FORMATION PYTHON

TP SUR LA PROGRAMMATION OBJET

TP 1

- Créer une classe
- Créer une méthode d'instance
- Créer des attributs statiques
- Créer une méthode d'affichage automatique

On veut créer un jeu de carte de type Bataille à 52 cartes. Chaque carte est identifiée par sa **valeur** et par sa **couleur**.

Les couleurs possibles : Cœur, Carreau, Trefle, Pique
Les valeurs sont : 2 à 10, puis Valet, Dame, Roi, As

On propose le diagramme de classe ci-dessous :

Carte
couleur **valeur** ***__init__(self, val, coul)*** **__str__(self)** **afficher(self)**

1- Créer la classe **Carte** dans le fichier **Carte.py** et définissez son constructeur en laissant les attributs publics (pour l'instant)

2- Testez cette classe dans le shell Python :
   a. Créer un objet **c1** représentant le 14 de Cœur : afficher la valeur, afficher la couleur.
   b. Faites un **print** qui affiche par exemple « 14 **de** Cœur ».

3- Ajoutez la méthode ***affiche(self)*** ayant pour unique instruction un **print** qui affiche par exemple « 14 de Cœur ». Puis testez la méthode dans le shell Python.

4- On va remplacer les valeurs et couleurs par les codes ci-dessous. Pour créer une carte, on fera désormais **c = Carte(8, 1)** si on veut créer un « 8 de Carreau ».

Code	Valeur
0	-
1	-
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	Valet
12	Dame
13	Roi
14	As

Code	Couleur
0	Coeur
1	Careau
2	Trefle
3	Pique

a- Modifiez le constructeur pour qu'il soit impossible de créer une carte :
- dont la valeur n'existe pas.
- dont la couleur n'existe pas.

b- Modifiez la méthode ***affiche(self)*** pour qu'elle affiche correctement comme ceci « Dame de Carreau » au lieu de « 12 de 1 » **en utilisant obligatoirement l'instruction if**.

5- Attributs statiques :
On veut ajouter les attributs statiques **valeurs** et **couleurs** à la classe.

couleurs contiendra : Cœur, Carreau, Trefle, Pique
valeurs contiendra : None, None, 2 à 10, puis Valet, Dame, Roi, As

a- Ajoutez ces deux attributs comme statiques à la classe sous forme de tuple (trouvez le bon endroit où les ajouter).

b- Modifiez ***affiche(self)*** pour qu'elle utilise ces deux attributs statiques ou attributs de classe (trouvez la bonne syntaxe).

c- Testez votre méthode ***affiche()*** dans le shell Python.

6- Définir la méthode automatique ***__str__(self)*** identique à affiche(self) mais avec obligatoirement un **return** et testez.

7- Testez l'ensemble en créant quelques cartes.

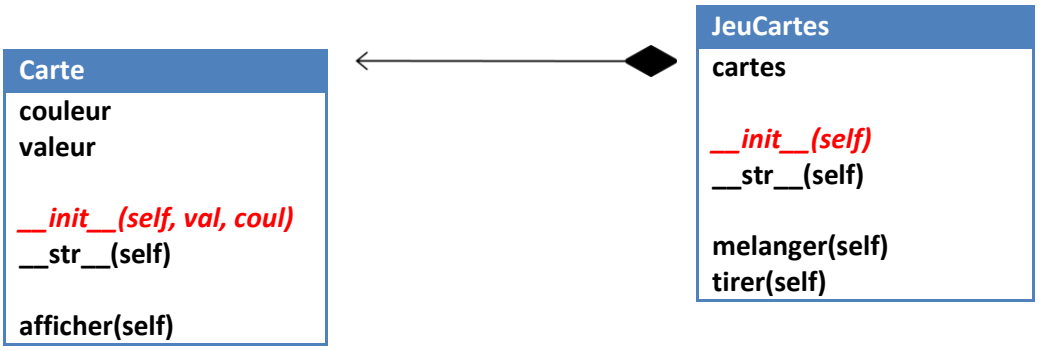
TP 2

- Coder la composition
- Utiliser un module en l'important
- Utiliser dir(.) et help(.) pour l'aide
- Mettre en œuvre l'encapsulation

La Bataille commence par un jeu de carte (52 cartes) qu'on va mélanger (méthode ***melanger(self)***) et et distribuer aux joueurs (***méthode tirer()***). Il faut disposer de la classe **JeuCartes**.

Composition

La modélisation montre que l'attribut cartes est constitué d'objets de la classe Carte créée précédemment. La classe **JeuCartes** utilise donc Carte : c'est la composition.



1- Créer la classe **JeuCartes** dans **JeuCartes.py** : l'attribut **cartes** est un tableau que le constructeur devra remplir avec les 52 cartes.

   a- Créer le constructeur
   b- Testez dans le shell Python en créant un jeu **j = JeuCartes()**
   c- Dans le shell, faire **print(j.cartes)** : que se passe t-il ? Quelle est la bonne écriture pour afficher une carte ?
   d- Dans le shell, faire **len(j.cartes)** : que se passe t-il ?

2- Ajoutez la méthode ***__str__(self)*** qui affiche les 52 cartes comme ceci : **« 2 de Cœur, 2 de Carreau, ….., As de Pique »**.

3- Ajoutez la méthode ***melanger(self)*** qui va mélanger les cartes en utilisant le module **random**. Utilisez **pydoc** avec **dir(.)** et **help(.)** ou la documentation de python https://docs.python.org/3 pour trouver la bonne méthode de random.

   NB : n'oubliez pas d'importer le module random.

4- Ajoutez la méthode ***tirer(self)*** qui va distribuer en utilisant la méthode ***pop(index)*** sur la liste **cartes** : pop(0) dépilera le 1er élément.

5- Modifiez ***tirer(self)*** pour gérer le cas où l'index n'existe pas avec un bloc try/except. Le bloc except sera configuré sur IndexError et fera un print « Il n'ya plus de cartes dans le jeu ».

6- Revoir la classe **Carte** et rendez les attributs **valeur** et **couleur** privés, puis rendez l'attribut **cartes** privé dans **JeuCartes**.

7- Testez l'ensemble avec le code :

```
from Carte import Carte
j = JeuCartes()
c = j.tirer()
print(c)
```

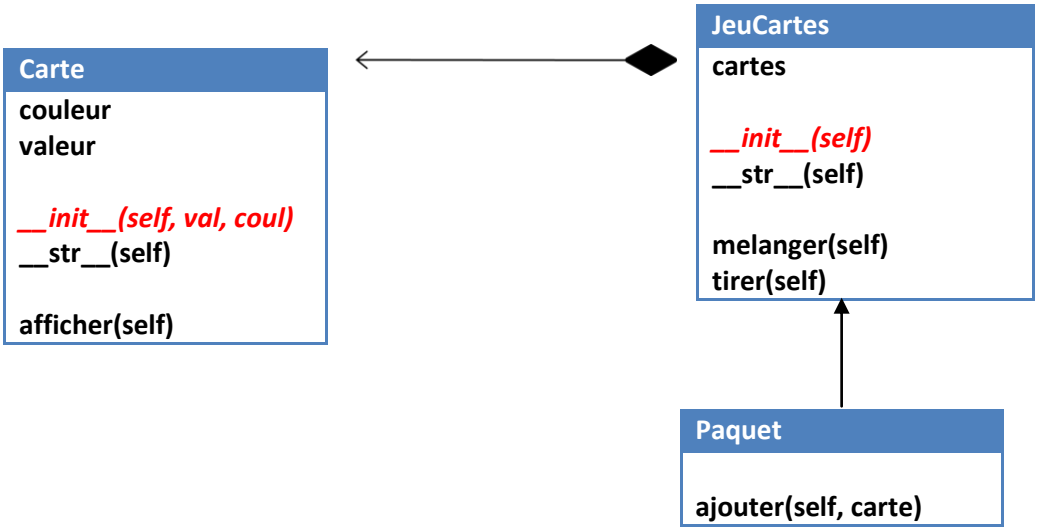
TP 3

- Coder l'héritage
- Créer une property

Lorsque le jeu de cartes est distribué, les 2 joueurs reçoivent un paquet de cartes que nous modélisons avec la classe Paquet. Paquet « est une sorte » de JeuCartes, il va hériter de lui : un paquet peut être mélangé et on peut y tirer une carte pour la lancer dans la bataille.

Héritage

Un paquet est un jeu de cartes particulier. Il est tout d'abord vide puis on y ajoute des cartes au cours de la distribution des cartes.



ajouter(self, carte) : ajoute une carte au paquet.

1- Comme un paquet est vide au départ, pour pouvoir créer un jeu de carte vide, modifiez le constructeur de **JeuCartes** :

a- ***def __init__(self, vide = False)*** :
b- Ne remplissez les 52 cartes que si vide est à False :

```
if not vide :
    for val in range(2,15) :
        for coul in range(4) :
            self.__cartes.append(Carte(val, coul))
```

2- **L'attribut cartes dans JeuCartes étant privé, Paquet ne pourra l'hériter.** Créer un getter ***getCartes(self)***, un setter ***setCartes(self, carte)*** et une property équivalente **cartes** qui pourra être utilisé par la classe fille Paquet et par les autres méthodes de JeuCartes.

3- Créer la classe **Paquet** dans le fichier **Paquet.py** qui hérite de JeuCartes :

a- Son constructeur doit appeler celui du parent, avec le paramètre **vide** à **True** car le paquet sera vide au départ : ***super().__init__(True)***

b- Créer la méthode ***ajouter(self, carte)*** qui ajoute une carte au paquet.

4- Testez la classe Paquet dans le shell Python :

```
>>> from Paquet import Paquet
>>> p = Paquet()
>>> print(p)

>>> from Carte import Carte
>>> c = Carte(12, 2)
>>> print(c)
Dame de Trefle
>>> p.ajouter(c)
>>> print(p)
Dame de Trefle

>>> print(p.tirer())
Dame de Trefle
>>> pint(p)
```