CAPITOLUL III PROGRAMARE ORIENTATĂ-OBIECT

În acest capitol vom învăța:

- Noțiunile de clasă, obiect, încapsulare, moștenire;
- Să realizăm programe simple folosind clase și obiecte.

3.1 Aspecte teoretice

Lumea reală este formată din obiecte. Fiecare obiect are anumite trăsături și un anumit comportament.

În programare, trăsăturile sunt reținute prin intermediul câmpurilor iar comportamentul este modelat prin intermediul metodelor. Funcțiile definite în interiorul claselor poartă numele de metode.

Aplicația 1

Numele clasei trebuie să înceapă cu literă mare.

Inițializarea câmpurilor obiectului se realizează prin intermediul unei metode __init__ numită **constructor**.

```
class Pisica:
   def __init__(self,nume,rasa):
      self.nume=nume
      self.rasa=rasa

pisica1=Pisica('Ica', 'Persana')
print(pisica1.nume,' ',pisica1.rasa)
pisica2=Pisica('Pica','Siameza')
print(pisica2.nume,' ',pisica2.rasa)
```

Se creează obiectul pisica1 prin apelul constructorului.

Ica

Pica

În aplicația următoare vom încerca, utilizând biblioteca Tkinter, să realizăm un program care să deseneze linii pe ecran, pornind de la ideea graficii turtle.

```
class Broasca:
   def __init__(self,x,y):
      self.x=x
      self.y=y

coco=Broasca(20,20)
print(coco.x,' ',coco.y)
ema=Broasca(30,30)
print(ema.x,' ',ema.y)
```

Creează obiectul coco poziționat în punctul de coordonate (20, 20).

Creează obiectul ema poziționat în punctul de coordonate (30, 30).

3.2 Încapsularea

O caracteristică a programării orientate obiect este încapsularea. Aceasta presupune definirea câmpurilor și metodelor în cadrul clasei.

Imaginați-vă că sunteți într-o rachetă care călătorește spre Planeta programatorilor. Nu puteți comunica cu exteriorul decât prin apăsarea unor butoane. Butoanele reprezintă metodele.

3.3 Moștenirea

Așa cum fiecare om moștenește anumite trăsături și comportamente de la părinții lui, și în programare există posibilitatea ca o clasă să moștenească câmpuri și metode de la altă clasă.

Unul din beneficiile posibilității de a crea subclase (construirea unei clase care este bazată pe o altă clasă numită clasă de bază) este acela că subclasa moștenește caracteristicile și comportamentul clasei părinte. Datorită moștenirii, un programator poate efectua o modificare în clasa părinte și această modificare va fi propagată la toate subclasele derivate. Mai mult, componentele clasei părinte pot fi transmise cu ajutorul moștenirii pe parcursul mai multor generații de subclase.

Sintetizând, moștenirea este un mecanism prin care o subclasă acumulează caracteristici și comportament de la una sau mai multe clase strămoș. Subclasa diferă de părinți fie prin faptul că i s-au adăugat caracteristici și comportamente noi, fie prin faptul că au fost modificate cele moștenite de la părinți.

În acest moment ne gândim că ar fi frumos fiecare broscuță să aibă asociată o culoare. Să modificăm clasa anterioară? Este o soluție dar mai există una mult mai elegantă. Să scriem o nouă clasă, numită Broasca_colorata, care va moșteni clasa broasca.

```
class Broasca:
   def __init__(self,x,y):
      self.x=x
      self.y=y
class Broasca_colorata(Broasca):
   def __init__(self,culoare,x,y):
      Broasca.__init__(self,x,y)
      self.culoare=culoare
coco=Broasca_colorata('Red',20,20)
print(coco.culoare,' ',coco.x,' ',coco.y)
ema=Broasca_colorata('Yellow',30,30)
 print(ema.culoare,' ',ema.x,' ',ema.y)
```

Aplicația 1 Mișcarea obiectelor este realizată cu ajutorul metodelor. Modificăm programul anterior adăugând clasei Broasca metoda deseneaza. Metoda trasează o linie din punctul în care este poziționată broasca până în punctul transmis ca parametru. Vă mai amintiți cărei funcții din biblioteca turtle îi corespunde o astfel de funcție?

```
from tkinter import *

class Broasca:
    def __init__(self,x,y):
        self.x=x
        self.y=y
class Broasca_colorata(Broasca):
    def __init__(self,culoare,x,y):
        Broasca.__init__(self,x,y)
        self.culoare=culoare
    def deseneaza_linie(self,x1,y1):
         plansa.create_line(self.x,self.y,x1,y1,fill=self.culoare)
         self.x=x1
         self.y=y1
coco=Broasca_colorata('Red',20,20)
ema=Broasca_colorata('Yellow',30,30)

fereastra=Tk()
fereastra.title("Broscute")
fereastra.resizable(0,0)
fereastra.wm_attributes("-topmost",1)
plansa=Canvas(fereastra,width=500,height=400,bd=0, highlightthickness=0)
plansa.pack()

coco.deseneaza_linie(50,40)
coco.deseneaza_linie(80,120)
ema.deseneaza_linie(200,230)
```

Aplicația 2 Presupunem că dorim să reprezentăm broasca sub forma unui cerc. Definim un nou câmp id în care reținem id_ul cercului. Apoi, utilizând funcția move, mutăm broscuța în poziția curentă.

```
from tkinter import *

class Broasca:
    def __init__(self,x,y):
        self.x=x
        self.y=y
class Broasca_colorata(Broasca):
    def __init__(self,plansa,culoare,x,y):
        Broasca.__init__(self,x,y)
        self.plansa=plansa
        self.culoare=culoare
        self.vechiul_x=x
        self.vechiul_y=y
```

```
        self.id=plansa.create_oval(x,y,x+5,y+5,fill=self.culoare)
    def deseneaza_linie(self,x1,y1):
        plansa.create_line(self.x,self.y,x1,y1,fill=self.culoare)
        self.vechiul_x=self.x
        self.vechiul_y=self.y
        self.x=x1
        self.y=y1
        self.plansa.move(self.id,self.x-self.vechiul_x,self.y-self.vechiul_y)
fereastra=Tk()
fereastra.title("Broscute")
fereastra.resizable(0,0)
fereastra.wm_attributes("-topmost",1)
plansa=Canvas(fereastra,width=500,height=400,bd=0, highlightthickness=0)
plansa.pack()

coco=Broasca_colorata(plansa,'Red',20,20)
ema=Broasca_colorata(plansa,'Yellow',30,30)

ema.deseneaza_linie(50,40)
ema.deseneaza_linie(80,120)
coco.deseneaza_linie(200,230)
coco.deseneaza_linie(250,320)
coco.deseneaza_linie(10,10)
```

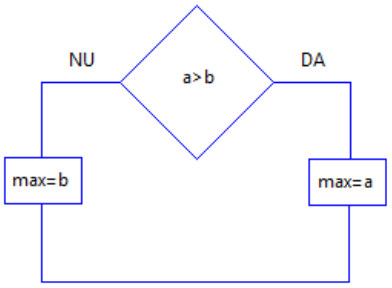
Adăugăm programului funcția ridica_stiloul. Vă mai amintiți cărei funcții din biblioteca turtle îi corespunde o astfel de funcție?

```
from tkinter import *

class Broasca:
    def __init__(self,x,y):
        self.x=x
        self.y=y
class Broasca_colorata(Broasca):
    def __init__(self,plansa,culoare,x,y):
        Broasca.__init__(self,x,y)
        self.plansa=plansa
        self.culoare=culoare
        self.vechiul_x=x
        self.vechiul_y=y
        self.id=plansa.create_oval(x,y,x+5,y+5,fill=self.culoare)
    def deseneaza_linie(self,x1,y1):
        plansa.create_line(self.x,self.y,x1,y1,fill=self.culoare)
        self.vechiul_x=self.x
        self.vechiul_y=self.y
        self.x=x1
        self.y=y1
        self.plansa.move(self.id,self.x-self.vechiul_x,self.y-self.vechiul_y)
    def ridica_stiloul (self, x1,y1):
        plansa.create_line(self.x,self.y,x1,y1,fill='white')
        self.old_x=self.x
```

```
        self.old_y=self.y
        self.x=x1
        self.y=y1
        self.plansa.move(self.id,self.x-self.vechiul_x,self.y-self.vechiul_y)

fereastra=Tk()
fereastra.title("Broscute")
fereastra.resizable(0,0)
fereastra.wm_attributes("-topmost",1)
plansa=Canvas(fereastra,width=500,height=400,bd=0, highlightthickness=0, bg='white')
plansa.pack()

coco=Broasca_colorata(plansa,'Red',100,120)
ema=Broasca_colorata(plansa,'Yellow',20,20)

ema.deseneaza_linie(50,40)
ema.deseneaza_linie(80,120)
coco.deseneaza_linie(200,230)
coco.deseneaza_linie(250,320)
coco.ridica_stiloul(200,250)
coco.deseneaza_linie(20,40)
```

Vom desena cu alb pe fundal alb pentru a simula ridicarea stiloului.

Aplicația 3 – rulați programul de mai jos. Trageți cu mouse-ul obiectele astfel încât să obțineți figura de mai jos.




```
from tkinter import *
fereastra = Tk()
plansa = Canvas(fereastra, width=600, height=500, bg='white')
plansa.pack()
plansa.create_polygon(200,50,150,100,200,150,250,100,outline="blue",fill="white")
plansa.create_rectangle(30,30,80,60,outline="blue")
plansa.create_rectangle(30,100,80,130,outline="blue")
plansa.create_text(30,150,text="a>b")
plansa.create_text(30,170,text="max=a")
plansa.create_text(30,190,text="max=b")
plansa.create_text(30,210,text="DA")
plansa.create_text(30,230,text="NU")
plansa.create_line(30,250,80,250,fill="blue")
plansa.create_line(30,270,80,270,fill="blue")
```

```
plansa.create_line(30,350,30,400,fill="blue")
plansa.create_line(50,350,50,400,fill="blue")
plansa.create_line(70,350,70,400,fill="blue")
plansa.create_line(90,350,90,400,fill="blue")
plansa.create_line(110,370,310,370,fill="blue")

class Miscare():
  def __init__(self):
    self.item = 0; self.previous = (0, 0)
  def select(self, event):
    widget = event.widget
    # Convertește coordonatele ecran în coordonate canvas
    xc = widget.canvasx(event.x); yc = widget.canvasx(event.y)
    self.item = widget.find_closest(xc, yc)[0]
    self.previous = (xc, yc)
    print((xc, yc, self.item))
  def drag(self, event):
    widget = event.widget
    xc = widget.canvasx(event.x); yc = widget.canvasx(event.y)
    plansa.move(self.item, xc-self.previous[0], yc-self.previous[1])
    self.previous = (xc, yc)

# se creeaza obiectul schema_logica care este o instanta a clasei miscare

schema_logica = Miscare()

# leaga evenimentele de metodele corespunzatoare
plansa.bind("<Button-1>", schema_logica.select)
plansa.bind("<B1-Motion>", schema_logica.drag)
```

Bibliografie

1. Jason R. Briggs, Python pour les kids, editions Eyrolles, 2013,
2. Paul Barry, David Griffiths, Head First Programming, O'Reilly,2000
3. Jim Muller – "The Great Logo Adventure: Discovering Logo on and Off the Computer", editura Doone Publications
4. Stan Munson – *"TerrapinLogo tutorial"*, ISBN 109876543

Această carte a fost înregistrată la Biblioteca Națională cu ISBN 978-973-0-30332-2.