

# **Tema 1 – Sessió 6**

## **Programació orientada a objecte**

---

## Exercici

Volem crear un conjunt de classes per poder guardar tots els productes del catàleg d'una empresa de **venda de productes online**.

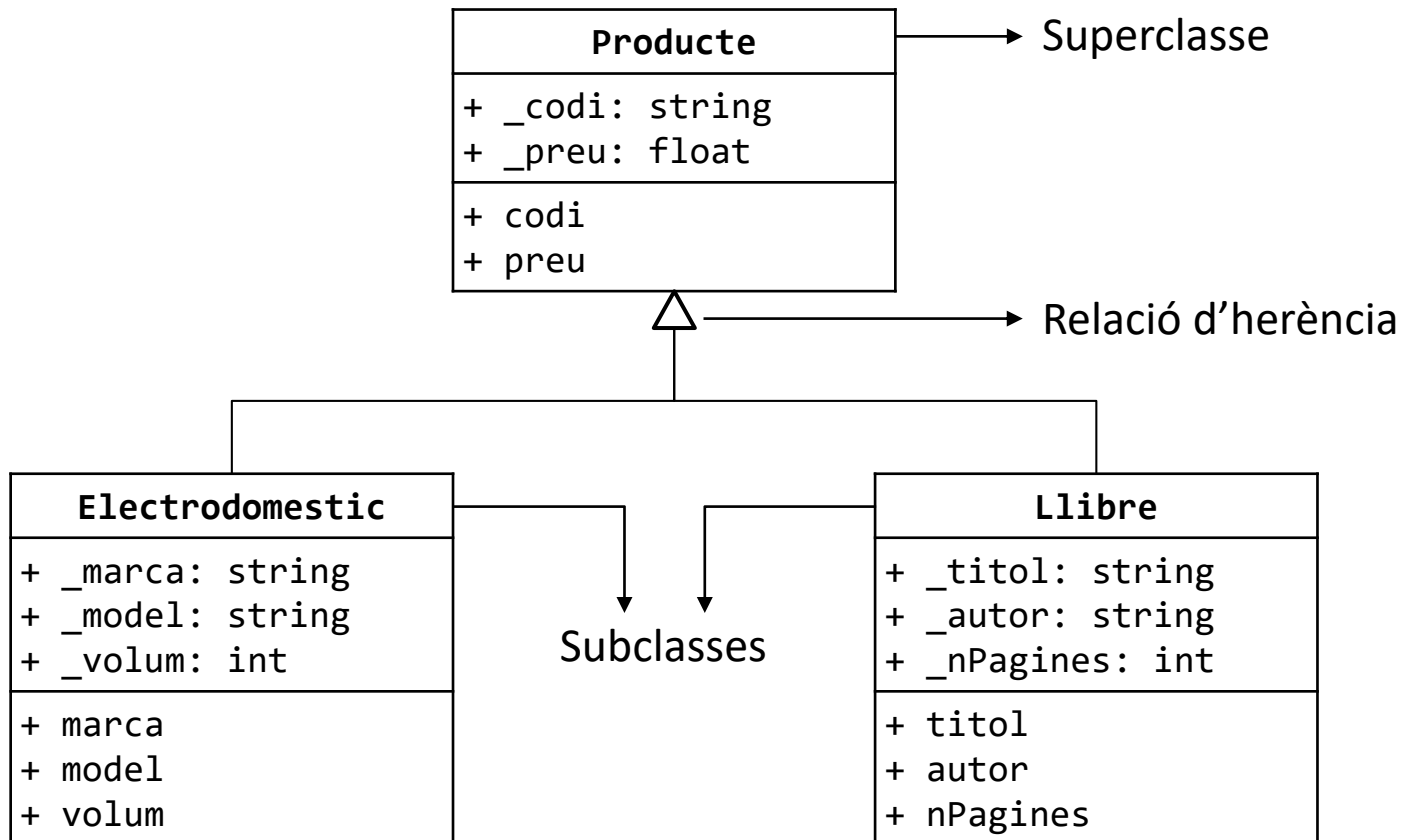
Al catàleg hi podria haver una varietat molt gran de productes. Tots els **productes** tenen un **codi** que els identifica i un **preu de venda**. A més a més, depenent del tipus de producte, cada producte té altres atributs que també s'han de guardar.

De moment, només considerarem **llibres i electrodomèstics**. Dels **llibres**, a més a més, volem guardar el **títol**, l'**autor** i el **nº de pàgines**. Dels **electrodomèstics**, la **marca**, el **model** i el **volum** que ocupa el seu embalatge.

- Quines classes hem de crear?
- Quins atributs ha de tenir cada classe?
- Quina relació han de tenir les classes entre elles?

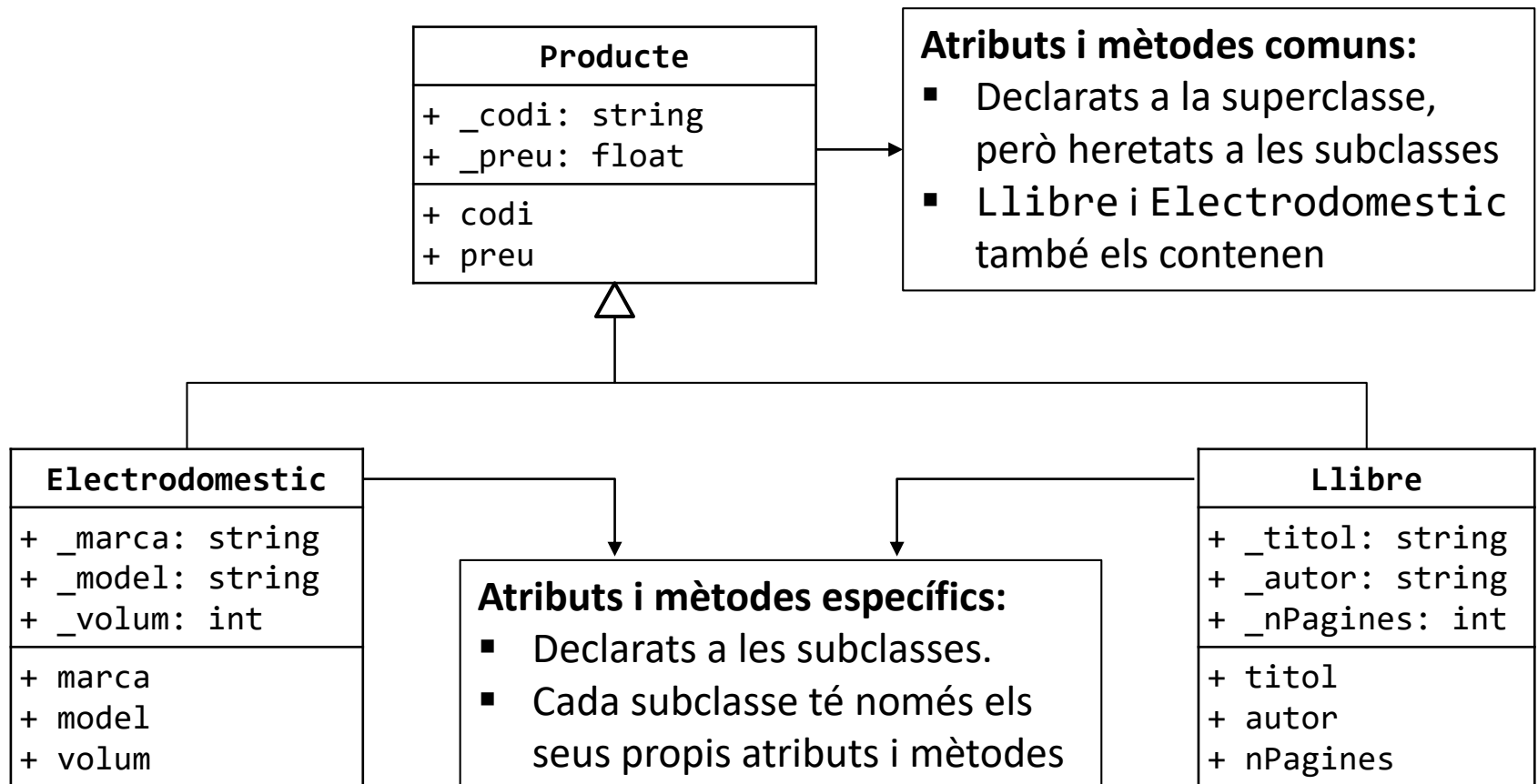
# Herència

- Utilització d'una classe genèrica ja existent (Producte) per crear altres classes més específiques (Llibre i Electrodomestic) que especialitzen la classe genèrica.
- Les classes més específiques (anomenades **subclasses**) són tipus especials de la classe genèrica (anomenada **superclasse**): *Els llibres i els electrodomèstics són tipus específics de **productes**.*



# Herència

- Les subclasses (o també **classes derivades**) **hereten** els **atributs** (dades) i els **mètodes** (comportament) de la superclasse (o també **classe base**).
- A més a més, les classes derivades poden afegir atributs i mètodes específics propis només de la subclasse



# Herència

## Declaració de la classe base

```
class Producte:  
    def __init__(self, codi = "", preu = 0.0):  
        self._codi = codi  
        self._preu = preu
```

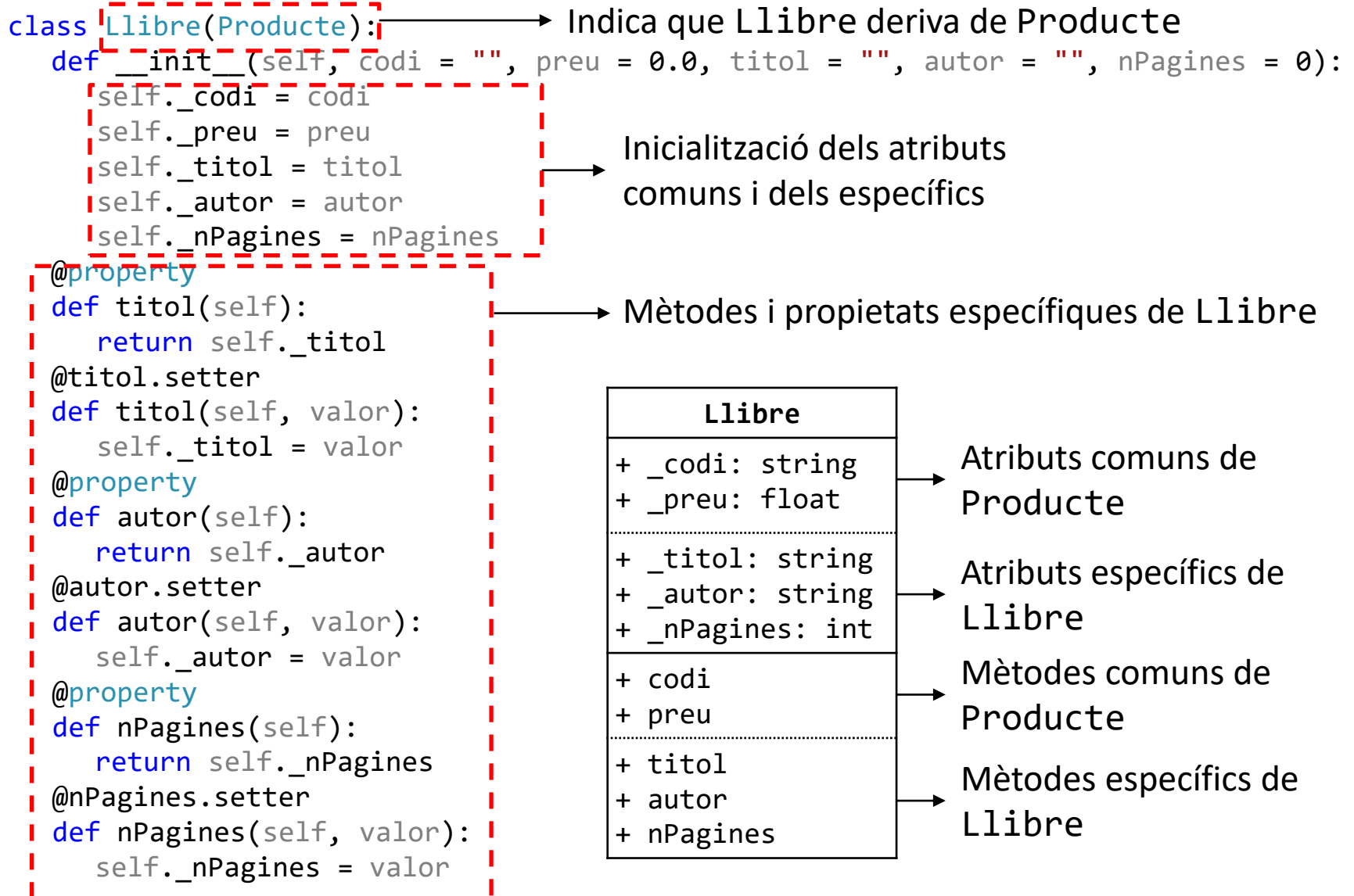
→ Declaració i inicialització dels atributs comuns

```
    @property  
    def codi(self):  
        return self._codi  
    @codi.setter  
    def codi(self, valor):  
        self._codi = valor  
  
    @property  
    def preu(self):  
        return self._preu  
    @preu.setter  
    def preu(self, valor):  
        self._preu = valor
```

→ Mètodes i propietats comunes

# Herència

## Declaració de la classe derivada



# Herència

## Declaració de la classe derivada

```
class Electrodomestic(Producte):  
    def __init__(self, codi = "", preu = 0.0, marca = "", model = "", volum = 0.0):  
        self._codi = codi  
        self._preu = preu  
        self._marca = marca  
        self._model = model  
        self._volum = volum  
  
    @property  
    def marca(self):  
        return self._marca  
    @marca.setter  
    def marca(self, valor):  
        self._marca = valor  
    @property  
    def model(self):  
        return self._model  
    @model.setter  
    def model(self, valor):  
        self._model = valor  
    @property  
    def volum(self):  
        return self._volum  
    @volum.setter  
    def volum(self, valor):  
        self._volum = valor
```

Electrodomestic deriva de Producte

Inicialització dels atributs comuns i dels específics

Mètodes i propietats específiques de Electrodomestic

Electrodomestic	
+ _codi: string + _preu: float	Atributs comuns de Producte
+ _marca: string + _model: string + _volum: int	Atributs específics de Electrodomestic
+ codi + preu	Mètodes comuns de Producte
+ marca + model + volum	Mètodes específics de Electrodomestic

# Herència

```
p = Producte("c_prod", 100)
print (p.codí, p.preu)
```

Inicialització de les classes derivades:  
atributs comuns i específics

```
l = Llibre("c_llibre", 100, "a_llibre", "t_llibre", 50)
e = Electrodomestic("c_elec", 500, "m_elec", "mod_elec", 10)
```

```
print (l.codí, l.preu, l.autor, l.títol, l.nPàgines)
print (e.codí, e.preu, e.marca, e.model, e.volum)
```

Utilització dels mètodes de  
la classe base Producte

Utilització dels mètodes de  
les classes derivades Llibre  
i Electrodomestic

```
class Llibre(Producte):
    def __init__(self, codi = "", ...):
        self._codi = codi
        self._preu = preu
        self._títol = títol
        self._autor = autor
        self._nPàgines = nPàgines
    @property
    def títol(self):
        ...
    @property
    def autor(self):
        ...
    @property
    def nPàgines(self):
        ...
```

```
class Producte:
    def __init__(self, codi = "", ...):
        self._codi = codi
        self._preu = preu
    @property
    def codi(self):
        ...
    @property
    def preu(self):
        ...
```



# Herència

```
class Producte:  
    def __init__(self, codi = "", preu = 0.0):  
        self._codi = codi  
        self._preu = preu
```

```
class Llibre(Producte):  
    def __init__(self, codi = "", preu = 0.0, titol = "", autor = "", nPagine = 0):  
        self._codi = codi  
        self._preu = preu  
        self._titol = titol  
        self._autor = autor  
        self._nPagine = nPagine
```

Redefinició (*overriding*) del mètode `__init__` de la classe base `Producte`:

- Les classes derivades poden redefinir el codi dels mètodes de la classe base per adaptar-los a la seva especificitat

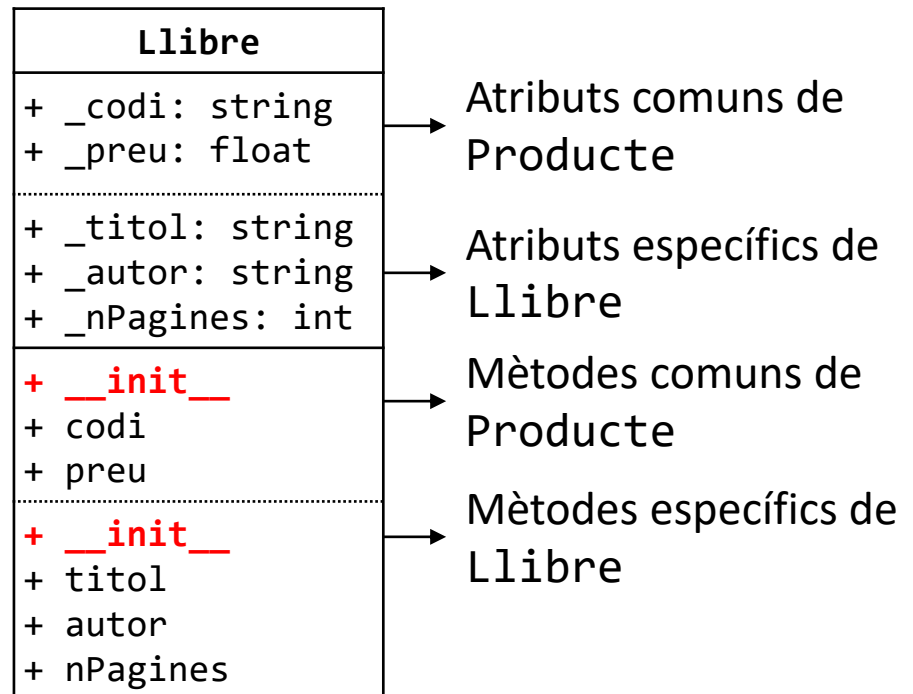
```
class Electrodomestic(Producte):  
    def __init__(self, codi = "", preu = 0.0, marca = "", model = "", volum = 0.0):  
        self._codi = codi  
        self._preu = preu  
        self._marca = marca  
        self._model = model  
        self._volum = volum
```

# Herència

```
class Producte:  
    def __init__(self, codi = "", preu = 0.0):  
        self._codi = codi  
        self._preu = preu
```

```
class Llibre(Producte):  
    def __init__(self, codi = "", preu = 0.0, titol = "", autor = "", nPagine = 0):  
        self._codi = codi  
        self._preu = preu  
        self._titol = titol  
        self._autor = autor  
        self._nPagine = nPagine
```

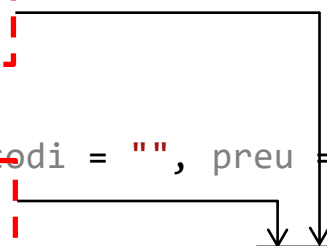
- Tenim dues versions del mètode `__init__`: la versió genèrica a `Producte` i la versió específica a `Llibre`
- Podem decidir quina volem utilitzar. Des de `Llibre` podem cridar al mètode `__init__` de `Producte`



# Herència

```
class Producte:
    def __init__(self, codi = "", preu = 0.0):
        self._codi = codi
        self._preu = preu

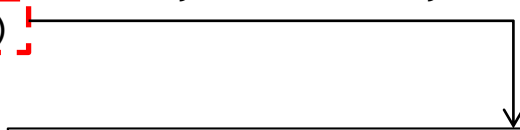
class Llibre(Producte):
    def __init__(self, codi = "", preu = 0.0, titol = "", autor = "", nPages = 0):
        self._codi = codi
        self._preu = preu
        self._titol = titol
        self._autor = autor
        self._nPages = nPages
```



- Codi repetit.
- Ho volem evitar: millor reutilització de codi.



```
class Llibre(Producte):
    def __init__(self, codi = "", preu = 0.0, titol = "", autor = "", nPages = 0):
        super().__init__(codi, preu)
        self._titol = titol
        self._autor = autor
        self._nPages = nPages
```



- `super()`: accés a la superclasse
- `super().__init__(codi, preu)`: crida al constructor de la classe base per inicialitzar els atributs comuns.

# Exercici

Completar la definició de les classes `Producte`, `Llibre` i `Electrodomestic`:

- Afegir mètodes a les tres classes per llegir de teclat les dades del producte, del llibre o de l'electrodomèstic.
- Afegir mètodes a les tres classes per mostrar per pantalla les dades del producte, del llibre o de l'electrodomèstic.
- Afegir un mètode `despesesEnviament` per determinar les despeses d'enviament d'un producte. Les despeses d'enviament es calculen de la forma següent:
  - Per tots els productes, si el preu del producte (sigui del tipus que sigui) és inferior a 100€ s'aplica una tarifa fixa de 1€. Si el preu del producte és superior a 100€ s'aplica un percentatge de l'1% sobre el preu del producte amb un màxim de 5€
  - A més a més, pels llibres, si el nº de pàgines és superior a 500€ s'aplica un sobrecost d'1€ sobre les despeses calculades segons el punt anterior.
  - Pels electrodomèstics s'hi afegeix un sobrecost d'1€ per cada 20 litres (o fracció) del seu volum.

Fer un programa que vagi llegint dades de productes de qualsevol tipus i els guardi a una llista de productes. Quan s'indiqui que ja no es volen introduir més productes s'han de mostrar per pantalla les dades de tots els productes que s'han introduït, incloent les despeses d'enviament.

# Exercici

Recuperem una versió lleugerament modificada de la classe Poligon que hem utilitzat en exercicis anteriors.

```
class Poligon:
    maxim = 1000
    def __init__(self, vertexs = []):
        assert len(vertexs)==0 or len(vertexs)>2
        self._vertexs = vertexs

    def afegeixVertex(self, pt):
        self._vertexs.append(pt)

    def perimetre(self):
        perimetre = 0
        for index in range(len(self._vertexs) - 1):
            perimetre += (self._vertexs[index] - self._vertexs[index+1])
        perimetre += (self._vertexs[0] - self._vertexs[-1])
        return perimetre

    def area(self):
        raise NotImplementedError('implementat a les subclasses')

    def __str__(self):
        resultat = "["
        for v in self._vertexs:
            resultat = resultat + str(v)
        resultat = resultat + "]"
        return resultat
```

## Exercici

1. Derivar a partir de la classe `Polygon` una subclasse `Triangle` específica per guardar i gestionar la informació de triangles.
  - Redefinir el constructor perquè, a més a més de la inicialització genèrica del polígon, assegurí que el nº de vèrtexs que es passen com a paràmetre a la inicialització és igual a 3
  - Redefinir el mètode `afegeixVertex` perquè provoqui una excepció (no es poden afegir vèrtexs a un triangle ja correctament inicialitzat)
  - Redefinir el mètode `area` perquè calculi l'àrea del triangle aplicant aquesta fórmula (on  $a$ ,  $b$  i  $c$  són la longitud de cadascun dels costats del triangle):

$$s = \frac{a + b + c}{2}$$
$$area = \sqrt{(s(s - a)(s - b)(s - c))}$$

## Exercici

2. Derivar a partir de la classe Polygon una subclasse Quadrat específica per guardar i gestionar la informació de quadrats.
  - Afegir un atribut (amb les seves propietats pel getter i setter) per guardar la longitud del costat del quadrat
  - Redefinir el constructor perquè, a més a més de la inicialització genèrica del polígon, asseguri que el nº de vèrtexs que es passen com a paràmetre a la inicialització és igual a 4 i que tots els costats tenen la mateixa longitud
  - Redefinir el mètode `afegeixVertex` perquè provoqui una excepció (no es poden afegir vèrtexs a un triangle ja correctament inicialitzat)
  - Redefinir el mètode `area` perquè calculi l'àrea del quadrat a partir de la longitud del costat.
  - Redefinir el mètode `perimetre` perquè calculi el perímetre del quadrat a partir de la longitud del costat.