Herència

Tecnologia de la Programació

Sebastià Vila-Marta

Enginyeria de Sistemes TIC
Universitat Politècnica de Catalunya
http://epsem.upc.edu

2 de març de 2020

En el tema anterior...

- Les classes d'objectes (i per tant els objectes) poden tenir mètodes de nom especial que permeten sobrecarregar operadors.

L'operador «is»

Fixem-nos en el següent exemple:

Example

```
>>> t1 = (1,2)
>>> t2 = (1,2)
>>> t3 = t2
>>> t1 == t2
True
>>> t1 is t2
False
>>> t3 is t2
True
```

A l'exemple hi ha dues operacions subtilment diferents:

- La identitat, **is**.

  Dues instàncies són idèntiques si són exactament la mateixa.

- La igualtat, ==.

  Dues instàncies són iguals si representen la mateixa entitat en el domini del problema.

Foto !!

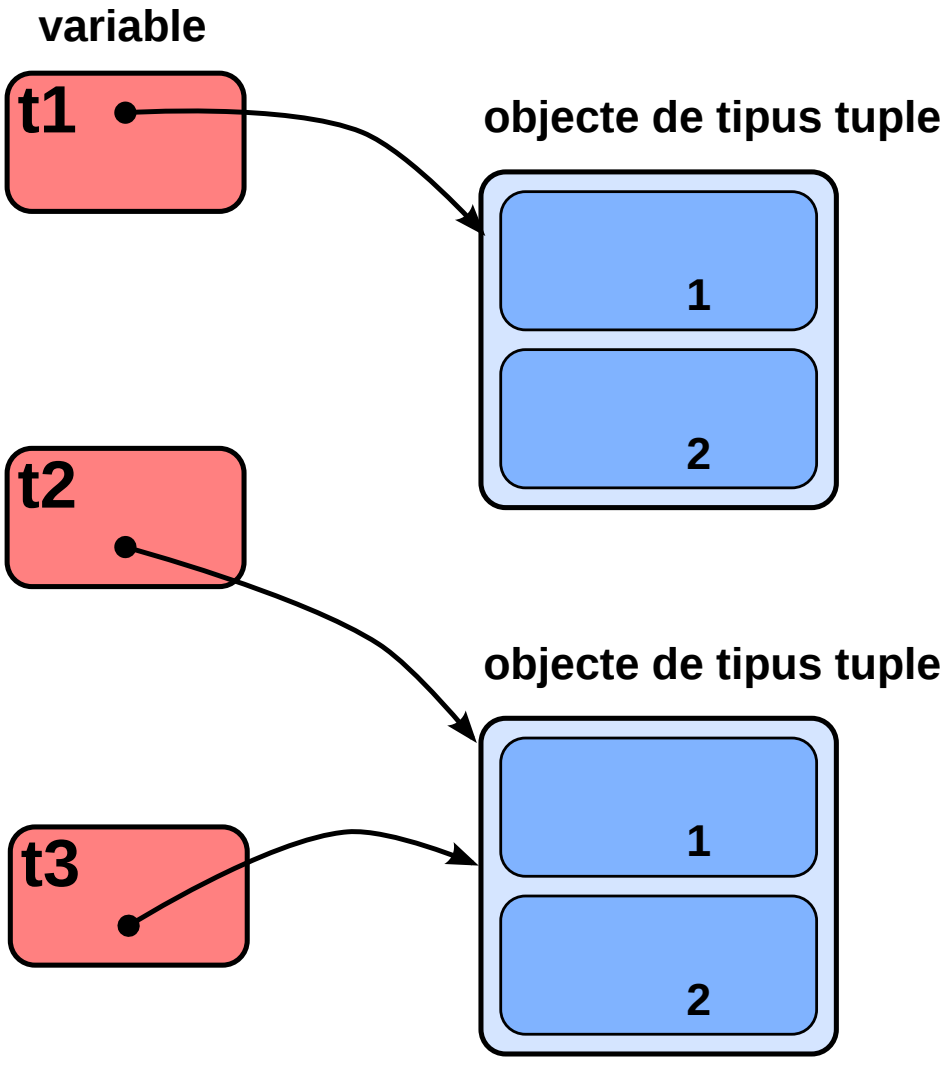

Figura: t1 i t2 són iguals, t2 i t3 idèntics

Classes, identitat i igualtat I

Situació

Estem dissenyant una aplicació de gestió acadèmica. Volem representar un estudiant amb una instància d'una classe. Un estudiant es caracteritza per tenir un dni, un nom, uns cognoms i un telèfon. Entenem que el dni és únic i, per tant, dues instàncies d'estudiant són iguals si tenen el mateix dni.

Si implementem una classe amb aquestes característiques obtenim:

```
class Estudiant(object):
    def __init__(self, dni, nom, cognoms, tel=0):
        self.dni = dni
        self.nom = nom
        self.cognoms = cognoms
        self.tel = tel
```

Classes, identitat i igualtat II

Fem unes proves sobre identitat:

```
>>> p = Estudiant(100, 'Pep', 'Ferran')
>>> n = Estudiant(200, 'Nuria', 'Llisach')
>>> j = Estudiant(100, 'Pep', 'Ferran')
>>> p is j
False
>>> p == j
False
```

Noteu que:

- p i j no són idèntics i això és correcte.
- p i j haurien de ser iguals ja que tenen el mateix dni i, segons la nostra definició d'estudiant, dos estudiants amb el mateix dni són iguals. Ara, però, es consideren diferents.

Classes, identitat i igualtat III

Si no es diu el contrari, la igualtat en classes definides per l'usuari està definida com la identitat. Si volem un concepte d'igualtat específica, cal recórrer als mètodes especials:

```
def __eq__(self, e):
    """ És self igual que 'e'? """
    return isinstance(e,Estudiant) and self.dni == e.dni

def __ne__(self, e):
    """ És self diferent d''e'? """
    return not isinstance(e,Estudiant) or self.dni != e.dni
```

Classes, identitat i igualtat IV

Si ara refem els experiments observarem un comportament com el que volíem:

```
>>> p1 = Estudiant(100, 'Pep', 'Ferran')
>>> n = Estudiant(200, 'Nuria', 'Llisach')
>>> p2 = Estudiant(100, 'Pep', 'Ferran')
>>> j = Estudiant(100, 'Josep', 'Ferran')
>>> j2 = Estudiant(300, 'Pep', 'Ferran')
>>> p1 is p2
False
>>> p1 == p2
True
>>> p1 == n
False
>>> p1 == j2
False
```

Classes, identitat i igualtat V

Per a definir la igualtat sobre les instàncies d'una classe cal:

- Definir dos mètodes: __eq__ i __ne__.
- Que es compleixi sempre aquest predicat:

$$\forall a, b : a \textbf{ is } b \implies a == b$$

Els estudiants són persones

A vegades, entre dues entitats del nostre problema existeix una relació del tipus «és un». Per exemple:

Carro	és un	Vehicle
Camió	és un	Vehicle
Estudiant	és un	Persona
Professor	és un	Persona

Que un carro i un cotxe siguin un vehicle té implicacions quan ho representem sobre classes:

- Tant carro com camió tindran atributs comuns. Per exemple, el número de matrícula, el cost per kilòmetre o la tara.
- Tant carro com camió tindran mètodes comuns. Per exemple, un mètode que retorna la tara per eix.
- Els mètodes i atributs comuns els tenen pel fet de ser vehicles.
- Els llenguatges orientats a objectes ofereixen un mecanisme per modelar aquesta situació: l'herència (*inheritance*) entre classes.

Herència entre classes

- Quan definim una classe B podem indicar que és una classe derivada o una subclasse de la classe A amb aquesta sintaxi:

```
class A(object):
    ...
class B(A):
    ...
```

- Que B sigui una subclasse d'A implica:
  - Que B hereta de forma automàtica els atributs d'A.
  - Que B hereta de forma automàtica els mètodes d'A.
- En cas que B sigui una subclasse d'A també direm que A és una *classe base* o una superclasse de B.

Persones, estudiants i professors I

Example

Representem les entitats persona, estudiant i professor en el context d'una institució acadèmica.

```
import datetime

class Persona(object):

    def __init__(self, dni, nom):
        self.dni = dni
        self.nom = nom

    def set_any_naixement(self, any):
        self.any_naixement = any

    def edat(self):
        today = datetime.date.today()
        delta = today.year - self.any_naixement
        return delta
```

```
class Estudiant(Persona):

    def __init__(self, dni, nom, cr_aprovats):
        super(Estudiant,self).__init__(dni, nom)
        self.ca = cr_aprovats

    def graduat(self):
        return self.ca >= 240
```

Persones, estudiants i professors II

Juguem una mica amb aquestes classes:

```
>>> p = Estudiant(100, 'pep', 200)
>>> p.set_any_naixement(1965)
>>> isinstance(p, Estudiant)
True
>>> p.edat()
46
>>> isinstance(p, Persona)
True
>>> p.graduat()
False
>>> q = Persona(200, 'manel')
>>> isinstance(q, Estudiant)
False
>>> q.set_any_naixement(1970)
>>> q.edat()
41
>>> q.graduat()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'Persona' object has no attribute 'graduat'
```

Persones, estudiants i professors III

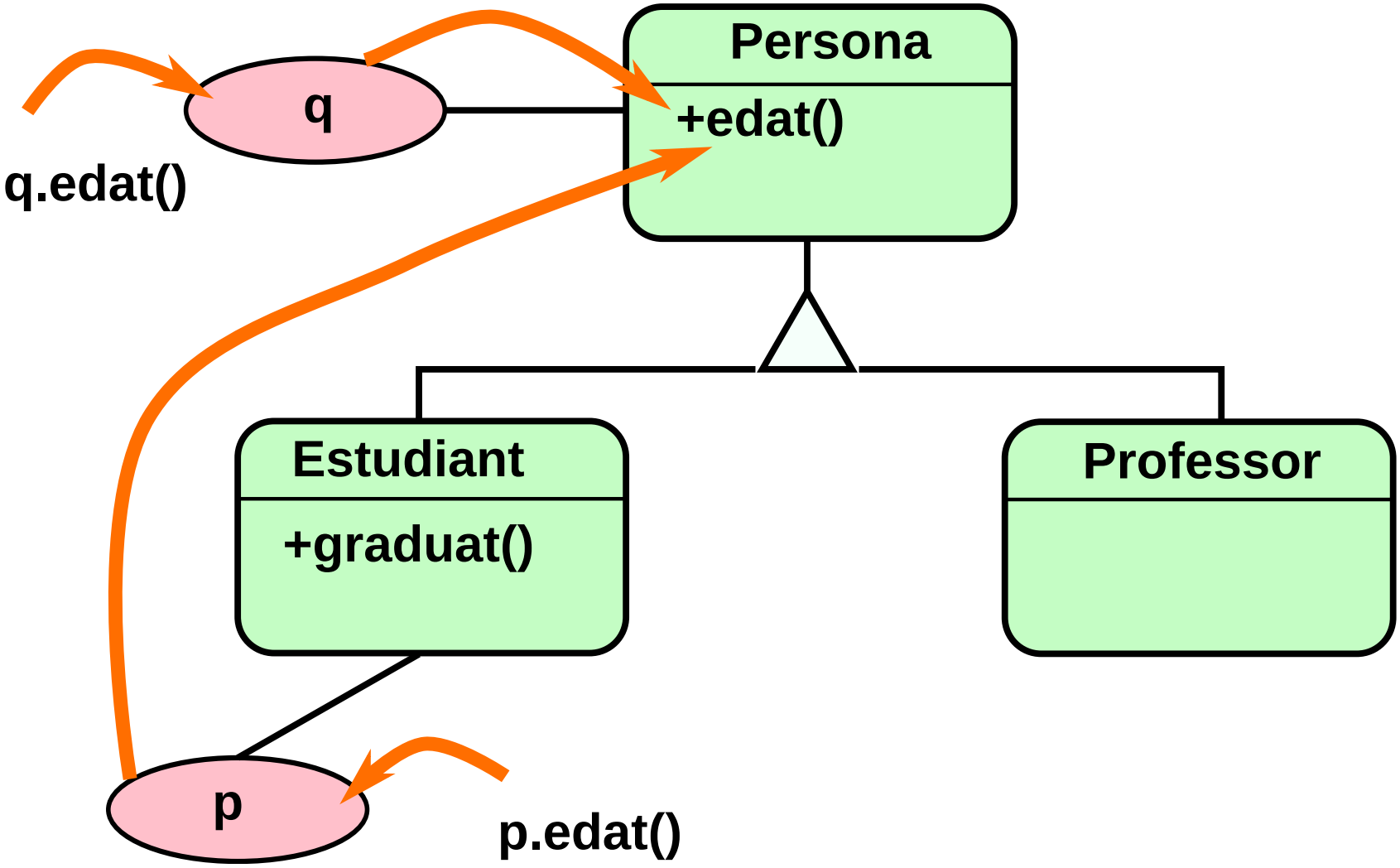

Figura: Efecte de l'herència sobre un Estudiant

Persones, estudiants i professors IV

Definim ara la classe Professor:

```
class Professor(Persona):

    def __init__(self, dni, nom, salari_anual=0):
        super(Professor,self).__init__(dni, nom)
        self.salari_anual = salari_anual

    def salari_mensual(self):
        return self.salari_anual / 14;
```

Si ara volguéssim dotar a les instàncies de Professor i Estudiant d'operació d'igualtat basada en el dni, allò més assenyat fóra afegir les operacions a la classe Persona atès que el dni és un atribut de la classe Persona.
Afegint els mètodes a la classe base, les subclasses els hereten automàticament. Una modificació, moltes classes afectades.

Persones, estudiants i professors V

```
import datetime

class Persona(object):

    def __init__(self, dni, nom):
        self.dni = dni
        self.nom = nom

    def set_any_naixement(self, any):
        self.any_naixement = any

    def edat(self):
        today = datetime.date.today()
        delta = today.year — self.any_naixement
        return delta

    def __eq__(self, p):
        return isinstance(p, Persona) and self.dni == p.dni

    def __ne__(self, p):
        return not isinstance(p, Persona) or self.dni != p.dni
```

Redefinició de mètodes I

Les subclasses poden redefinir els mètodes que hereten de les seves superclasses. Fixem-nos en les modificacions a les classes Persona i Professor indicades en el següent exemple:

```
class Persona(object):

    def __init__(self, dni, nom):
        self.dni = dni
        self.nom = nom

    def set_any_naixement(self, any):
        self.any_naixement = any

    ...

    def print_fitxa(self):
        print "Sra. {0}".format(self.nom)
        print "DNI {0}".format(self.dni)
```

```
class Professor(Persona):

    def __init__(self, dni, nom, salari_anual=0):
        super(Professor,self).__init__(dni, nom)
        self.salari_anual = salari_anual

    def salari_mensual(self):
        return self.salari_anual / 14;

    def print_fitxa(self):
        print "Prof. {0}".format(self.nom)
        print "DNI {0}".format(self.dni)
        print "Salari: {0}".format(self.salari)
```

Redefinició de mètodes II

Si ara juguem una mica amb la nova versió de les classes observarem el següent:

```
>>> per = Persona(100, "Marta")
>>> pro = Professor(200, "Jaume", 3000)
>>> est = Estudiant(300, "Peronella")
>>> per.print_fitxa()
Sra. Marta
DNI 100
```

```
>>> pro.print_fitxa()
Prof. Jaume
DNI 200
Salari: 3000
>>> est.print_fitxa()
Sra. Peronella
DNI 300
```

Redefinició de mètodes III

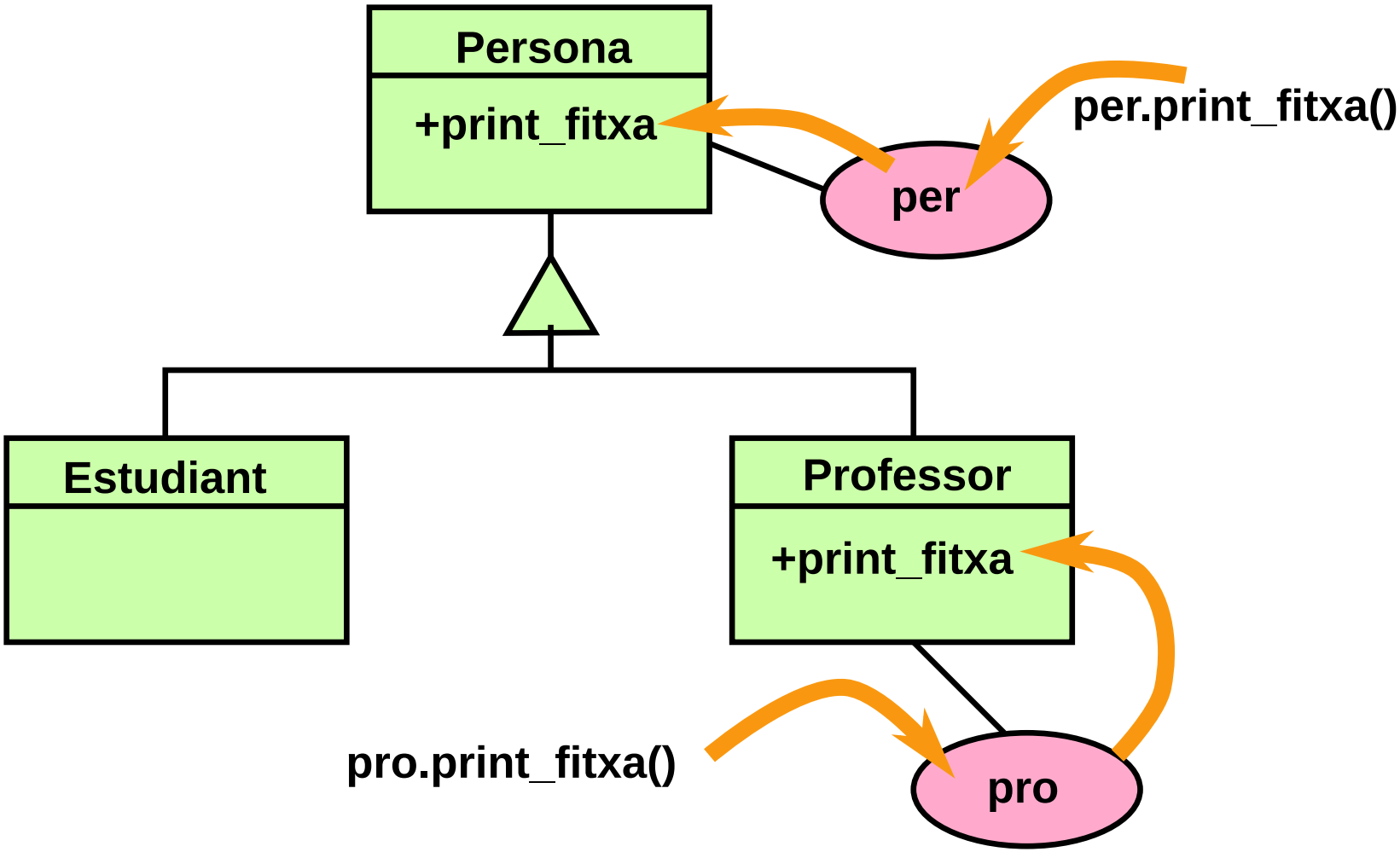

Figura: Efecte de la redefinició de mètodes

Redefinició de mètodes IV

Observacions

- Invocar print_fitxa sobre un Estudiant executa el mètode definit a la classe Persona. Actua l'herència de mètodes.
- Invocar print_fitxa sobre un Professor executa el mètode definit a la classe Professor. Actua la redefinició d'aquest mètode.
- Redefinir un mètode a una subclasse requereix definir el mètode amb idèntica signatura que a la superclasse.

Redefinició de mètodes V

També podríem haver redefinit el mètode print_fitxa a la classe Professor complementant el mètode de la seva superclasse. El resultat fóra:

```
class Professor(Persona):

def __init__(self, dni, nom, salari_anual=0):
    super(Professor,self).__init__(dni, nom)
    self.salari_anual = salari_anual

def salari_mensual(self):
    return self.salari_anual / 14;

def print_fitxa(self):
    super(Professor,self).print_fitxa()
    print "Salari: {0}".format(self.salari)
```

Una i altra implementacions són funcionalment equivalents. Aquesta darrera, però, és més senzilla de mantenir. Per que?

Delegació I

Observem el següent exemple:

```
class Animal(object):
    def parla(self):
        so = self.get_so()
        print so.capitalize()
```

```
class Gat(Animal):
    def get_so(self):
        return "meeeuuuu!!"
```

```
class Gos(Animal):
    def get_so(self):
        return "bup! bup!"
```

Juguem una mica amb l'exemple:

```
>>> x = Gat()
>>> x.parla()
Meeeuuuu!!
>>> y = Gos()
>>> y.parla()
Bup! bup!
>>> z = Animal()
>>> z.parla()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in parla
AttributeError: 'Animal' object has no attribute 'get_so'
```

Delegació II

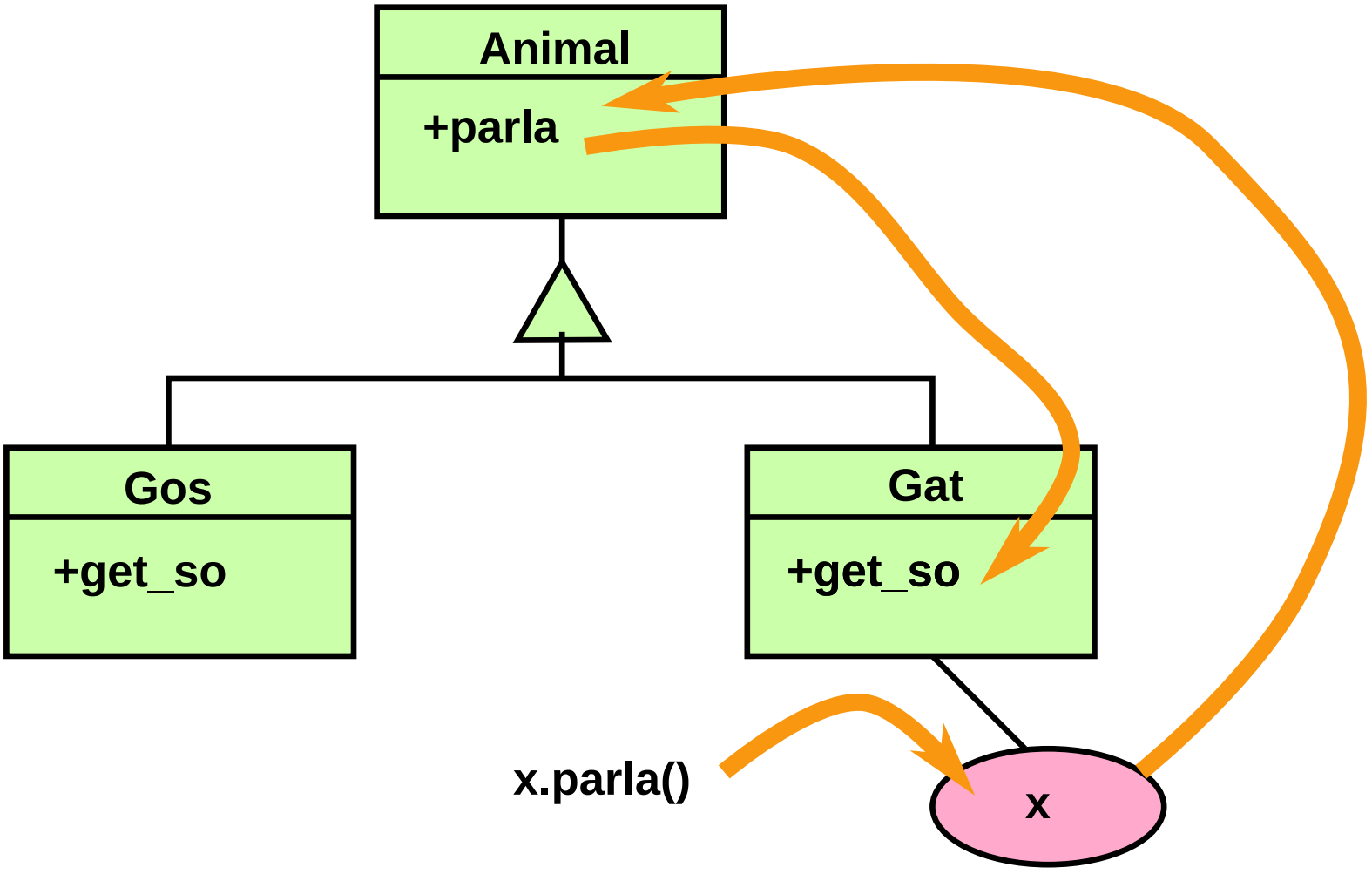

Figura: Efecte de la delegació.

Delegació III

Observacions

- En la delegació, un mètode d'una superclasse utilitza mètodes de les subclasses.
- Diem que el mètode de la superclasse delega part de responsabilitats a la subclasse.

Feina per aquesta setmana

1. Estudi de la teoria. A partir de les transparències i les notes que heu pres. Inclou provar els conceptes en el computador.
2. Incrementar el xuletari i el glossari del tema.
3. Solució dels problemes del tema.
4. Solució del problema especial, que té com objectiu aconseguir la solució més senzilla i entenedora possible.