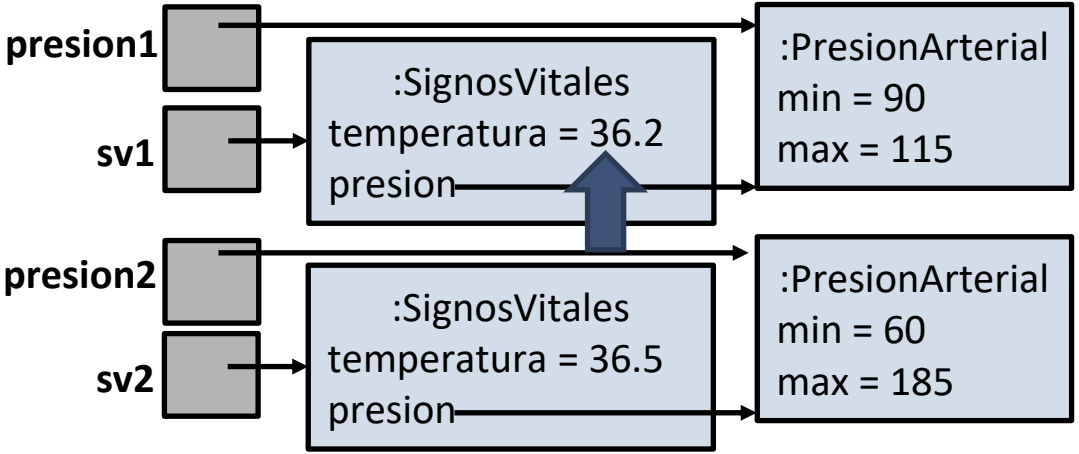
```
presion1 = PresionArterial(90,115)
presion2 = PresionArterial(60,185)
sv1 = SignosVitales(36.2,presion1)
sv2 = SignosVitales(36.5,presion2)
```



Asociación entre clases

```
def alarma(self)
    '''Retorna true si
    temperatura > umbralTemp o hay
    Alarma de Hipertension'''
    return self.temperatura > self.umbralTemp or
           self.presion.alarmaHipertension()
```

```
presion1 = PresionArterial(90,115)
presion2 = PresionArterial(60,185)
sv1 = SignosVitales(36.2,presion1)
sv2 = SignosVitales(36.5,presion2)
if sv1.alarma():
   print("Alarma a las 6")
```



umbralTemp

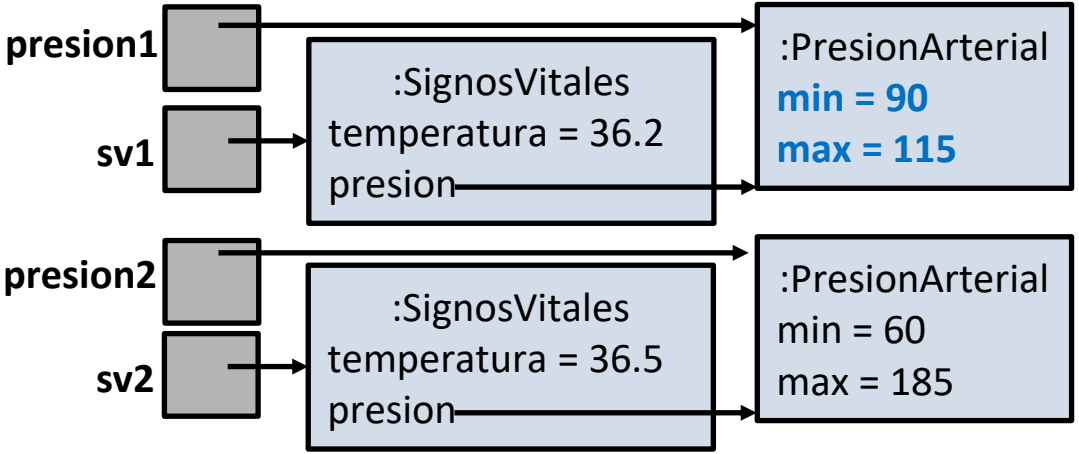
36.2 > 37.5

Asociación entre clases

```
def alarmaHipertension(self):
    '''
    retorna true si
    max > umbralMax o
    min < umbralMin
    '''
    return self.max > self.umbralMax or self.min <
           self.umbralMin
```

PresionArterial

```
presion1 = PresionArterial(90,115)
presion2 = PresionArterial(60,185)
sv1 = SignosVitales(36.2,presion1)
sv2 = SignosVitales(36.5,presion2)
if sv1.alarma():
   print("Alarma a las 6")
```



umbralMin

90 < 75

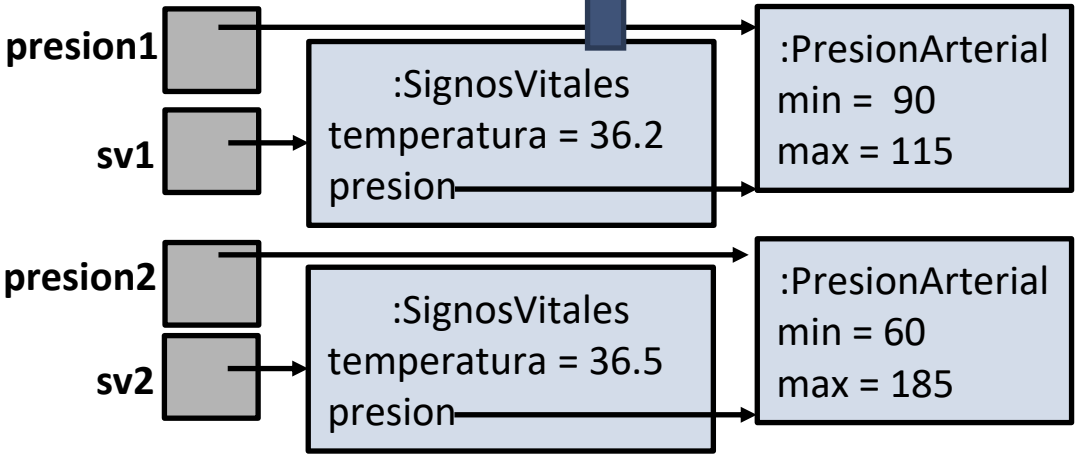
115 > 140

umbralMax

Asociación entre clases

```
def alarma(self)
    '''Retorna true si
    temperatura > umbralTemp o hay
    Alarma de Hipertension'''
    return self.temperatura > self.umbralTemp or
           self.presion.alarmaHipertension()
```

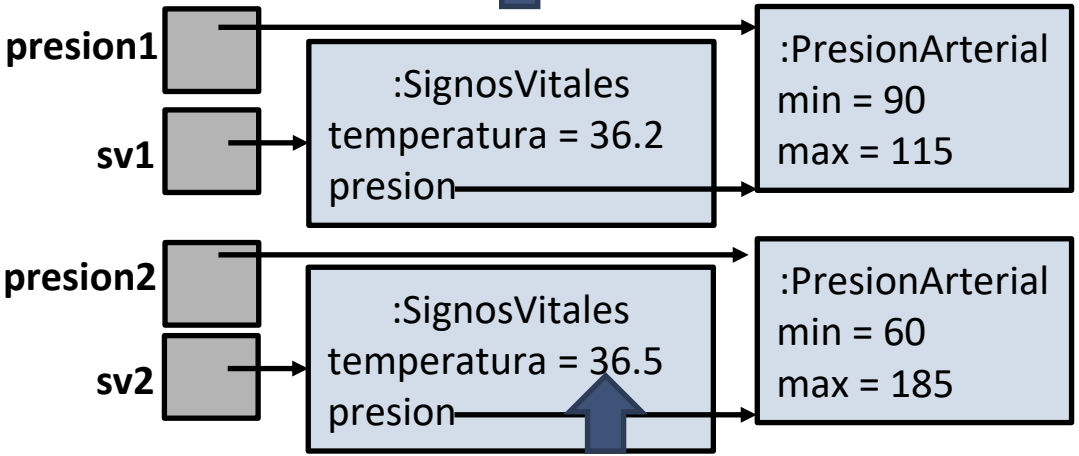
```
presion1 = PresionArterial(90,115)
presion2 = PresionArterial(60,185)
sv1 = SignosVitales(36.2,presion1)
sv2 = SignosVitales(36.5,presion2)
if sv1.alarma():
   print("Alarma a las 6")
```



Asociación entre clases

```
def alarma(self)
    '''Retorna true si
    temperatura > umbralTemp o hay
    Alarma de Hipertension'''
    return self.temperatura > self.umbralTemp or
           self.presion.alarmaHipertension()
```

```
presion1 = PresionArterial(90,115)
presion2 = PresionArterial(60,185)
sv1 = SignosVitales(36.2,presion1)
sv2 = SignosVitales(36.5,presion2)
if sv1.alarma():
   print("Alarma a las 6")
if sv2.alarma():
   print("Alarma a las 12")
```



umbralTemp

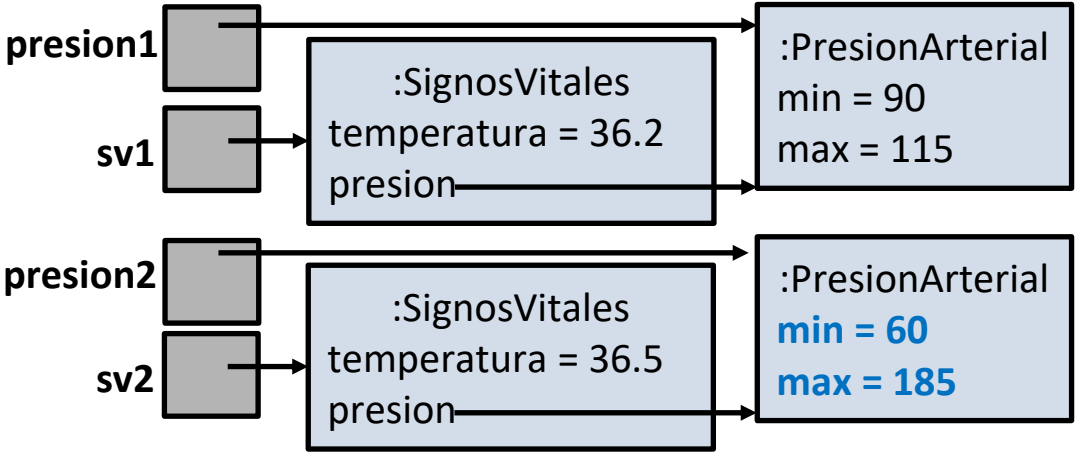
36.5 > 37.5

Asociación entre clases

```
def alarmaHipertension(self):
    '''
    retorna true si
    max > umbralMax o
    min > umbralMin
    '''
    return self.max > self.umbralMax or self.min >
           self.umbralMin
```

PresionArterial

```
presion1 = PresionArterial(60,115)
presion2 = PresionArterial(90,185)
sv1 = SignosVitales(36.2,presion1)
sv2 = SignosVitales(36.5,presion2)
if sv1.alarma():
   print("Alarma a las 6")
if sv2.alarma():
   print("Alarma a las 12")
```



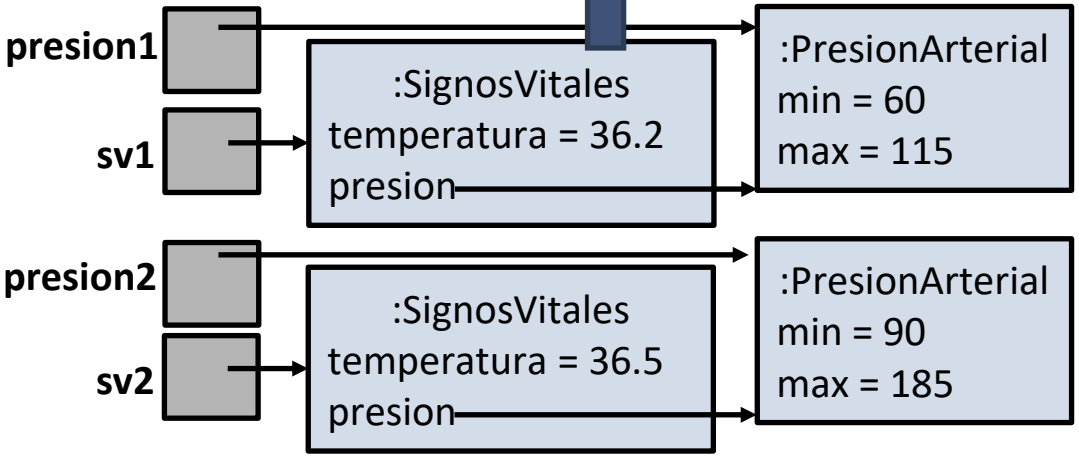
umbralMin

60 < 75
185 > 140

umbralMax

Asociación entre clases

```
def alarma(self)
    '''Retorna true si
    temperatura > umbralTemp o hay
    Alarma de Hipertension'''
    return self.temperatura > self.umbralTemp or
           self.presion.alarmaHipertension()
```



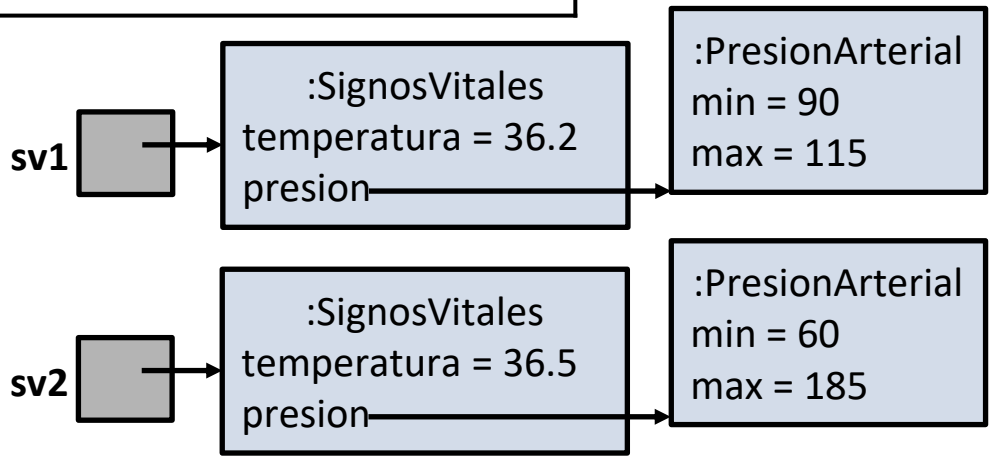

```
presion1 = PresionArterial(60,115)
presion2 = PresionArterial(90,185)
sv1 = SignosVitales(36.2,presion1)
sv2 = SignosVitales(36.5,presion2)
if sv1.alarma():
   print("Alarma a las 6")
if sv2.alarma():
   print("Alarma a las 12")
```

Alarma a las 12

Asociación entre clases

PresionArterial
<<Atributo de clase>> **umbralMin=75** **umbralMax=140** **<<Atributo de instancia>>** min: entero max: entero

SignosVitales
<<Atributo de clase>> **umbralTemp=37.5** **<<Atributo de instancia>>** temperatura: real presion: presionArterial



Entender qué dice el diagrama, cuándo el estado de un paciente dispara una alarma.

Alarma a las 12

Clientes y proveedoras

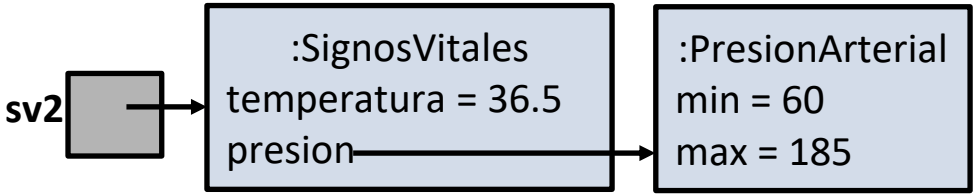
La clase `PresionArterial` **brinda** servicios que la clase SignosVitales **usa**.
Decimos que clase `PresionArterial` cumple el rol de **proveedora** y `SignosVitales` es su **cliente**.
La clase `tester` también **usa** los servicios de `PresionArterial` y además **usa** a `SignosVitales`.
De modo que `SignosVitales` es al mismo tiempo **cliente** y **proveedora**.

__str__ y clases asociadas

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real presion: presionArterial
__str__():String

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min,max: entero
__str__():String

```
print(sv2.__str__())
```



36.5 60 185

__str__ y clases asociadas

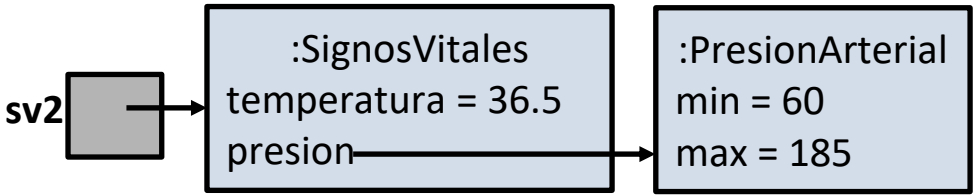
SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real presion: presionArterial
__str__():String

```
def __str__(self):
    return str(self.temperatura)+' '+
           self.presión.__str__()
```

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min,max: entero
__str__():String

```
def __str__(self):
    return str(self.min)+' '+str(self.max)
```

```
print(sv2.__str__())
```



36.5 60 185

__str__ y clases asociadas

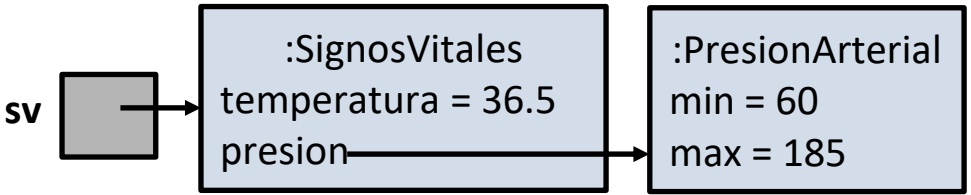
SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real presion: presionArterial
__str__():String

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min,max: entero
__str__():String

```
def __str__(self):
    return str(self.temperatura)+' '+
           self.presión.__str__()
```

```
def __str__(self):
    return str(self.min)+' '+str(self.max)
```

```
print(sv2.__str__())
```



36.5 60 185

__str__ y clases asociadas

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real presion: presionArterial
__str__():String

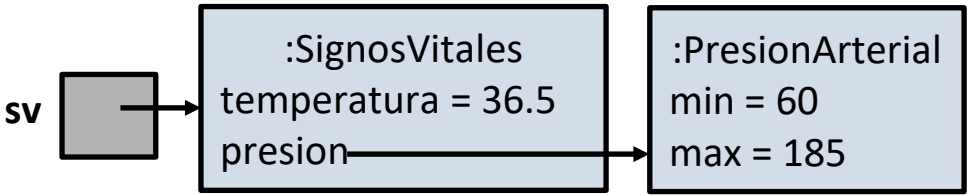
PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min,max: entero
__str__():String

```
def __str__(self):
    return str(self.temperatura)+' '+
           self.presión.__str__()
```

```
def __str__(self):
    return str(self.min)+' '+str(self.max)
```

```
print(sv2.__str__())
```

'36.5 60 185'



36.5 60 185

__str__ y clases asociadas

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real presion: presionArterial
__str__():String

```
def __str__(self):
    return str(self.temperatura)+' '+
           self.presión.__str__()
```

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min,max: entero
__str__():String

```
def __str__(self):
    return str(self.min)+' '+str(self.max)
```

La clase `SignosVitales` define un método `__str__()` que envía el mensaje `__str__()` a un objeto de clase `PresioArterial`.

Mensajes y métodos

La clase del objeto determina la ligadura entre un mensaje y un método.

__str__ y clases asociadas

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real presion: presionArterial
__str__():String

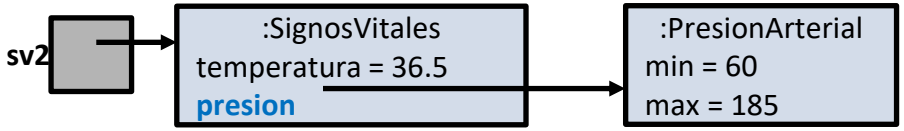
```
def __str__(self):
    return str(self.temperatura)+' '+
           self.presión.__str__()
```

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min,max: entero
__str__():String

```
def __str__(self):
    return str(self.min)+' '+str(self.max)
```

`sv2.__str__()` el mensaje se liga al método __str__() definido en `SignosVitales`

`self.presion.__str__()` el mensaje se liga al método `__str__()` definido en `PresionArterial`



Clientes y proveedoras

La clase `SignosVitales` puede implementarse conociendo qué hace la clase `PresionArterial`, pero no cómo lo hace.

La clase `PresionArterial` puede implementarse sin saber que va a ser usada por la clase `SignosVitales`.

Es decir, cada clase debe conocer los servicios que brindan sus clases proveedoras, pero no necesita conocer quienes son sus clientes.

Cada clase va a ser verificada por separado y luego en conjunto con las demás clases relacionadas.

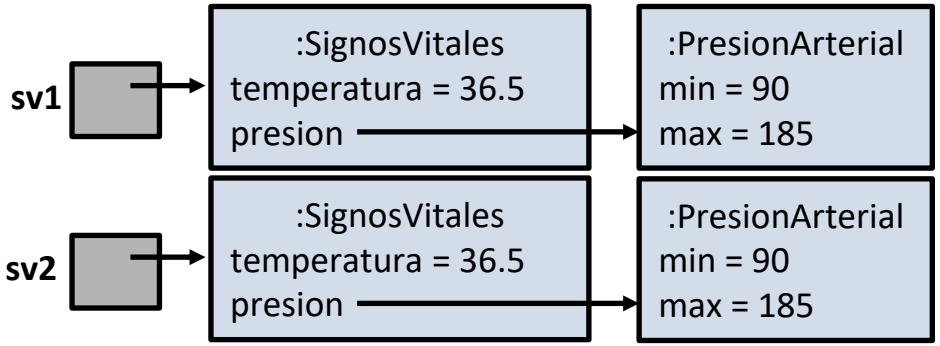
Las responsabilidades establecen un **contrato** entre una clase, sus clientes y sus proveedores.

equals

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real presion: presionArterial
equals(sigVital:SignosVitales):boolean

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min,max: entero
equals(pres:PresionArterial):boolean

```
if(sv1.equals(sv2)):
```

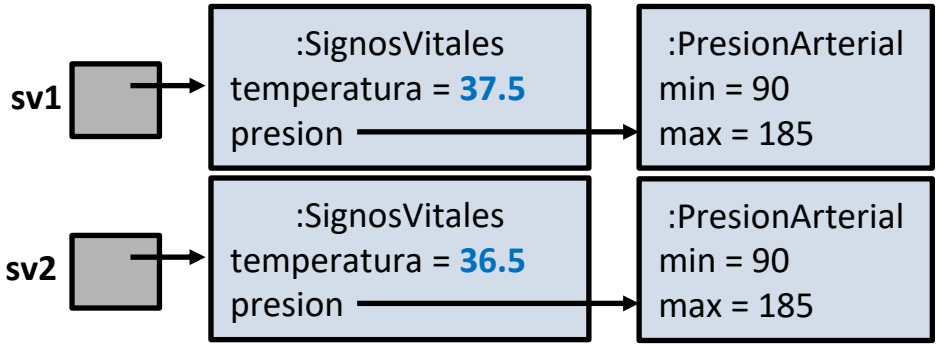


equals

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real presion: presionArterial
equals(sigVital:SignosVitales):boolean

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min,max: entero
equals(pres:PresionArterial):boolean

```
if(sv1.equals(sv2)):
```

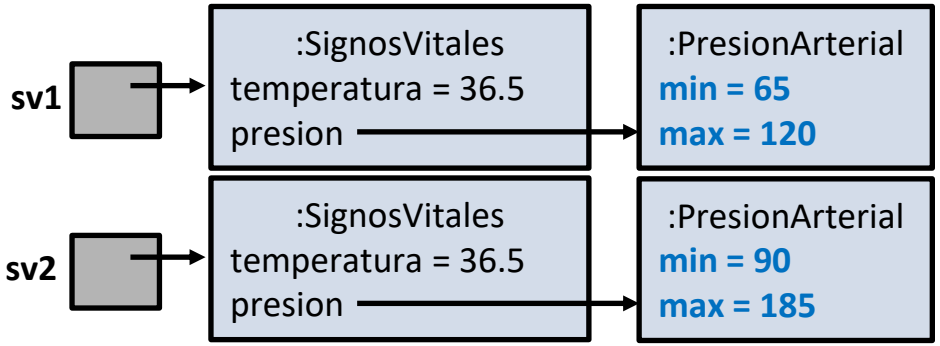


equals

SignosVitales
<<Atributo de clase>> umbralTemp=37.5 **<<Atributo de instancia>>** temperatura: real presion: presionArterial
equals(sigVital:SignosVitales):boolean

PresionArterial
<<Atributo de clase>> umbralMin=75 umbralMax=140 **<<Atributo de instancia>>** min,max: entero
equals(pres:PresionArterial):boolean

```
if(sv1.equals(sv2)):
```



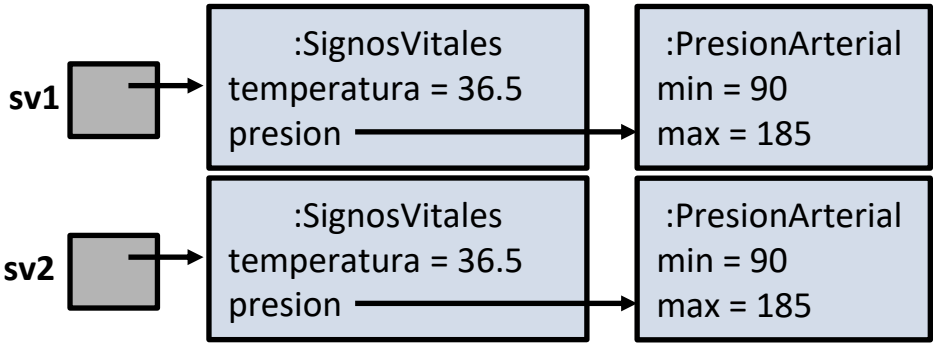
equals

SignosVitales

```
def equals(self,sigVital):
    # requiere sigVital ligado
    return self.temperatura == sigVital.obtenerTemperatura()
and self.presion.equals(sigVital.obtenerPresion())
```

PresionArterial

```
def equals(self,pres):
    # requiere pres ligado
    return self.min == pres.obtenerMin()
and self.max == pres.obtenerMax()
```



equals

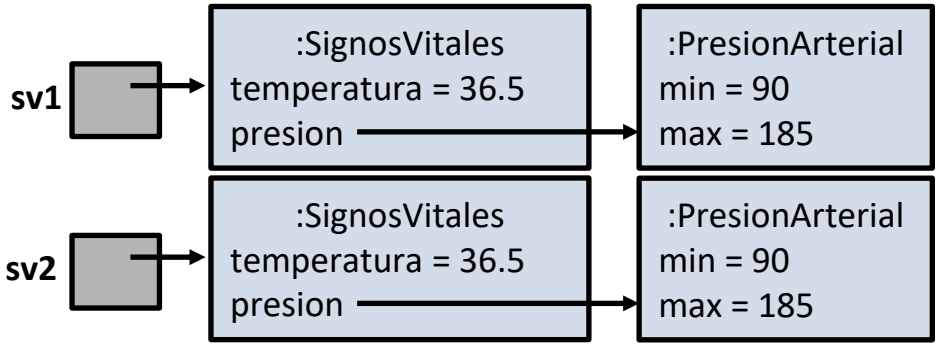
SignosVitales

```
def equals(self,sigVital):
    # requiere sigVital ligado
    return self.temperatura == sigVital.obtenerTemperatura()
and self.presion.equals(sigVital.obtenerPresion())
```

PresionArterial

```
def equals(self,pres):
    # requiere pres ligado
    return self.min == pres.obtenerMin()
and self.max == pres.obtenerMax()
```

```
sv1 = SignosVitales(36.5,PresionArterial(90,185))
sv2 = SignosVitales(36.5,PresionArterial(90,185))
```



Son diferentes mediciones, aunque la temperatura y la presión arterial del paciente fue la misma en las dos oportunidades.

equals

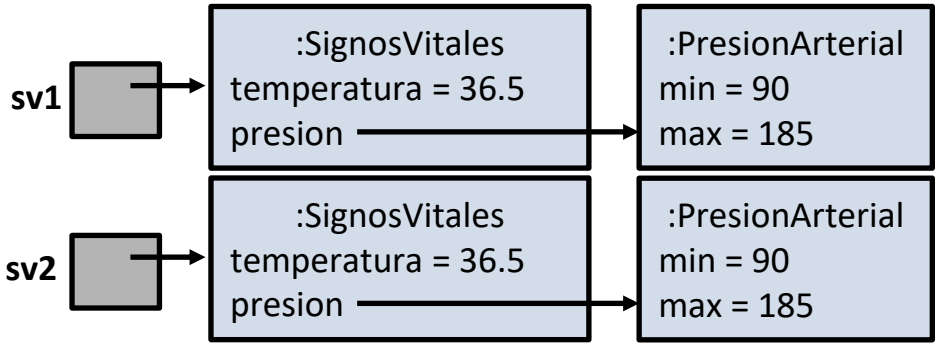
SignosVitales

```
def equals(self,sigVital):
    # requiere sigVital ligado
    return self.temperatura == sigVital.obtenerTemperatura()
and self.presion.equals(sigVital.obtenerPresion())
```

PresionArterial

```
def equals(self,pres):
    # requiere pres ligado
    return self.min == pres.obtenerMin()
and self.max == pres.obtenerMax()
```

```
sv1 = SignosVitales(36.5,PresionArterial(90,185))
sv2 = SignosVitales(36.5,PresionArterial(90,185))
if sv1.equals(sv2):
```



Los objetos de clase `PresionArterial` tienen el mismo **estado interno** pero diferente **identidad**.

equals

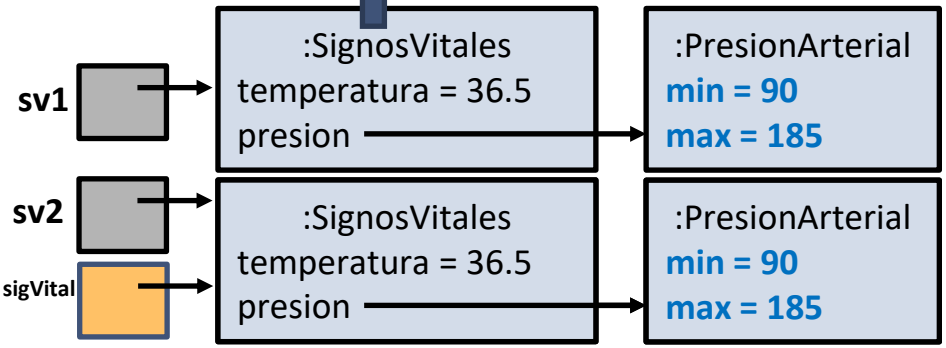
SignosVitales

```
def equals(self,sigVital):
    # requiere sigVital ligado
    return self.temperatura == sigVital.obtenerTemperatura()
and self.presion.equals(sigVital.obtenerPresion())
```

PresionArterial

```
def equals(self,pres):
    # requiere pres ligado
    return self.min == pres.obtenerMin()
and self.max == pres.obtenerMax()
```

```
if sv1.equals(sv2):
```



Los objetos de clase `PresionArterial` tienen el mismo **estado interno** pero diferente **identidad**.

equals

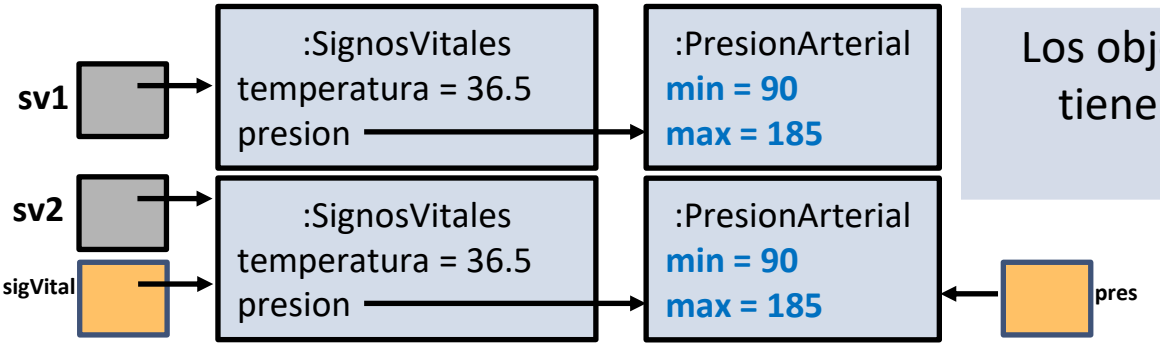
SignosVitales

```
def equals(self,sigVital):
    # requiere sigVital ligado
    return self.temperatura == sigVital.obtenerTemperatura()
and self.presion.equals(sigVital.obtenerPresion())
```

PresionArterial

```
def equals(self,pres):
    # requiere pres ligado
    return self.min == pres.obtenerMin()
and self.max == pres.obtenerMax()
```

```
if sv1.equals(sv2):
```



Los objetos de clase `PresionArterial` tienen el mismo **estado interno** pero diferente **identidad**.

equals

SignosVitales

```
def equals(self,sigVital):
    # requiere sigVital ligado
    return self.temperatura == sigVital.obtenerTemperatura()
and self.presion.equals(sigVital.obtenerPresion())
```

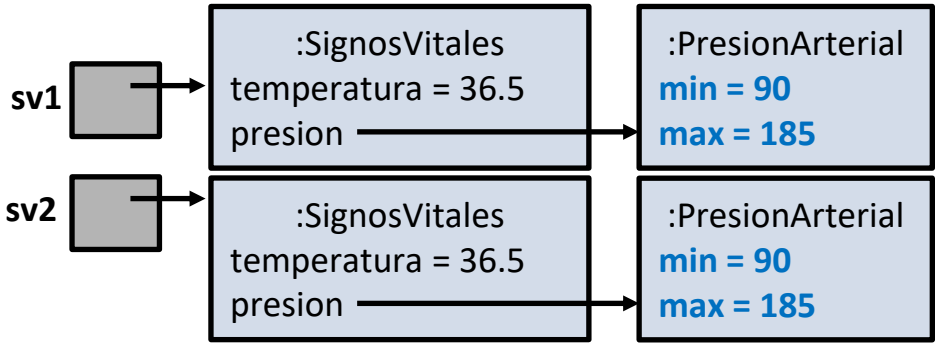
PresionArterial

```
def equals(self,pres):
    # requiere pres ligado
    return self.min == pres.obtenerMin()
and self.max == pres.obtenerMax()
```

True

True

```
if sv1.equals(sv2):
```



Cada medición es un objeto del problema diferente, cada objeto del problema se modela con un objeto de software diferente, aunque tengan los mismos valores

Tarea

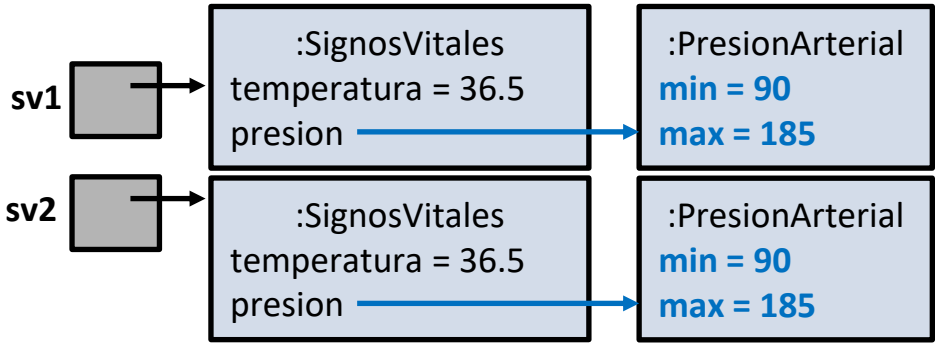
SignosVitales

```
def equals(self,sigVital):
    # requiere sigVital ligado
    return self.temperatura == sigVital.obtenerTemperatura()
and self.presión == sigVital.obtenerPresion()
```

PresionArterial

```
def equals(self,pres):
    # requiere presion ligado
    return self.min == pres.obtenerMin()
and self.max == pres.obtenerMax()
```

```
sv1 = SignosVitales(36.5,PresionArterial(90,185))
sv2 = SignosVitales(36.5,PresionArterial(90,185))
if sv1.equals(sv2):
```



¿Qué valor computa la expresión condicional si `equals` en la clase `SignosVitales` compara la identidad con ==?

Videojuego infantil

Videojuego infantil

Videojuego infantil

En un videojuego para niños algunos de los personajes son **aliens**.
Cada alien se crea con una **cantidad de vidas**, entre 0 y 5, que se van reduciendo cada vez que recibe una herida. Cuando está muerto ya no tienen efecto las heridas, su fuerza es 0 y no puede recuperar vidas.
Cuando un alien logra llegar a la base recupera 1 vida, sin superar nunca el valor 5.
Los aliens tienen cierta cantidad de **ojos** y de **manos**, que determinan su capacidad visual y su capacidad de lucha respectivamente. Un alien tiene también una nave espacial.

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**

La clase Alien

Alien(nave:NaveEspacial,vidas,ojos,manos:entero)
Requiere nave ligado y 0 < vidas <= 5

establecerNave(nave:NaveEspacial)
Requiere nave ligada

recuperaVidas()
Incrementa en 1 las vidas hasta llegar a maxVidas

recibeHerida ()
Pierde 1 vida hasta llegar a 0

copy(alien:Alien)
Requiere alien ligado. Copia el estado interno del alien "alien" en el estado interno del alien que recibe el mensaje.

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Constructor>> Alien(nave:**NaveEspacial**,vidas,ojos,manos:entero) **<<Comandos>>** establecerNave(nave:NaveEspacial) establecerOjos(ojos:entero) establecerManos(manos:entero) recuperaVidas() recibeHerida() copy(a:Alien)

La clase cliente de `Alien` tiene la responsabilidad de asegurar que el parámetro real **nave** está ligado.

La clase Alien

obtenerFuerza():real
Requiere nave ligado. Retorna la capacidad visual, más su capacidad de lucha, todo multiplicado por el número de vidas por un quinto de la cantidad de combustible de la nave

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero **obtenerFuerza():real** clone():Alien equals(alien: Alien): boolean

La consulta `obtenerFuerza` de la clase `Alien` envía el mensaje `obtenerCombustible` a un objeto de la clase asociada Nave.

La clase Alien

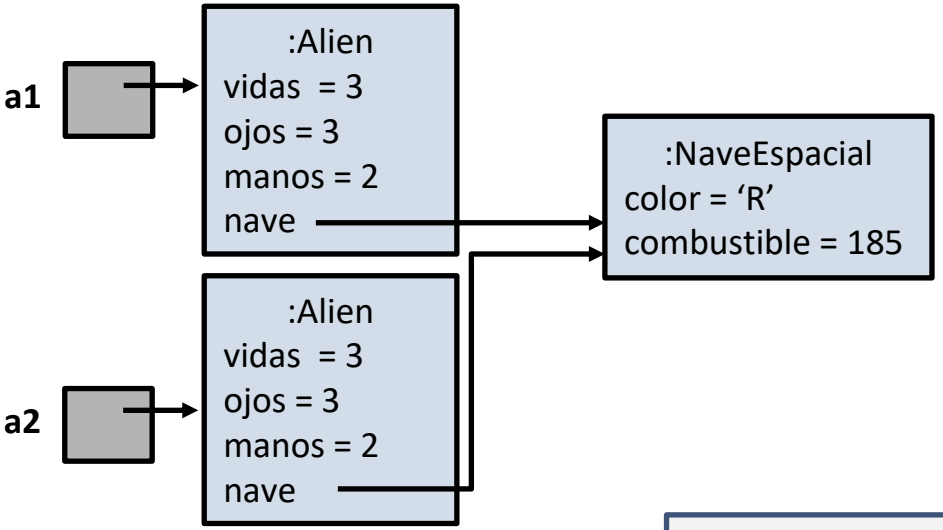
clone():Alien
Crea y retorna un alien con los mismos valores en cada uno de los atributos

equals(alien: Alien): boolean
Requiere alien ligado. Dos aliens son equivalentes si tienen la misma cantidad de vidas, ojos y manos y están ligados a la misma nave

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero **obtenerFuerza():real** **clone():Alien** **equals(alien: Alien): boolean**

La clase cliente de `Alien` tiene la responsabilidad de asegurar que el parámetro real está ligado cuando se envía el mensaje `equals` a un alien.

Identidad

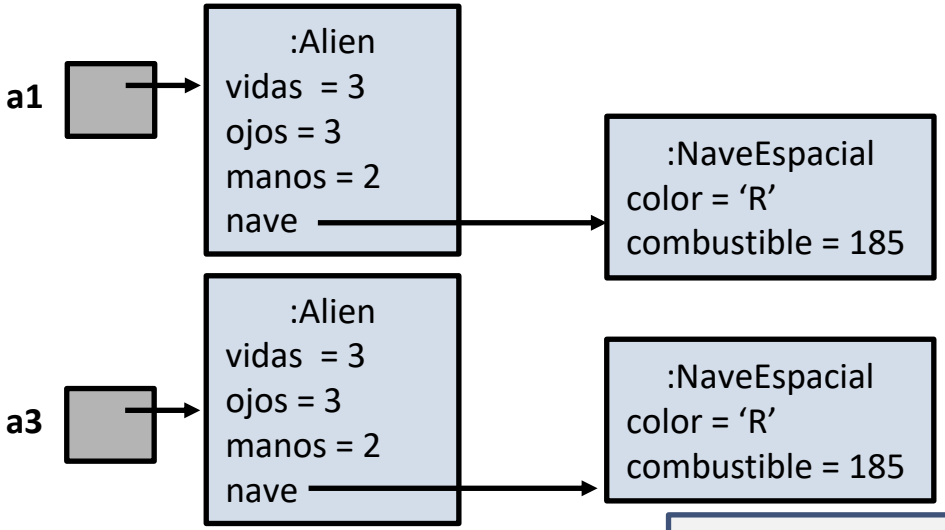



```
a1.equals(a2)
```

equals(alien: Alien): boolean
Requiere alien ligado. Dos aliens son equivalentes si tienen la misma cantidad de vidas, ojos y manos y están ligados a la misma nave

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien equals(alien: Alien): boolean

Equivalencia




```
a1.equals(a3)
```

equals(alien: Alien): boolean
Requiere alien ligado. Dos aliens son equivalentes si tienen la misma cantidad de vidas, ojos y manos y están ligados a la misma nave

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien equals(alien: Alien): boolean

La asociación entre Alien y NaveEspacial

```
class Alien():
    #Atributos de classe
    maxVidas = 5
    def __init__(self,nave,vidas,ojos,manos):
        # requiere nave ligado y 0<vidas<=5
        self.vidas=vidas
        self.ojos=ojos
        self.manos=manos
        self.nave=nave
```

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Constructor>> Alien(nave:**NaveEspacial**,vidas,ojos,manos:entero)

Cuando se crea un objeto de clase `Alien` queda asociado a un objeto de clase `NaveEspacial`.

Tarea

establecerNave(nave:NaveEspacial)
Requiere nave ligada

recuperaVidas()
Incrementa en 1 las vidas hasta llegar a maxVidas

recibeHerida ()
Pierde 1 vida hasta llegar a 0

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Constructor>> Alien(nave:**NaveEspacial**,vidas,ojos,manos:entero) **<<Comandos>>** **establecerNave(nave:NaveEspacial)** **establecerOjos(ojos:entero)** **establecerManos(manos:entero)** **recuperaVidas()** **recibeHerida()** copy(a:Alien)

Tarea

obtenerFuerza():real
Requiere nave ligado. Retorna la capacidad visual, más su capacidad de lucha, todo multiplicado por el número de vidas por un quinto de la cantidad de combustible de la nave

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> **obtenerNave():NaveEspacial** **obtenerVidas():entero** **obtenerOjos():entero** **obtenerManos():entero** **obtenerFuerza():real** clone():Alien equals(alien: Alien): boolean

Igualdad superficial

```
def equals(self,alien):
    #requiere alien ligado
    return self.vidas==alien.obtenerVidas() and
           self.ojos==alien.obtenerOjos() and
           self.manos==alien.obtenerManos() and
           self.nave==alien.obtenerNave()
```

equals(alien: Alien): boolean
Requiere alien ligado. Dos aliens son equivalentes si tienen la misma cantidad de vidas, ojos y manos y están ligados a la misma nave

Dos objetos de clase `Alien` son **equivalentes** si tienen la misma cantidad de vidas, ojos y manos y están ligados a la misma nave

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien **equals(alien: Alien): boolean**

Igualdad superficial

```
def equals(self,alien):
    #requiere alien ligado
    return self.vidas==alien.obtenerVidas() and
           self.ojos==alien.obtenerOjos() and
           self.manos==alien.obtenerManos() and
           self.nave==alien.obtenerNave()
```

equals(alien: Alien): boolean
Requiere alien ligado. Dos aliens son equivalentes si tienen la misma cantidad de vidas, ojos y manos y están ligados a la misma nave

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien **equals(alien: Alien): boolean**

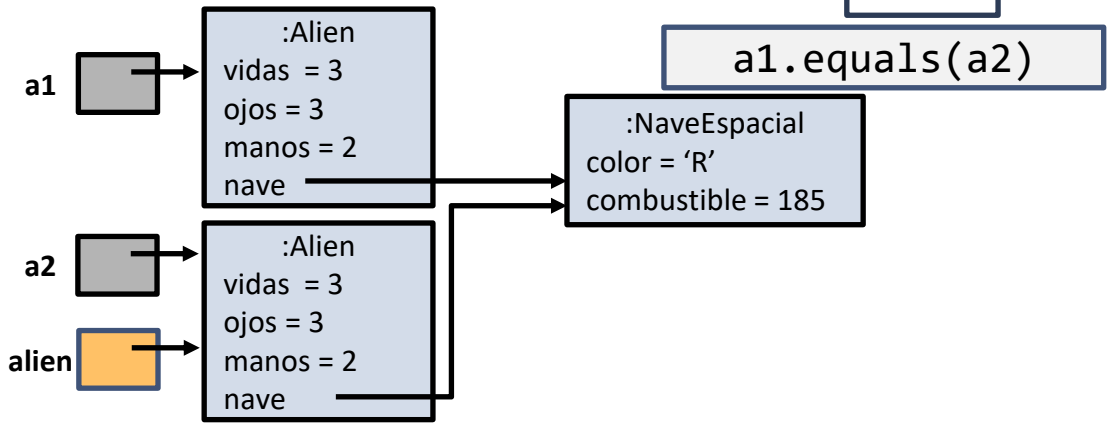
En la **igualdad superficial** se compara la **identidad** de los objetos

Igualdad superficial

```
def equals(self,alien):
    #requiere alien ligado
    return self.vidas==alien.obtenerVidas() and
           self.ojos==alien.obtenerOjos() and
           self.manos==alien.obtenerManos() and
           self.nave==alien.obtenerNave()
```

True

a1.equals(a2)



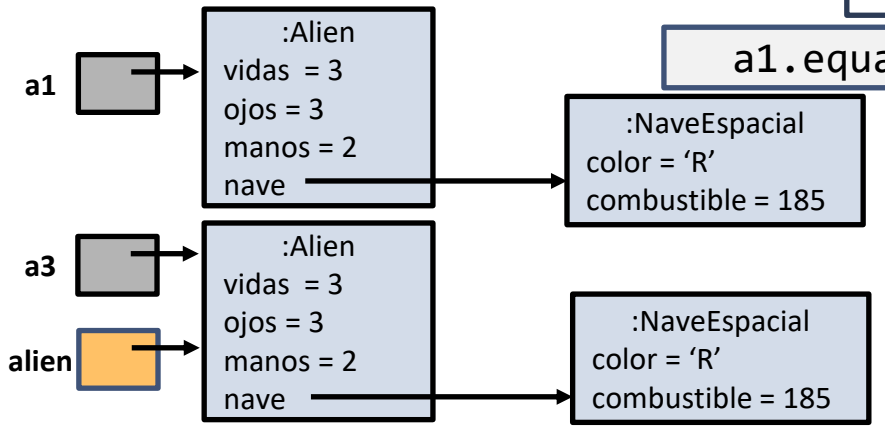
Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien **equals(alien: Alien): boolean**

Igualdad superficial

```
def equals(self,alien):
    #requiere alien ligado
    return self.vidas==alien.obtenerVidas() and
           self.ojos==alien.obtenerOjos() and
           self.manos==alien.obtenerManos() and
           self.nave==alien.obtenerNave()
```

False

a1.equals(a3)

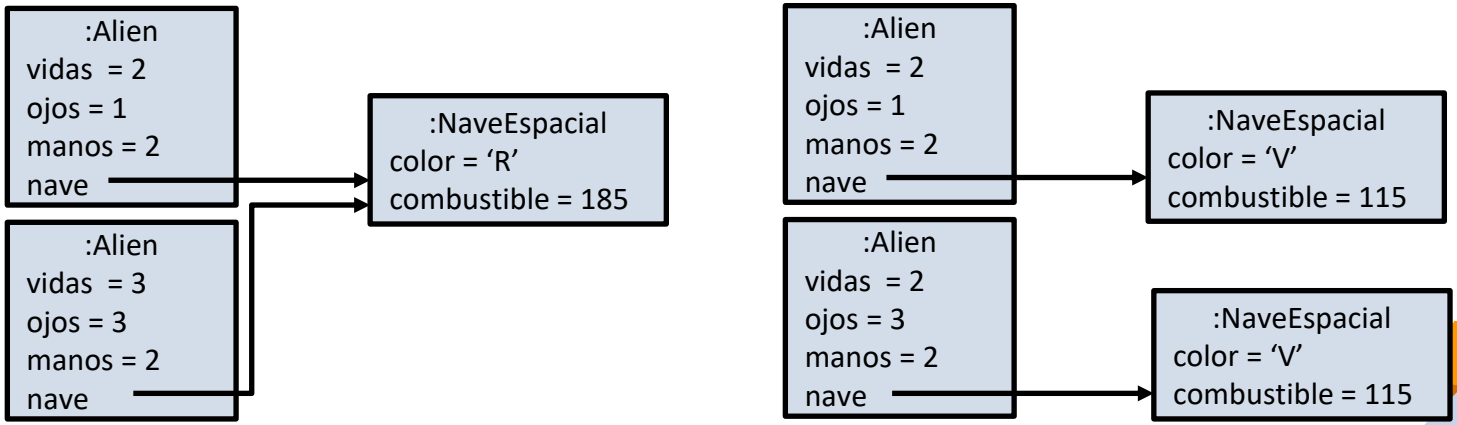


Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien **equals(alien: Alien): boolean**

Referencias y estado interno

Cada **alien** está representado en ejecución por un **objeto de software** cuyo estado interno mantiene los valores de los atributos que lo caracterizan.

Cada **nave espacial** también está representada por un único **objeto de software**, independientemente de cuántos aliens están asociados a esa nave.



Igualdad en profundidad

```
def equals(self,alien):
    #requiere alien ligado
    return self.vidas==alien.obtenerVidas() and
           self.ojos==alien.obtenerOjos() and
           self.manos==alien.obtenerManos() and
           self.nave.equals(alien.obtenerNave())
```

equals(alien: Alien): boolean
Requiere alien ligado. Dos aliens son **equivalentes** si tienen la misma cantidad de vidas, ojos y manos y sus naves tienen el mismo estado interno

Dos objetos de clase `Alien` son equivalentes si tienen la misma cantidad de vidas, ojos y manos y están ligados a naves equivalentes

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien **equals(alien: Alien): boolean**

Igualdad en profundidad

```
def equals(self,alien):
    #requiere alien ligado
    return self.vidas==alien.obtenerVidas() and
           self.ojos==alien.obtenerOjos() and
           self.manos==alien.obtenerManos() and
           self.nave.equals(alien.obtenerNave())
```

equals(alien: Alien): boolean
Requiere a ligado. Dos aliens son equivalentes si tienen la misma cantidad de vidas, ojos y manos y sus naves tienen el mismo estado interno

Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien **equals(alien: Alien): boolean**

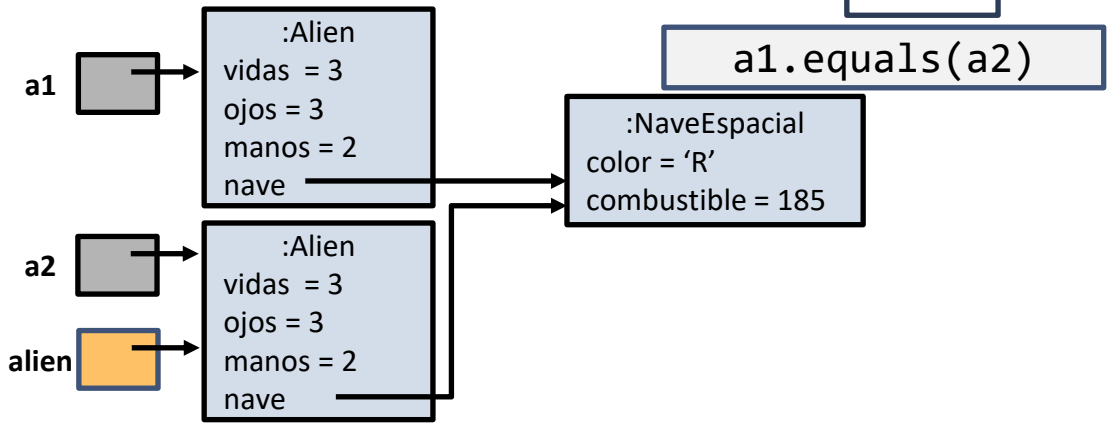
En la **igualdad en profundidad** se compara el **estado interno** de los objetos

Igualdad en profundidad

```
def equals(self,alien):
    #requiere alien ligado
    return self.vidas==alien.obtenerVidas() and
           self.ojos==alien.obtenerOjos() and
           self.manos==alien.obtenerManos() and
           self.nave.equals(alien.obtenerNave())
```

True

a1.equals(a2)



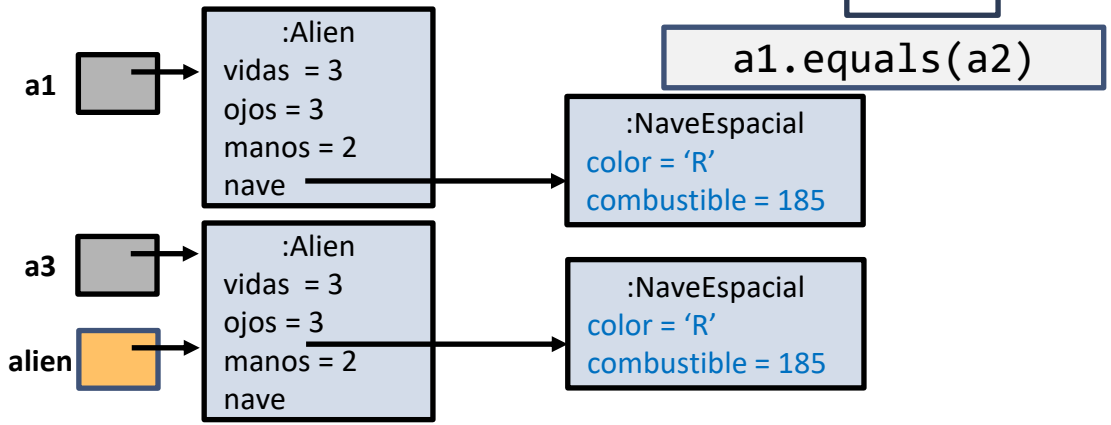
Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien **equals(alien: Alien): boolean**

Igualdad en profundidad

```
def equals(self,alien):
    #requiere alien ligado
    return self.vidas==alien.obtenerVidas() and
           self.ojos==alien.obtenerOjos() and
           self.manos==alien.obtenerManos() and
           self.nave.equals(alien.obtenerNave())
```

True

a1.equals(a2)



Alien
<<Atributo de clase>> maxVidas:entero **<<Atributo de instancia>>** vidas: entero ojos: entero manos: entero nave: **NaveEspacial**
<<Consultas>> obtenerNave():NaveEspacial obtenerVidas():entero obtenerOjos():entero obtenerManos():entero obtenerFuerza():real clone():Alien **equals(alien: Alien): boolean**

Igualdad en profundidad

Alien

```
def equals(self,alien):
    #requiere alien ligado
    return self.vidas==alien.obtenerVidas() and
           self.ojos==alien.obtenerOjos() and
           self.manos==alien.obtenerManos() and
           self.nave.equals(alien.obtenerNave())
```

NaveEspacial

```
def equals(self,nav):
    #requiere nav ligado
    return self.color==nav.obtenerColor() and
           self.combustble==nav.Combustible() and
```

```
a1.equals(a3)
```

La clase `Alien` es cliente de la clase `NaveEspacial`

Igualdad superficial e Igualdad en profundidad

En la **igualdad superficial** se compara la **identidad** del objeto de la clase asociada.

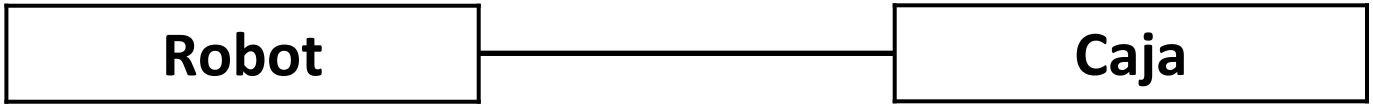
En la **igualdad en profundidad** se compara el **estado interno** del objeto de la clase asociada.

Clases Asociadas

- En la programación orientada a objetos el punto de partida para la construcción de un sistema es un proceso de abstracción y clasificación.
- Los **objetos de una clase** se caracterizan por los mismos atributos y comportamiento, pero además **comparten entre sí el mismo modo de relacionarse con objetos de otras clases**.
- Un objeto está **asociado** a otro objeto, si **tiene un** atributo de su clase.
- La relación entre los objetos provoca una relación entre las clases, que se dicen **asociadas**.

Dependencia entre clases

DEPENDENCIA



USA UN

Existe una **relación de dependencia** cuando un servicio de una clase declara una **variable local**, recibe un **parámetro** o **retorna como resultado** un **objeto de la clase dependiente**

Dependencia entre clases

En una fábrica de autos de juguete una parte de la producción la realizan **robots**. Cada robot tiene una carga de energía que se va consumiendo a medida que ejecuta las órdenes que recibe. Cada robot es capaz de conectarse de modo tal que se recargue su energía hasta su capacidad máxima de 5000 unidades. Esta acción puede ejecutarse ante una orden externa o puede iniciarla el robot mismo cuando su energía está por debajo de las 100 unidades. Cada robot cuenta con piezas de diferentes tipos: ruedas, ópticas y chasis. La cantidad de piezas se incrementa cuando un robot recibe una orden de abrir una **caja de piezas** y se decrementa cuando arma un vehículo. Cada caja tiene piezas de todos los tipos. Desarmar una caja cualquiera demanda 50 unidades de energía. Armar un auto consume 70 unidades de energía, 4 ruedas, 6 ópticas y 1 chasis. Es responsabilidad de cada servicio que consuma energía recargarla cuando corresponda.

Dependencia entre clases

Robot
<<Atributos de clase>> energiaMaxima : 5000 energiaMinima : 100 **<<Atributos de instancia>>** nroSerie:entero energia: entero ruedas: entero opticas: entero chasis: entero
<<Constructor>> Robot(nro:entero,caja:Caja) **<<Comandos>>** abrirCaja (caja: Caja) recargar() armarAuto()

Caja
<<Atributos de instancia>> ruedas: entero opticas: entero chasis : entero
<<Constructor>> Caja (rue,opt,chas: entero) **<<Comandos>>** establecerRuedas(n:entero) establecerOpticas(n:entero) establecerChasis(n:entero) vaciar() **<<Consultas>>** obtenerChasis () : entero obtenerRuedas () : entero obtenerOpticas () : entero equals(c:Caja):boolean

Dependencia entre clases

Robot
<<Atributos de clase>> energiaMaxima : 5000 energiaMinima : 100 **<<Atributos de instancia>>** nroSerie:entero energia: entero ruedas: entero opticas: entero chasis: entero
<<Constructor>> Robot(nro:entero,caja:Caja) **<<Comandos>>** abrirCaja (caja: Caja) recargar() armarAuto()

recargar()
recarga la energía del robot hasta llegar al máximo.

abrirCaja(caja:Caja)
aumenta las piezas disponibles de acuerdo a las cantidades de la caja y la vacía. Requiere caja ligada.

armarAuto()
decrementa las piezas disponibles, requiere que se haya controlado si hay piezas disponibles antes de enviar el mensaje armarAuto a un robot.

Dependencia entre clases

Robot
<<Atributos de clase>> energiaMaxima : 5000 energiaMinima : 100 **<<Atributos de instancia>>** nroSerie:entero energia: entero ruedas: entero opticas: entero chasis: entero
<<Consultas>> obtenerNroSerie():entero obtenerEnergia (): entero obtenerChasis () : entero obtenerRuedas () : entero obtenerOpticas () : entero cantAutos() : entero clone():Robot equals(Robot r):Boolean

cantAutos():entero
retorna la cantidad de autos que puede armar el robot con las piezas que tiene disponibles, sin desarmar una caja.

Dependencia entre clases

Robot
<<Atributos de clase>> energiaMaxima : 5000 energiaMinima : 100 **<<Atributos de instancia>>** nroSerie:entero energia: entero ruedas: entero opticas: entero chasis: entero
<<Responsabilidades>> El constructor establece la energía en el valor máximo y las cantidades de piezas en 100. Todos los servicios que consumen energía deciden recargar cuando energía es menor que la mínima. Requiere que haya piezas disponibles para armar un auto.

Dependencia entre clases

```
class Robot():
      #Atributos de clase
      energiaMaxima=5000
      energiaMinima=100
      def __init__(self,nro,caja):
          # requiere caja ligada
          self.nroSerie = nro
          self.energia = self.energiaMaxima
          self.ruedas = caja.obtenerRuedas()
          self.opticas = caja.obtenerOpticas()
          self.chasis = caja.obtenerChasis()
          caja.vaciar()
```

Robot
<<Atributos de clase>> energiaMaxima : 5000 energiaMinima : 100 **<<Atributos de instancia>>** nroSerie:entero energia: entero ruedas: entero opticas: entero chasis: entero
<<Constructor>> Robot(nro:entero,caja:Caja) **<<Comandos>>** abrirCaja (caja: Caja) recargar() armarAuto()

Dependencia entre clases

```
#Comandos
def recargar(self):
    self.energia = self.energiaMaxima
def armarAuto(self):
     '''Requiere que se haya controlado si hay
     piezas disponibles'''
     self.ruedas -= 4
     self.opticas -= 6
     self.energia -= 70
     self.chasis -= 1
     #Controla si es necesario recargar energía
     if self.energia < self.energiaMinima:
        self.recargar()
```

Robot
<<Atributos de clase>> energiaMaxima : 5000 energiaMinima : 100 **<<Atributos de instancia>>** nroSerie:entero energia: entero ruedas: entero opticas: entero chasis: entero
<<Constructor>> Robot(nro:entero,caja:Caja) **<<Comandos>>** abrirCaja (caja: Caja) **recargar()** **armarAuto()**

Dependencia entre clases

```
def abrirCaja(self,caja):
    ‘‘‘Aumenta sus cantidades según las de la caja
    y la vacía. Requiere
    caja ligada’’’
    self.ruedas += caja.obtenerRuedas()
    self.opticas += caja.obtenerOpticas()
    self.chasis += caja.obtenerChasis()
    self.energia -= 50
    caja.vaciar()
    #Controla si es necesario recargar energía
    if self.energia < self.energiaMinima:
       self.recargar()
```

Robot
<<Atributos de clase>> energiaMaxima : 5000 energiaMinima : 100 **<<Atributos de instancia>>** nroSerie:entero energia: entero ruedas: entero opticas: entero chasis: entero
<<Constructor>> Robot(nro:entero,caja:Caja) **<<Comandos>>** **abrirCaja (caja: Caja)** recargar() armarAuto()

Dependencia entre clases

```
#Consultas
def obtenerRuedas(self):
    return self.ruedas
def obtenerOpticas(self):
    return self.opticas
def obtenerChasis(self):
    return self.chasis
def obtenerNroSerie(self):
    return self.nroSerie
def obtenerEnergia(self):
    return self.energia
```

Robot
<<Atributos de clase>> energiaMaxima : 5000 energiaMinima : 100 **<<Atributos de instancia>>** nroSerie:entero energia: entero ruedas: entero opticas: entero chasis: entero
<<Consultas>> obtenerNroSerie():entero obtenerEnergia (): entero obtenerChasis () : entero obtenerRuedas () : entero obtenerOpticas () : entero cantAutos() : entero clone():Robot equals(Robot r):Boolean

Dependencia entre clases

```
class FabricaJuguetes():
      …
      def producir(self):
         caja = Caja(100,100,100)
         unRobot = Robot(111,caja)
         …
         print(caja.obtenerRuedas())
         …
         if unRobot.cantAutos() == 0:
            unRobot.abrirCaja(caja)
            unRobot.armarAuto()
         …
```

Tarea

Robot
<<Atributos de clase>> energiaMaxima : 5000 energiaMinima : 100 **<<Atributos de instancia>>** nroSerie:entero energia: entero ruedas: entero opticas: entero chasis: entero
<<Consultas>> obtenerEnergia (): entero obtenerChasis () : entero obtenerRuedas () : entero obtenerOpticas () : entero cantAutos() : entero clone():Robot equals(Robot r):boolean

Caja
<<Atributos de instancia>> ruedas: entero opticas: entero chasis : entero
<<Constructor>> Caja (rue,opt,chas: entero) **<<Comandos>>** establecerRuedas(n:entero) establecerOpticas(n:entero) establecerChasis(n:entero) vaciar() **<<Consultas>>** obtenerChasis () : entero obtenerRuedas () : entero obtenerOpticas () : entero equals(c:Caja):boolean

Ejercicio: Implemente la clase `Caja` y complete la clase `Robot` de acuerdo al diseño.

Relación entre clases

La **dependencia** es una forma de relación entre clases y se produce cuando los servicios que brinda un objeto **usan** a un objeto de otra clase.

La **asociación** es una forma de relación entre clases y se produce cuando un objeto de una clase **tiene** como atributo a un objeto de otra clase.

La interfaz de una clase

- Una clase que **usa** los servicios provistos por otra clase, es cliente de la clase que provee dichos servicios.
- Una clase que **tiene** atributos de otras clases, es cliente de las clases a las que corresponden esos atributos; estas ultimas son proveedoras de servicios.
- La clase cliente accede a la clase proveedora a través de su **interfaz.**
- La programación orientada a objetos propone **minimizar** la interfaz a través de la cual se comunican una clase cliente y una clase proveedora, de modo que se minimice también el impacto de los cambios.

La interfaz de una clase está formada por la signatura de los servicios públicos.

La signatura incluye nombre del servicio y el número de sus parámetros.

El contrato

- Entre una clase cliente y una clase proveedora de servicios, se establece un **contrato** que determina el **compromiso** que asume cada una.
- Las condiciones del contrato se especifican en la etapa de diseño del sistema e incluyen la **funcionalidad** y las **responsabilidades** inherentes a cada servicio.
- En la **implementación** es necesario interpretar el contrato para reflejar en el código las funcionalidades y responsabilidades especificas.

Bibliografia básica

- Óscar Ramírez Jiménez: ***"Python a fondo".*** 1era Edición. Ed. Marcombo S.L.. 2021.
- Allen Downey. ***"Think Python"***. 2Da Edición. Green Tea Press. 2015.
- Eirc Matthes: ***"Python Crash Course"***. 1era Edición. Ed. No Starch Press. 2016.
- Allen Downey, Jeffrey Elkner y Chris Meyers, ***"Aprenda a Pensar Como un Programador con Python"***. Bostton, USA, 2002.
- Peter C. Norton, Peter C. Norton: ***"Beginning Python"***. Ed. Wiley Publishing. 2005.
- Luis Joyanes Aguilar: **"Fundamentos de programación, algoritmos, estructura de datos y objetos".** Ed. Mc Graw Hill. 2008