

# Information hiding

## Tecnologia de la Programació

Sebastià Vila-Marta

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

16 de febrer de 2020

- 1 En el tema anterior...
- 2 Classes i instàncies
- 3 La classe Wallet
- 4 Diagrames de classe UML
- 5 Mòduls, classes i programes
- 6 Information hiding
- 7 Paràmetres: valors per omisió
- 8 Per a la setmana entrant...

- L'estructura **class** per definir nous tipus de dades.
- A les instàncies les hi podem afegir atributs.
- En una classe podem definir mètodes. Tots els **objectes instància** en tindran una còpia.
- Hi ha mètodes constructors, consultors i modificadors.
- Els doctests s'apliquen també a les classes.

## Pregunta

Quina és la relació entre classes i instàncies?

- En el paradigma d'orientació a objectes, el centre és l'objecte instància o instància.
- Per no haver de definir els objectes un a un s'introdueix el concepte de classe.
- Conceptualment, una classe d'objectes és el conjunt de tots els objectes instància que tenen els mateixos mètodes i atributs.
- Quan definim una classe, definim el «patró comú» de totes les instàncies d'aquella classe.
- Quan diem que una instància és d'una classe, estem «explicant» quina és l'estructura de la instància.

Anem a modelar una entitat senzilla: un moneder. Un moneder és:

- Un contenidor de diversos tipus de moneda.
- Per simplicitat, assumirem que únicament pot contenir monedes de 1,2 i 5.

## Representació

Representarem la informació d'un moneder com un diccionari en que:

**clau** El valor facial de la moneda.

**valor** La quantitat que en conté.

## Implementació

```
class Wallet(object):  
    def __init__(self, q1, q2, q5):  
        self.mon = {1:q1, 2:q2, 5:q5}  
  
    def value(self):  
        return sum([k*v for k,v in self.mon.items()])
```

Noteu que:

- Els atributs poden ser de qualsevol tipus, per exemple diccionaris.

És molt convenient afegir operacions per manipular les monedes del moneder. Per exemple, una operació per afegir/treure monedes del moneder:

## Implementació

```
def add_coins(self, v, q):  
    self.mon[v] += q
```

També seria interessant un operació per «sumar» moneders. Aquesta operació hauria de modificar un moneder incorporant-hi les monedes d'un segon moneder.

## Implementació

```
def add(self,m):  
    for v in self.mon:  
        self.mon[v] += m.mon[v]
```

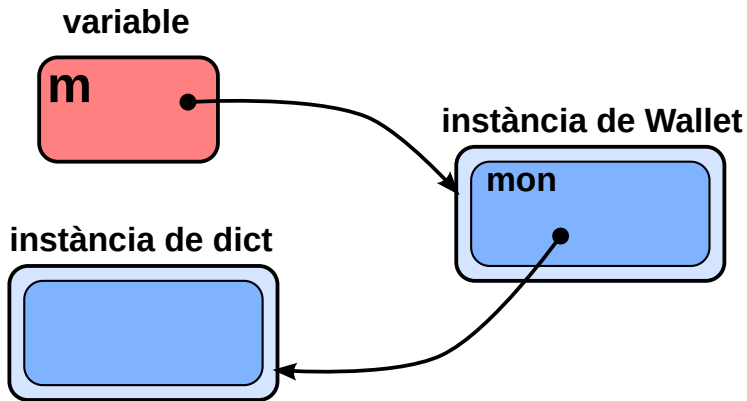
Fixeu-vos:

- `add` és un mètode modificador.
- `self` indica la instància de `Wallet` a la que s'aplica el mètode i `m` la instància de `Wallet` que se suma.
- S'accedeix al diccionari de la instància `m` usant `m.mon`.
- Els diccionaris de Python també són instàncies d'objecte, en aquest cas de la classe `dict`. Per aquesta raó, sovint usem els seus mètodes:

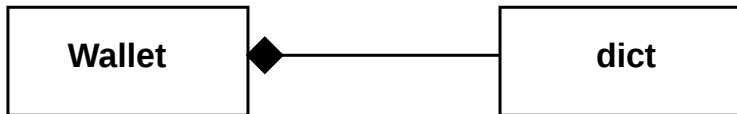
```
d = {1: 'a', 2: 'b'}  
print a.keys()
```

- Així, un `Wallet` té un atribut que és una instància d'una altra classe.





- Podem representar el que hem vist a la «foto» amb un diagrama de classes.
- Usarem la notació **UML Class Diagram** (Unified Modeling Language).
- La relació entre wallet's i dict's és de **composició**. Un wallet està compost d'un dict.
- La composició es representa amb un línia acabada en un diamant negre pel costat de la classe que agrega.
- En una composició, el **temps de vida** de la **instància contenidora** condiona el temps de vida de la **instància continguda**.



Un programa que treballa amb moneders:

## Example

```
m = Wallet(5,5,5)
m.add_coins(1,4)
m.add_coins(2,-1)
m.add_coins(5,1)
q = Wallet(1,2,3)
m.add(q)
print m.value()
```

Què escriu aquest programa?

# Com s'organitza un programa amb classes? I

Hi ha diversitat d'opcions. Una opció habitual és emprar un mòdul per cada classe:

## wallet.py

```
class Wallet(object):
    def __init__(self, q1, q2, q5):
        self.mon = {1:q1, 2:q2, 5:q5}

    def value(self):
        return sum([k*v for k,v
                    in self.mon.items()])

    def add_coins(self, v, q):
        self.mon[v] += q

    def add(self,m):
        for v in self.mon:
            self.mon[v] += m.mon[v]
```

## main.py

```
from wallet import Wallet

if __name__ == "__main__":
    m = Wallet(5,5,5)
    m.add_coins(1,4)
    m.add_coins(2,-1)
    m.add_coins(5,1)
    q = Wallet(1,2,3)
    m.add(q)
    print m.value()
```

Noteu que:

- Només cal importar el símbol Wallet.
- Anomenem al mòdul que conté la classe de forma similar a la classe.
- El programa principal és molt elegant!!

# Comparem dos programes

Aquí tenim dos programes **funcionalment equivalents**:

- Quines són les diferències?
- Quines implicacions tenen?
- Són importants?

mainA.py

```
from wallet import Wallet

if __name__ == "__main__":
    m = Wallet(5,5,5)
    m.add_coins(1,4)
    m.add_coins(2,-1)
    m.add_coins(5,1)
    q = Wallet(1,2,3)
    m.add(q)
    print m.value()
```

mainB.py

```
from wallet import Wallet

if __name__ == "__main__":
    m = Wallet(5,5,5)
    m.mon[1] += 4
    m.mon[2] += -1
    m.add_coins(5,1)
    q = Wallet(1,2,3)
    m.add(q)
    print m.value()
```

Per entendre les diferències, cal situar-se en el context apropiat:

- Cal imaginar-se que un i altra programa són de mides més reals, diguem que uns milers de línies de codi.
- Cal imaginar-se que no són aplicacions «acadèmiques».

## Pregunta

En aquesta situació, què passaria si volguéssim canviar la forma com s'emmagatzema la informació d'un Wallet i usar una llista en comptes d'un diccionari?

## wallet.py

```
class Wallet(object):  
    def __init__(self, q1, q2, q5):  
        self.mon = [q1, q2, q5]  
  
    def value(self):  
        return sum(self.mon)
```

```
def add_coins(self, v, q):  
    self.mon[(1,2,5).index(v)] += q  
  
def add(self, m):  
    for i in self.mon:  
        self.mon[i] += m.mon[i]
```

# Comparem els programes modificats

Si canviem la representació de Wallet, cal modificar els programes principals.  
Pels dos programes principals anteriors, el resultat fóra el següent:

## mainA.py

```
from wallet import Wallet

if __name__ == "__main__":
    m = Wallet(5,5,5)
    m.add_coins(1,4)
    m.add_coins(2,-1)
    m.add_coins(5,1)
    q = Wallet(1,2,3)
    m.add(q)
    print m.value()
```

## mainB.py

```
from wallet import Wallet

if __name__ == "__main__":
    m = Wallet(5,5,5)
    m.mon[0] += 4
    m.mon[1] += -1
    m.add_coins(5,1)
    q = Wallet(1,2,3)
    m.add(q)
    print m.value()
```

## Canvis

0

## Canvis

Molts!!



- Les instàncies tenen dues cares: mètodes i atributs.
- Si des del programa  $P$  accedim als atributs de les instàncies,  $P$  està vinculat a la implementació d'aquests atributs.  
⇒  
Quan els atributs canvien, cal modificar  $P$ .
- Si des del programa  $P$  accedim als mètodes de les instàncies però no als seus atributs,  $P$  no està vinculat a la implementació d'aquests atributs.  
⇒  
Quan els atributs canvien, NO cal modificar  $P$ .

## Principi d'«Information hiding»

Ocultar als usuaris dels objectes com emmagatzemen la informació i obligar-los a accedir-hi a través dels mètodes fa les aplicacions més robustes davant els canvis.

[Enunciat per David Parnas, 1972]

## L'amenaça del 2000

«L'arribada de l'any 2000 amenaça de provocar un caos sense precedents. El problema informàtic associat amb l'ús de dos dígits en la representació de la data (passarem del 99 al 00) pot afectar milions d'empreses d'arreu del món i fins i tot provocar problemes en el subministrament de serveis bàsics»



## L'amenaça del 2000



*L'arribada de l'any 2000 amenaça de provocar un caos sense precedents. El problema informàtic associat amb l'ús de dos dígits en la representació de la data (passarem del 99*

solucionar i de fet ja fa temps que hi treballen amb èxit. No obstant això, hi ha un símptoma que fa pensar que alguna cosa no acaba de rutllar. Fa un mes, l'empresa Microsoft, la principal multinacional de programes informàtics, va reconèixer que el seu producte més modern, el Windows 98, conté un error relacionat amb l'efecte 2000.

A banda de la nova relliscada de Bill Gates, els experts en informàtica reconeixen que el primer inconvenient per solucionar l'efecte 2000 és que existeixen milions de programes desenvolupats i usats

Diari Avui 9 de gener del 1999

<https://pandora.girona.cat/viewer.vm?id=0001189278&page=27>

- Els llenguatges de programació poden donar suport al principi d'Information Hiding a través d'atributs **privats**.
- Un programa no pot, ni que vulgui, accedir a un atribut privat d'una instància.
- Aquest és el cas de Java o C++.
- Les instàncies de Python, per contra, no tenen atributs privats.
- Quan un atribut cal que sigui considerat privat, la tradició demana que el seu identificador comenci per **guió baix**, com per exemple `_mon`.
- Els usuaris són respectuosos i no accedeixen als atributs prefixats per un guió baix. Privat per pacte!
- L'atribut `mon` de la classe `Wallet` és un bon candidat a ser prefixat.

# Un constructor més interessant

No és gens esbojarrat pensar que seria còmode poder instanciar la classe Wallet amb moneders buits. Fer-ho és senzill:

```
m = Wallet(0,0,0)
```

Si aquest és un cas freqüent podem donar als paràmetres del constructor **valors per omisió** definint-lo així:

```
def __init__(self, q1=0, q2=0, q5=0):  
    self.mon = {1:q1, 2:q2, 5:q5}
```

Això permet instanciar objectes d'aquesta manera:

```
m = Wallet()
```

```
n = Wallet(6)
```

```
o = Wallet(4,5)
```

I, fins i tot, fer crides com la següent, en que s'indica explícitament quin valor es transfereix a quin paràmetre. La resta de paràmetres prendran el valor per omisió:

```
m = Wallet(q5=10)
```

Finalment, la classe Wallet una vegada incorporades totes les millores, és:

wallet.py

```
class Wallet(object):
```

```
    def __init__(self, q1=0, q2=0, q5=0):  
        self._mon = {1:q1, 2:q2, 5:q5}
```

```
    def value(self):  
        return sum([k*v for k,v in self.mon.items()])
```

```
    def add_coins(self, v, q):  
        self._mon[v] += q
```

```
    def add(self,m):  
        for v in self._mon:  
            self._mon[v] += m._mon[v]
```

- 1 Estudi de la teoria. A partir de la referència principal i les transparències. Inclou provar els conceptes en el computador.
- 2 Confecció d'un xuletari i un 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.