Programación Orientada a Objetos

Clase 6 – Herencia

UTN - La Plata

Herencia

1) Jerarquías de clases
2) Mecanismo de herencia en la POO
3) Uso de las pseudo variables self y super
4) Actividad 6

1) Jerarquías de clases

En POO un programa puede concebirse como una red de objetos que interactúan enviándose mensajes.

En POO el código es más compacto, más flexible y requiere menos esfuerzo de modificación.

Cuando se diseña un sistema orientado a objetos se debe tener en cuenta las relaciones que existe entre las clases de objetos ya que existe una jerarquía de objetos y no todos las clases se encuentran en el mismo nivel. La jerarquía puede tener varios niveles.

En una jerarquía puede haber clases genéricas o abstractas y clases concretas.

De las clases abstractas no se pueden tener instancias, de las clases concretas sí.

Una clase es subclase de otra cuando especializa los conceptos definidos en ésta.

Las subclases generalizan a las subclases agrupando comportamiento común de las mismas.

1) Jerarquías de clases

Al programar se buscan definir conceptos abstractos y luego los subconceptos, subclases concretas.

Las jerarquías de clases en OO coinciden con jerarquías taxonómicas que existen agregando o clasificando seres de la naturaleza. Por ejemplo, en el mundo de los seres vivos, los animales se clasifican según tengan o no esqueleto y luego según otras características especiales.

Podemos decir que las distintas especies de animales, se relacionan formando una jerarquía del tipo “es-un” donde cada ejemplar de una subclasificación es también un ejemplar de su clase superior.

Un Vertebrado es un Animal

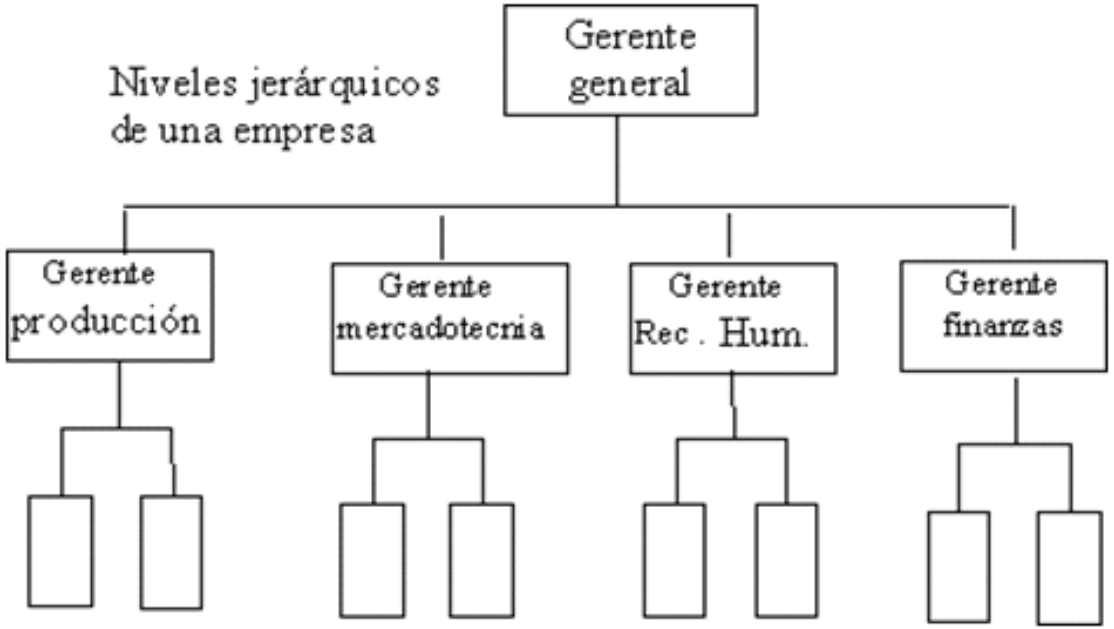
Un Pez es un Vertebrado

Un Pez es un Animal

1) Jerarquías de clases

En la vida cotidiana, existen otros tipos de jerarquías que no siguen esta forma taxonómica de agrupación, por ejemplo las jerarquías que se muestran en el organigrama de una Empresa.

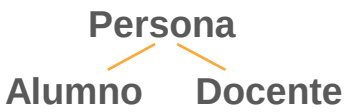
En este caso no se cumple la relación es-un. Por lo tanto no es una jerarquía en Objetos



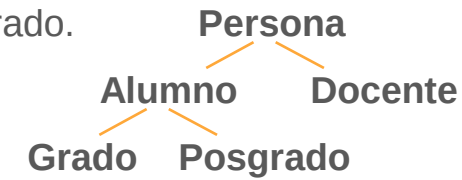
1) Jerarquías de clases

Ejemplo:

Si tenemos que realizar una aplicación para el manejo administrativo de la Universidad, aparecerán las clases Alumno y Docente que sirven para instanciar objetos alumnos y docentes. Son distintos porque van a responder mensajes distintos. Pero tanto alumno como docente tienen algunas características en común, por ser ambas personas. Ambos (alumnos y docentes) pueden ser subclases de Persona porque cumplen con la relación “es-un” con las clases superiores en la jerarquía.



Podemos decir que definir jerarquías de clases consiste en definir un árbol donde más abajo pongo las clases más concretas y más arriba las clases más abstractas. Además se pueden agregar más clases, por ejemplo especializando la clase alumno con Alumno de grado y de posgrado.



2) Mecanismo de Herencia en POO

Generalización: se agrupan clases creando superclases que tengan características comunes. Es decir, se abstraen características comunes a dos o más clases formando una clase más abstracta.

Especialización: se agrega un nuevo nivel a la jerarquía, es decir se agrega comportamiento a la jerarquía creando nuevas subclases.

Herencia: La herencia es el mecanismo mediante el cual se pueden definir nuevas clases basadas en otras ya existentes, a fin de reutilizar el código, generando así una jerarquía de clases dentro de una aplicación. Si una clase deriva de otra, esta hereda sus atributos y métodos y puede añadir nuevos atributos, métodos o redefinir los heredados.

2) Mecanismo de Herencia en POO

Hay dos tipos de herencia: de estructura y de comportamiento.

Herencia de estructura: en Smalltalk es total, las subclases heredan todos los atributos de la superclase. No hay forma de no heredar algún atributo.

Herencia de comportamiento: es parcial, se puede no heredar un método de una superclase, redefiniéndolo en la subclase.

Ventajas de la herencia:

1) Se escribe menos código.

2) Favorece el mantenimiento del programa.

Para programar usando herencia se utilizan las pseudo variables ***self*** *y* ***super***:

- self: hace referencia al objeto receptor del mensaje
- super:hace referencia a la superclase inmediata de la clase que contiene el método en el cual aparece dicha pseudovariable-

3) Ejemplos de uso de self y super

Dada la siguiente jerarquía:

Clase A

```
>>m1
    ^ 2.
 >>m2
    ^ 8.
 >>m3
    ^ self m6 + self m2.
>>m6
    ^ self m2.
```

Clase B

```
>>m1
    ^ 5.
 >>m4
      ^ self m2 + super m3.
>>m3
    ^ 3.
>>m6
    ^ self m2.
```

Clase C

```
>>m1
    ^ self m4.
 >>m2
    ^ 8.
>>m7
    ^ super m6.
```

Jerarquía

3) Ejemplos de uso de self y super

Resolver el siguiente ejercicio:

a) |unObjeto|

```
unObjeto:= C new.
unObjeto m7 inspect.
```

Qué resultado muestra????

b) |unObjeto|

```
unObjeto:= C new.
unObjeto m1 inspect.
```

Qué resultado muestra????

3) Ejemplos de uso de self y super

Resultados:

|unObjeto|
unObjeto:= C new.

a) unObjeto m7 inspect.

super m6

self m2

8 **Retorna 8**

|unObjeto|
unObjeto:= C new.

b) unObjeto m1 inspect.

self m4

self m2 + super m3

8 + (self m6 + self m2)

8 + (self m2 + self m2)

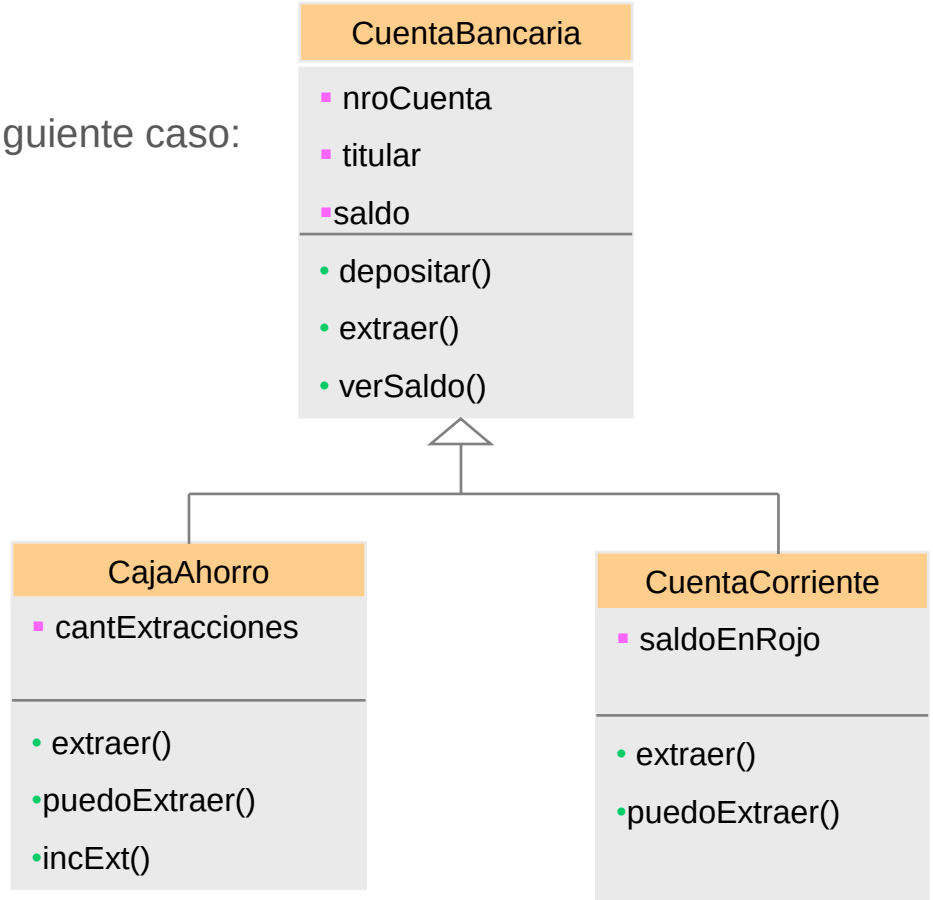
8 + (8 + self m2)

8 + 8 + 8 **Retorna 24**

3) Ejemplos de uso de self y super

Ejemplo:

Supongamos que tenemos el siguiente caso:



3) Ejemplos de uso de self y super

- Los métodos depositar y el verSaldo se los puede implementar en la superclase CuentaBancaria ya que las instancias de CajaDeAhorro y CuentaCorriente entienden ambos mensajes de la misma manera.

CuentaBancaria>>depositar:unMonto

saldo:=saldo + unMonto.

CuentaBancaria>>verSaldo

^ saldo.

En cambio la implementación del extraer es diferente porque las instancias de CajaDeAhorro y de CuentaCorriente no entienden el extraer de la misma manera.

3) Ejemplos de uso de self y super

- La implementación del extraer se puede implementar de dos maneras diferentes:

Opción 1)

a) Se implementa el método extraer en la clase CajaDeAhorro para que lo entiendan las instancias de CajaAhorro.

CajaAhorro>>extraer:unMonto

```
(self puedoExtraer:unMonto) ifTrue:[saldo:=saldo – unMonto.
                                    self incExt].
```

3) Ejemplos de uso de self y super

b) Se implementa el método extraer en la clase CuentaCorriente para que lo entiendan las instancias de CuentaCorriente.

CuentaCorriente>>extraer:unMonto

(self puedoExtraer:unMonto) ifTrue:[super extraer: unMonto].

c) En la clase CuentaBancaria se escribe el método pero no se implementa.

CuentaBancaria>>extraer:unMonto

^(self implemented by Subclass).

En este caso el extraer está implementado en las 3 lases.

3) Ejemplos de uso de self y super

Aplicación:

| ca cc|

ca:=CajaDeAhorro crear….

cc:=CuentaCorriente crear….

ca depositar: 1500.

…..

ca extraer: 500.

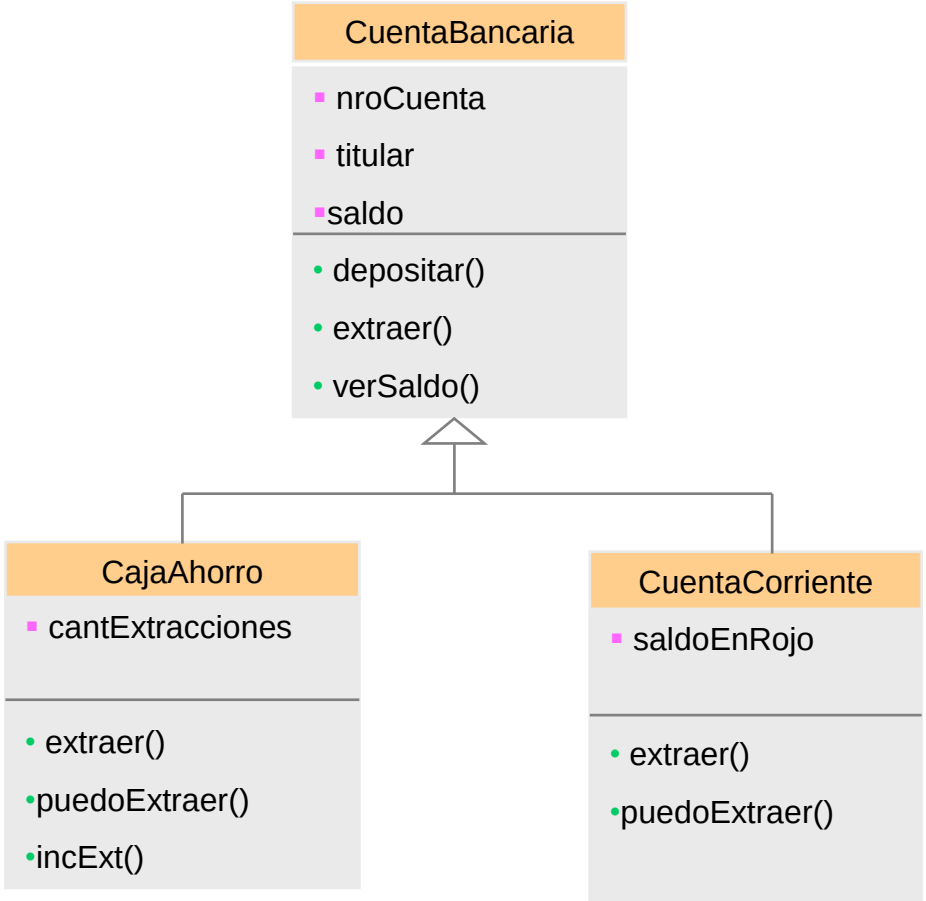
ca verSaldo inspect.

cc depositar: 3000.

…..

cc extraer: 1000.

cc verSaldo inspect.



3) Ejemplos de uso de self y super

Opción 2)

Se descompone el extraer en distintas operaciones, por ejemplo:

a) **verificarCuentaHabilitada** (verifCtaHab)

- En CajaDeAhorro significa que no supera la cantidad de extracciones permitidas por el Banco
- En CuentaCorriente significa que no emitió un cheque sin fondo

b) verificarOperaciónVálida (verifOpVálida)

- En CajaDeAhorro significa que el monto es menor o igual que el saldo
- En CuentaCorriente significa que el monto es menor o igual que el saldo más el saldo en rojo

c) realizarOperación (realizarOp)

- En CajaDeAhorro decrementa el saldo e incrementa la cantidad de extracciones
- En CuentaCorriente decrementa el saldo

3) Ejemplos de uso de self y super

Opción 2)

a) Existe un sólo extraer implementado en la clase CuentaBancaria. Este extraer lo entienden de distinta forma las instancias de CajaDeAhorro y de CuentaCorriente.

CuentaBancaria>>extraer:unMonto

```
^ ((self verifCtaHab) & (self verifOpVálida)) ifTrue:[self realizarOp:unMonto].
```

Volvemos a ver la aplicación y probamos…

3) Ejemplos de uso de self y super

Aplicación:

| ca cc|

ca:=CajaDeAhorro crear….

cc:=CuentaCorriente crear….

ca depositar: 1500.

…..

ca extraer: 500.

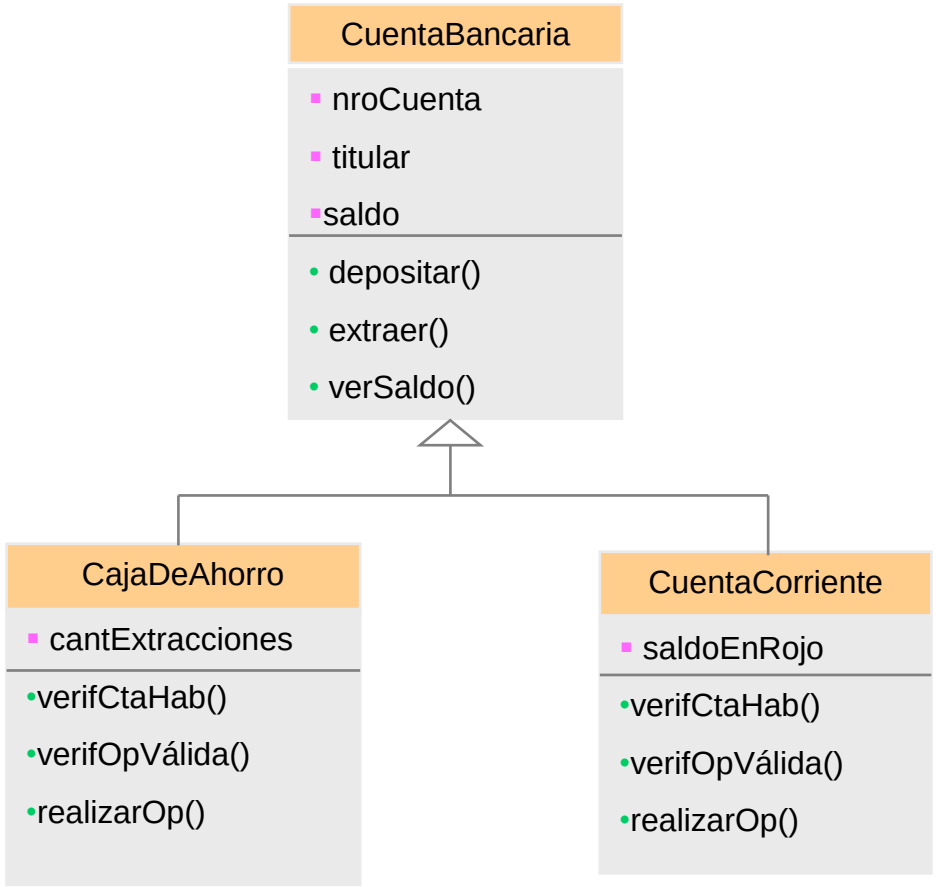
ca verSaldo inspect.

cc depositar: 3000.

…..

cc extraer: 1000.

cc verSaldo inspect.



5) Actividad 6

Dada la siguiente jerarquía:

Clase A

```
>>m1
    ^ 10.
 >>m2
    ^ 8.
>>m3
    ^ self m1 + self m2.
```

Clase B

```
>>m1
    ^ 9.
>>m2
    ^ self m1 + super m3.
>>m3
    ^ 15.
```

Clase C

```
>>m1
    ^ 5.
>>m2
    ^ 7.
>>m3
    ^ super m2+self m1.
```

Jerarquía

Qué retornán las siguientes líneas de código?

a) |unObjeto|

```
unObjeto:= C new.
unObjeto m3 inspect.
```

5) Actividad

Qué retornán las siguientes líneas de código?

a) |unObjeto|

```
unObjeto:= C new.
unObjeto m3 inspect.
```

```
super m2 + self m1
self m1 + super m3 + self m1
5 + self m1 + self m2 + 5
5 + 5 + 7 + 5  = 22
```