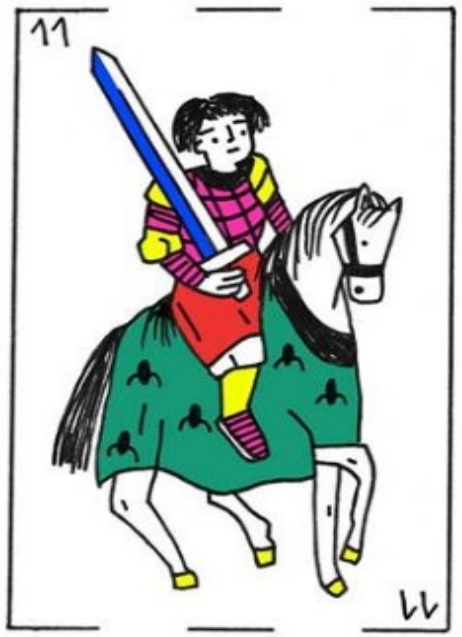
TIA: Técnicas de Inteligencia Artificial

Herencia



Clases del juego:

Mazo

Carta

Mano

ManoPersona

LaMona

Carta: Atributos y métodos

- Atributos:
  - palo
  - rango

```
class Carta(object):
    palo_nombres = ["Bastos", "Oros", "Copas", "Espadas"]
    rango_nombres = [None, "As", "2", "3", "4", "5", "6", "7", "Sota", "Caballo", "Rey"]

def __init__(self, palo=0, rango=2):
        self.palo = palo
        self.rango = rango:
```

Carta: Atributos y métodos

- Métodos:
  - sobreescribir str y cmp (que se empleará en el sort)

```
def __str__(self):
    return '%s of %s' % (Carta.rango_nombres[self.rango],
                         Carta.palo_nombres[self.palo])

def __cmp__(self, other):
    rdo = 0
    if self.rango == other.rango:
        if self.palo > other.palo: rdo = 1
        elif self.palo < other.palo: rdo = -1
        else:rdo = 0
    return rdo
```

Carta: Atributos y métodos

- Atributos:
  - palo
  - rango

```
class Carta(object):
    palo_nombres = ["Bastos", "Oros", "Copas", "Espadas"]
    rango_nombres = [None, "As", "2", "3", "4", "5", "6", "7", "Sota", "Caballo", "Rey"]

def __init__(self, palo=0, rango=2):
        self.palo = palo
        self.rango = rango:
```

Carta: Atributos y métodos

- Atributos:
  - palo
  - rango

Metodos sobre-escritos:

```
def __cmp__(self, otra):

def __eq__(self, otra):

def __ne__(self, otra):
```

Mazo: Atributos y métodos

- Atributos:
  - cartas =[]

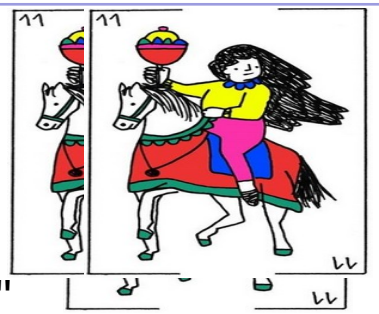
```
class Mazo(object):

def __init__(self):
      self.cartas  = [ ]
      for palo in range(4):
            for rango in range(1, 11):
                carta = Carta(palo, rango)
                self.cartas.append(carta)

def __str__(self):
        res = [ ]
        for carta in self.cartas:
            res.append(str(carta))
        return '\n'.join(res)
```

Mazo: Atributos y métodos

- Métodos:



```
def anadir_carta(self, carta):
    """Anade una carta al mazo."""

def eliminar_carta(self, carta):
    """Elimina un acarta del mazo."""

def esta_carta(self, carta):
    """Comprueba si una carta es mazo. True
       si esta False sino"""

def pop_carta(self, i=-1):
    """Saca una carta del mazo.
    i: el indice de la carta a sacar
    (por defecto -1, es decir, la ultima); """
```

```
def barajar(self):
    """Baraja las cartas del mazo."""

def ordenarAsc(self):
    """Ordena las cartas del mazo en orden ascendente."""

def mover_cartas(self, mano, num):
    """Mueve num cartas desde el mazo a la mano.
    mano: objeto destino perteneciente a la clase Mano
    num:  numero de cartas a desplazar """

def imprimir(self):
    """Imprime las cartas contenidas en el mazo"""

def esta_vacio(self):
    """True si la lista cartas esta vacia.""")
```

Mano: Atributos y métodos

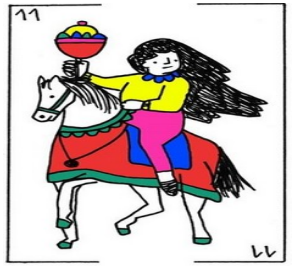
- Atributos:
  - Los que hereda (cartas)
  - jugador

```
def __init__(self, jugador=''):
        self.cartas = [ ]
        self.jugador = jugador

def imprimir(self):
        toPrint ='la mano de {0} '.format(self.jugador)
         #### TODO
```

ManoPersona: Atributos y métodos

- Atributos:
  - Los que hereda (Mano)



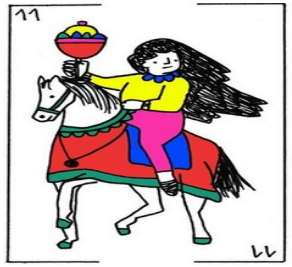
```
def eliminar_parejas(self):
        cont = 0
        salir = False#chivato para salir cuando la pareja que se pretende eliminar no sea tal

        while not salir:
            super().imprimir()
            print("Que cartas forman pareja? de 0 a {0}".format(len(self.cartas)-1))
            idxCarta,idxPareja = map(int, input().split(','))
            carta = self.cartas[idxCarta]
            posiblePareja = self.cartas[idxPareja]
            pareja = Carta(3 - carta.palo, carta.rango)#
            if #TODO comprobar si la que debiera ser pareja es la esperada y eliminar ambas de la mano
                print("En la mano de {0} forman par {1}  {2}".format(self.jugador, carta, pareja))
                cont += 1
            else:
                salir = True
        return cont
```

LaMona: Atributos y métodos

- Atributos:
  - Los que hereda (Mano)

```
def eliminar_parejas(self):
        cont = 0
        #TODO
        return cont
```



Se pide:

- Completar el código de los siguientes métodos:

Carta:

```
        def __eq__(self, other):
```

Mazo:

```
        def mover_cartas(self, mano, num):
    def esta_vacio(self):
```

Mano:

```
    def imprimir(self):
        """ Reescribir tal que indique el nombre del propietario de esa mano"""
```

ManoPersona y LaMona:

```
        def eliminar_parejas(self):
```

Se pide (opcionalmente):

- Implementar el juego tal que:

un jugador pueda decidir a quien le roba una carta y luego se ha de comprobar si forma nueva pareja con la robada la máquina podrá almacenar las que le ha robado cada cual de forma que si precisa una carta que le han robado anteriormente pueda recuperarla el juego acabara cuando alguien se quede sin cartas o haya una ronda completa en la que nadie haga parejas en ese caso el que tenga mas cartas en la mano pierde.

Para ejecutar el juego:

>>> python JuegoCartas.py

Se comenzará ejecutardo el main

```
if __name__ == "__main__":
    print('Ejecutando como programa principal')
    juego = JuegoCartas()
    n_jugadores = int(input("Cuantos jugadores tiene el juego (sin contar las
    print(n_jugadores)
    manos = juego.inicializar_partida(n_jugadores)

    juego.mazo.repartir_cartas(juego.manos,40)

    juego.jugar_ronda()

    ###############################################################
    #TODO opcional si quereis podeis implementar el juego
    # tal que un jugador pueda decidir
    # a quien le roba una carta y luego se ha de comprobar si forma
    # nueva pareja con la robada
    # la maquina podra almacenar las que le ha robado cada cual
    # de forma que si precisa una carta que le han robado anteriormente pueda
    # recuperarla
    # el juego acabara cuando alguien se quede sin cartas
    # o haya una ronda completa en la que nadie haga parejas
    # en ese caso el que tenga mas cartas en la mano pierde
    ###############################################################
```