# Clojure - Herència amb clausures

Als exercicis de classe he treballat com fer objectes sense classe aprofitant el mecanisme de les clausures. En aquest projecte aprofundirem en el tema tractant també l'herència.

Partirem d'una classe `animal` que té un mètode `parlar` (sortida: `grr`). A partir d'aquesta classe podem definir dues subclasses `gat` i `tigre` a partir de la classe animal. Per a la subclasse `gat` redefinirem el mètode `parlar` (sortida: `mèu`) i per a la subclasse `tigre` volem que cridi a la de la classe mare (`animal`).

Així, en cridar el mètode d'un `gat` o `animal` s'hauria d'accedir al seu propi mètode, però en fer-ho amb `tigre` hauria de cridar al de la seva superclasse.

Així mateix, volem definir una funció polimòrfica que rebi un animal i un natural `n` i que cridi `n` vegades al mètode parlar que correspongui.

La solució al problema plantejat en Python seria quelcom com el codi següent:

```
class Animal:
    def parlar(self):
        print("grr")

class Gat(Animal):
    def parlar(self):
        print("mèu")

class Tigre(Animal):
    pass

def parlarN(animal, n):
   print(" ".join([animal.parlar() for _ in range(n)]))

animal = Animal()
gat = Gat()
tigre = Tigre()

parlarN(animal, 3)    👉 grr grr grr
parlarN(gat, 3)       👉 mèu mèu mèu
parlarN(tigre, 3)     👉 grr grr grr
```

**Enunciat**:

- Implementeu l'exemple anterior en clojure seguint les restriccions següents:

- Heu d'implementar les classes amb clausures
- La classe `tigre` ha d'accedir al mètode del pare, no podeu replicar el mètode parlar.

**Joc de proves**:

```
(use 'animals :reload-all)

(def a (animal))
(parlarn a 3)
👉 grr grr grr

(def g (gat (animal)))
(parlarn g 3)
👉 mèu mèu mèu

(def t (tigre (animal)))
(parlarn t 3)
👉 grr grr grr
```

**Regles del joc**:

- Heu de fer el treball en parelles.
- Heu de lliurar un arxiu `clj` amb el nom i cognoms dels dos membres del grup en un comentari a la part de dalt de l'arxiu.
- Lliureu només un dels membres del grup.

**Annex**:

Si no sabeu python, a continuació teniu un exemple de com funcionaria aquest tipus de mecanisme en C++:

```
class Animal {
    virtual void parlar() {
        print("grr");
    }
}

class Gat: Animal {
    virtual void parlar() {
        print("mèu");
    }
}

void parlarN(Animal* animal, n: int) {
    for (int i = 0; i <n; ++i) {
        animal->parlar();
    }
}


Animal animal;
Gat gat;

parlarN(animal, 3);
parlarN(&gat, 3);
```