oggetti & game

Python

oggetto

- le applicazioni utilizzano ***oggetti***
- gli oggetti hanno uno ***stato*** interno in campi (attributi) privati
  - concetto di ***incapsulamento*** (black box)
- gli oggetti hanno un comportamento funzioni (***metodi)*** pubblici
- gli oggetti con le stesse caratteristiche fanno parte di una classe

- ogni oggetto ha una ***classe*** di origine (*è istanziato da una classe*)
- la classe definisce la stessa ***forma iniziale*** (campi e metodi) a tutti i suoi oggetti
- ma ogni oggetto
  - ha la sua ***identità***
  - ha uno stato e una locazione in memoria distinti da quelli di altri oggetti
    - sia instanze di classi diverse che della stessa classe

definizione della classe

- ***incapsulamento*** dei dati: convenzione sui nomi
  - prefisso _ per i nomi dei campi privati

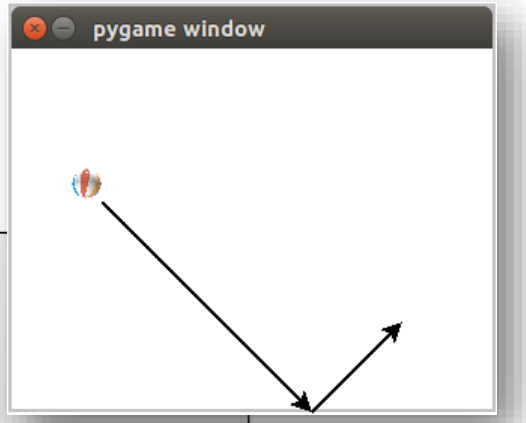
```
class Pallina:

    def __init__(self, x: int, y: int):
        self._x = x
        self._y = y
        self._dx = 5
        self._dy = 5
        self._w = 20
        self._h = 20
    # ...
```

- costruzione di oggetti (***istanziazione***)
- **`__init__`**: metodo ***inizializzatore***
- eseguito ***automaticamente*** alla creazione di un oggetto
- **`self`**: primo parametro di tutti i metodi
  - non bisogna passare un valore esplicito
  - rappresenta l'oggetto di cui si chiama il metodo
  - permette ai metodi di accedere ai campi

```
p = Pallina(40, 80)  # Creazione e inizializzazione
```

le funzioni (metodi)



- espongono ***servizi*** ad altri oggetti

```
ARENA_W, ARENA_H = 320, 240

class Pallina:
    # ...
    def muovi(self):
        if not (0 <= self._x + self._dx <= ARENA_W - self._w):
            self._dx = -self._dx
        if not (0 <= self._y + self._dy <= ARENA_H - self._h):
            self._dy = -self._dy
        self._x += self._dx
        self._y += self._dy

    def posizione(self) -> (int, int, int, int):
        return self._x, self._y, self._w, self._h
```

applicazione


```
from cl_pallina import Pallina  # Pallina definite in cl_pallina.py

# Crea due oggetti, istanze della classe Pallina
p1 = Pallina(40, 80) p2 = Pallina(80, 40)

for i in range(25):
    print('Ball 1 @', p1.posizione())
    print('Ball 2 @', p2.posizione())
    p1.muovi()
    p2.muovi()
```

self (primo parametro)

- il primo parametro di ogni metodo si chiama ***self***
  - (*per convenzione*)
- self rappresenta l'oggetto di cui viene invocato il metodo

animazione di due palline

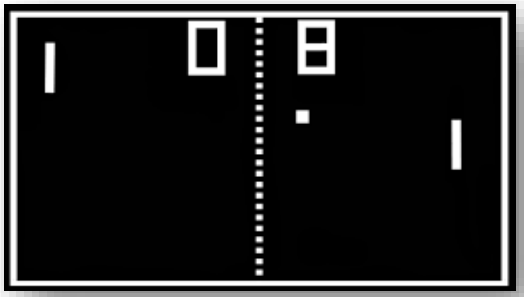
A. Ferrari

```
import g2d
from cl_pallina import Pallina, ARENA_W, ARENA_H

def update():
    g2d.fill_canvas((255, 255, 255))  # BG
    p1.muovi()
    p2.muovi()
    g2d.draw_rect((127, 127, 127), p1.posizione())
    g2d.draw_rect((127, 127, 127), p2.posizione())

p1 = Pallina(40, 80)
p2 = Pallina(80, 40)
g2d.init_canvas((ARENA_W, ARENA_H))
g2d.main_loop(update, 1000 // 30)
```

- ***campi***: memorizzano i ***dati caratteristici*** di una istanza
  - ogni pallina ha la sua posizione (x, y) e la sua direzione (dx, dy)
- ***parametri***: ***passano*** altri ***valori*** ad un metodo
  - se alcuni dati necessari non sono nei campi
- ***variabili locali***: memorizzano ***risultati parziali***
  - generati durante l'elaborazione del metodo
  - nomi ***cancellati*** dopo l'uscita dal metodo
- ***variabili globali***: definite ***fuori*** da tutte le funzioni
  - usare sono se strettamente necessario
  - meglio avere qualche parametro in più, per le funzioni



modello per giochi su arena

- i giochi su arena sono caratterizzati
  - da personaggi che si muovono nell'arena di gioco
  - i personaggi interagiscono con altri personaggi del gioco
  - ogni elemento del gioco può essere considerato un personaggio
  - il comportamento dei singoli personaggi dipende dalla loro tipologia
  - la logica di gioco prevede il raggiungimento di situazioni particolari
    - fine del gioco
    - superamento di un livello
    - …

i personaggi

- le caratteristiche comuni a ogni personaggio vengono definite nella classe Personaggio
- è una interfaccia che dovrà essere implementata da ogni specifico personaggio del gioco
- l'interfaccia prevede i metodi:
  - ***muovi***
    - movimento caratteristico ad ogni fotogramma del gioco
  - ***urta***
    - gestione della collisione con un altro personaggio del gioco
  - ***posizione***
    - zona di arena occupata dal personaggio
  - ***disegna***
    - visualizzazione sulla finestra di gioco

classe Personaggio

A. Ferrari

```
class Personaggio():
    '''Interfaccia che deve essere implementata dai vari tipi
       di personaggi del gioco '''
    def muovi(self):
        '''Chiamato da Gioco, ad ogni turno del personaggio '''
        pass   #funzione da implementare da parte dei personaggi

    def urta(self, altro: 'Personaggio'):
        '''Chiamato da Gioco, quando il personaggio (self)
           entra in collisione con un altro personaggio (altro) '''
        pass   #funzione da implementare da parte dei personaggi

    def posizione(self) -> (int, int, int, int):
        '''Restituisce il rettangolo che contiene il personaggio
           tupla di 4 valori interi: (left, top, larghezza, altezza)'''
        pass   #funzione da implementare da parte dei personaggi

    def disegna(self):
        '''Disegna il personaggio nella finestra del gioco '''
        pass   #funzione da implementare da parte dei personaggi
```

l'arena di gioco

- l'arena di gioco contiene la lista dei personaggi
- ha metodi per
  - inserire un nuovo personaggio
  - eliminare un personaggio
  - far muovere tutti i personaggi
  - verificare collisioni fra i personaggi
  - ...

class Gioco (1)


```
class Gioco():
    '''Generico gioco 2D, cui vengono assegnate le dimensioni di gioco
       e che contiene la lista dei personaggi del gioco '''
    def __init__(self, larghezza: int, altezza: int):
        '''Crea una Gioco con specifica altezza e larghezza
           e lista di personaggi inizialmente vuota '''
        self._w, self._h = larghezza, altezza
        self._Personaggi = []

    def aggiungi(self, p: Personaggio):
        '''Aggiunge un personaggio al gioco. I pesonaggi sono gestiti
        seguendo il loro ordine di inserimento '''
        if p not in self._Personaggi:
            self._Personaggi.append(p)

    def rimuovi(self, p: Personaggio):
        '''Elimina un personaggio dal gioco '''
        if p in self._Personaggi:
            self._Personaggi.remove(p)
```

class Gioco(2)

A. Ferrari

```
def muovi_tutti(self):
    '''chiama il metodo muovi di ogni personaggio
       dopo aver effettuato il movimento verifica
       se è avvenuta un collisione tra il personaggio
       e un altro personaggio e in tal caso chiama
       il metodo urta di entrambi '''
    Pers = list(reversed(self._Personaggi))
    for p in Pers:
        posizione_precedente= p.posizione()
        p.muovi()
        if p.posizione() != posizione_precedente:
            for altro in Pers:
                if altro is not p and self.verifica_collisione(p, altro):
                        p.urta(altro)
                        altro.urta(p)

def verifica_collisione(self, p1: Personaggio, p2: Personaggio) -> bool:
    '''Verifica se i due personaggi (parametri) sono in collisione
       (bounding-box collision detection) '''
    x1, y1, w1, h1 = p1.posizione()
    x2, y2, w2, h2 = p2.posizione()
    return (y2 < y1 + h1 and y1 < y2 + h2
        and x2 < x1 + w1 and x1 < x2 + w2
        and p1 in self._Personaggi and p2 in self._Personaggi)
```

i veri personaggi

- ogni gioco ha i suoi personaggi specifici
- ogni personaggio ha un comportamento specifico
- realizza (implementa) in modo diverso l'interfaccia Personaggio
- ha eventualmente altre caratteristiche e comportamenti

es Palla (1)

A. Ferrari

```
class Palla(Personaggio):
    '''
    Pallina che si muove in diagonale e rimbalza
    se incontra i limiti dell'gioco
    e se si scontra con altri personaggi '''
    def __init__(self, gioco, x, y):
        self._x, self._y = x, y
        self._w, self._h = 64, 64
        self._speed = 5
        self._dx, self._dy = self._speed, self._speed
        self._gioco = gioco
        self._immagine = g2d.load_image("palla.png")
        gioco.aggiungi(self)

    def muovi(self):
        ''' Rimbalza sui bordi del gioco '''
        gioco_w, gioco_h = self._gioco.dimensione()
        if not (0 <= self._x + self._dx <= gioco_w - self._w):
            self._dx = -self._dx
        if not (0 <= self._y + self._dy <= gioco_h - self._h):
            self._dy = -self._dy
        self._x += self._dx
        self._y += self._dy
```

es Palla(2)

A. Ferrari

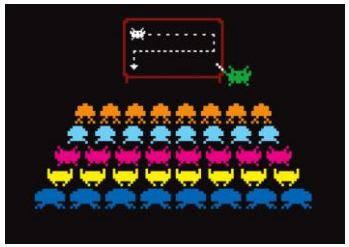
```
    def urta(self, altro_pers):
        if not isinstance(altro_pers, Fantasma):
            x, y, w, h = altro_pers.posizione()
            if x < self._x:
                self._dx = self._speed  # rimbalzo a ds
            else:
                self._dx = -self._speed # rimbalzo a sn
            if y < self._y:
                self._dy = self._speed  # rimbalzo in basso
            else:
                self._dy = -self._speed # rimbalzo in alto

    def posizione(self):
        return self._x, self._y, self._w, self._h

    def disegna(self):
        '''Disegna il personaggio nella finestra del gioco '''
        g2d.draw_image(self._immagine, self.posizione())
```

4.1 animazione di una pallina

- partire dalla classe ***Pallina***
- eseguire l'animazione:
  - per ogni frame, chiamare il metodo ***muovi*** della pallina
  - rappresentare un rettangolo o un cerchio nella ***posizione aggiornata*** della pallina
- ***modificare*** però il metodo muovi
  - la pallina si sposta sempre di pochi pixel in orizzontale
  - la pallina non si sposta verticalmente
  - se esce dal bordo destro, ricompare al bordo sinistro e viceversa



4.2 classe degli alieni

- creare una classe ***Alien***
  che contenga i ***dati*** ed il ***comportamento*** dell'alieno
  - campi privati: x, y, dx
  - metodo ***muovi*** per avanzare
  - metodo ***posizione*** per ottenere la posizione attuale
- istanziare un ***oggetto*** Alien e farlo muovere sullo schermo
  - chiamare il metodo muovi ad ogni ciclo
  - visualizzare un rettangolo nella posizione corrispondente

definire nella classe delle opportune costanti