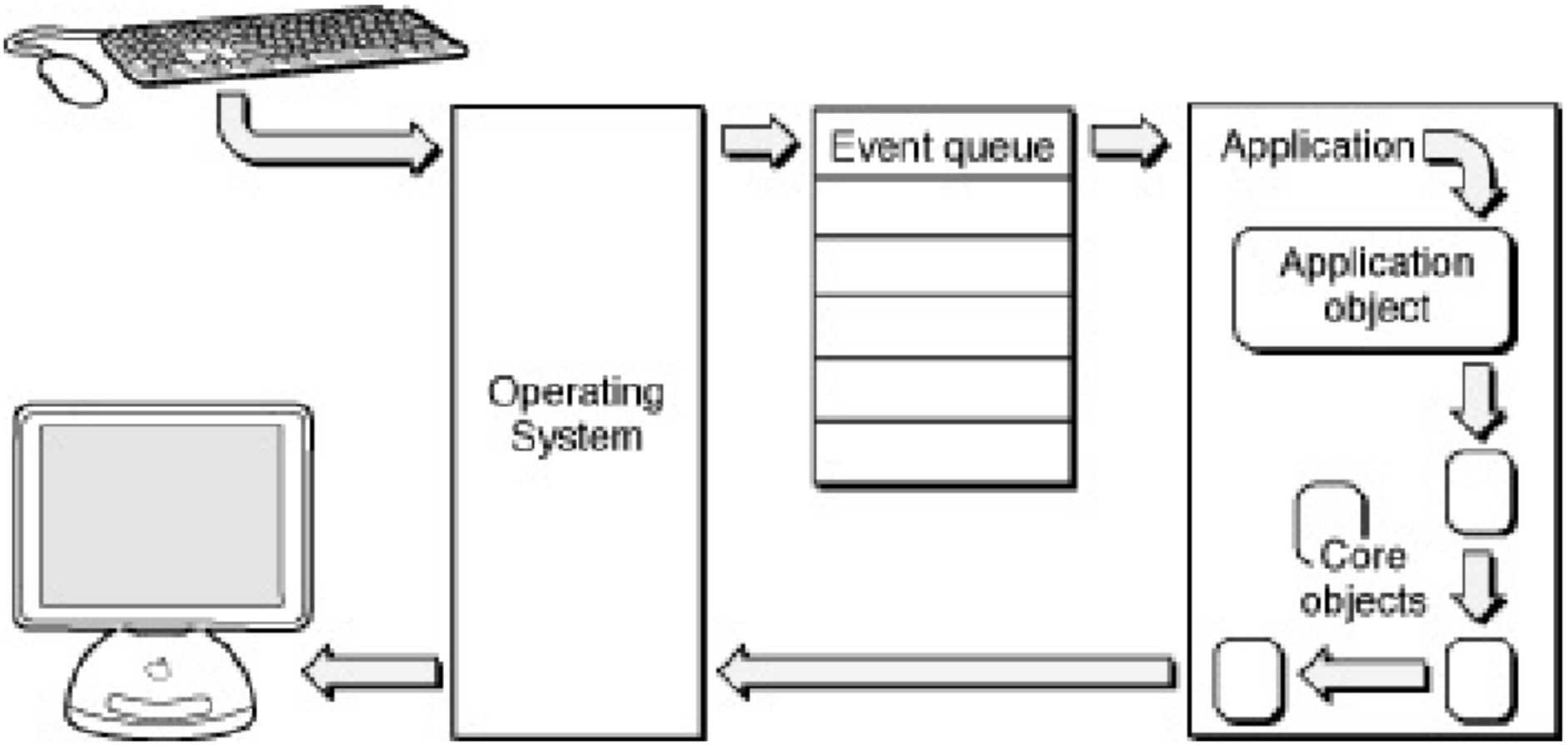
Paradigma Orientata Eveniment

Cursul nr. 5

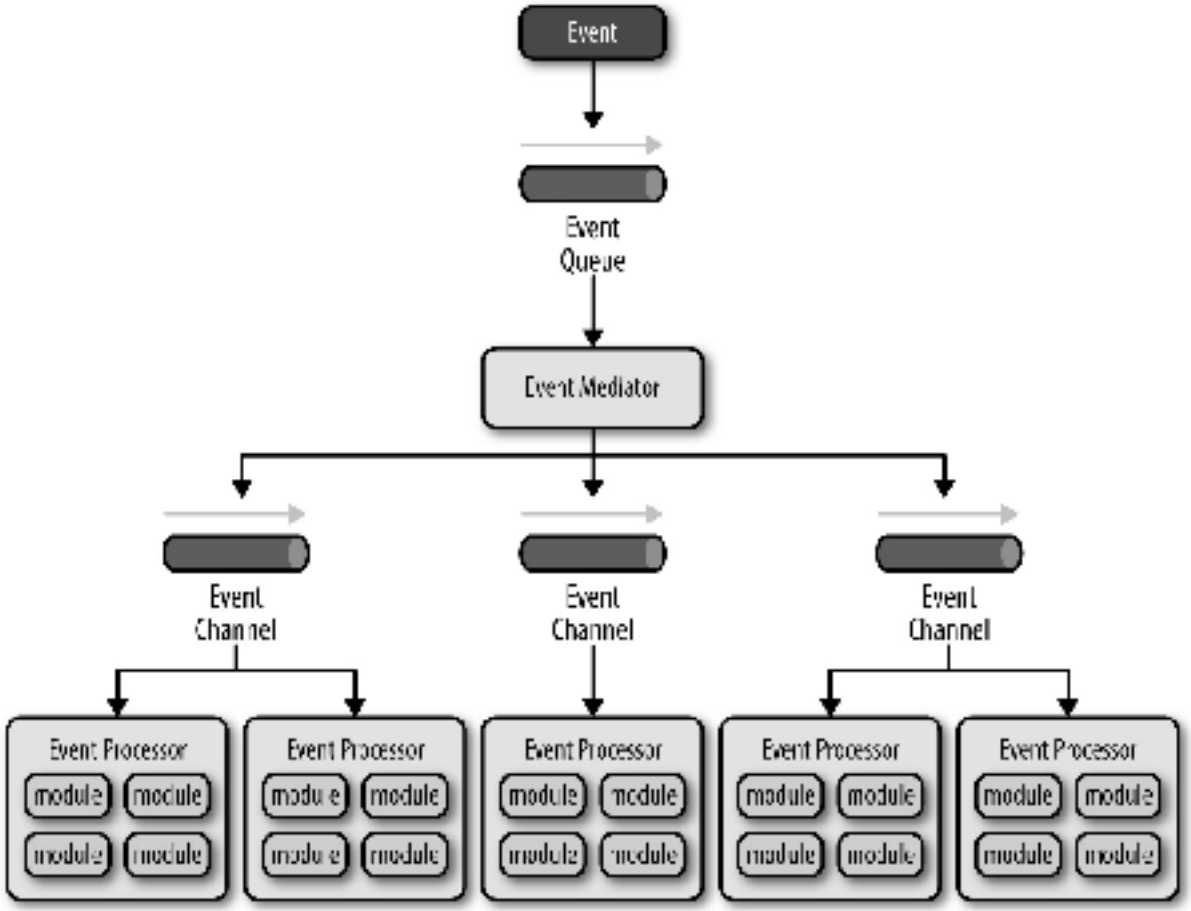
Mihai Zaharia

Ce este un eveniment?

O manieră de gestionare/tratare



Abordarea modernă de proiectare arhitecturală



Tratare evenimente asincrone

Polling

- **Interrupt-driven**

Event-driven

Paradigma orientată eveniment

Metoda 1 - Polling

Interacțiunea este guvernată de o buclă infinită:

```
Loop forever:
{ i=1..n
   read input i
   answer to input i
  inc i }
```

Metoda 2 - Interrupt-driven

1. Activează dispozitivul, apoi
2. Începe procesarea de bază (instalează sistemul de gestiune a evenimentelor/intreruperi)
3. Așteaptă apariția unei întreruperi
4. La apariția unei întreruperi
   1. Salvează starea curentă (schimbare context)
   2. Încarcă și execută metoda de tratare a întreruperii
   3. Restaurează contextul anterior
   4. Go to #1

Metoda #3: Event-driven

Interacțiunea este din nou guvernată de o buclă:

```
main()
{
 ...inițializează structurile de date ale aplicației ...
 ...inițializează și lansează în execuție  GUI....
 // intră în bucla de eveniment
 while(true)
        {
        Event e = get_event();//primește evenimentul
        process_event(e); // tratează evenimentul
        }
}
```

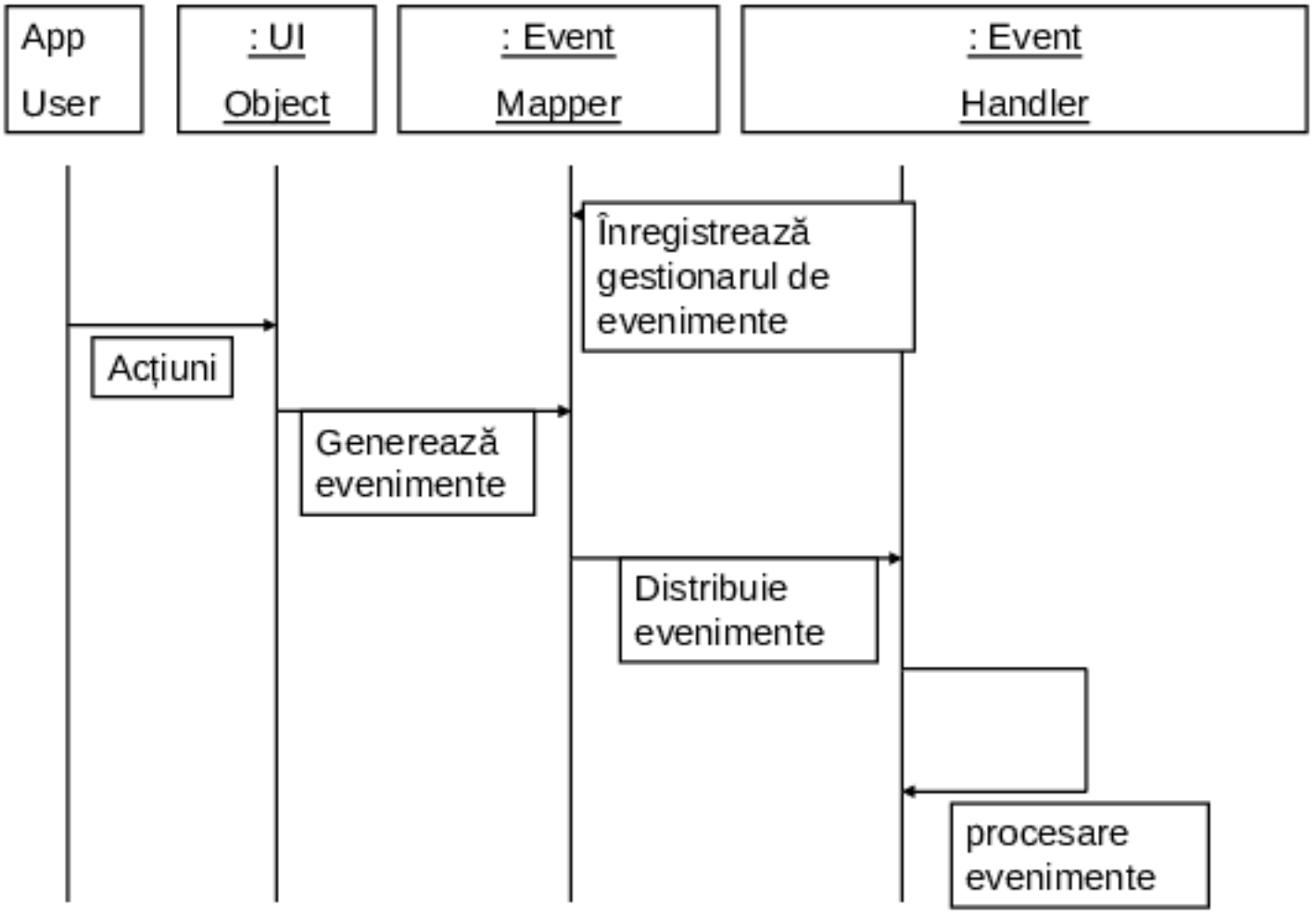
Avantajele procesării orientate eveniment

- Sunt mai portabile
- Permit tratarea mai rapidă
- Se pot folosi în time-slicing)
- Încurajează reutilizarea codului
- se potrivesc cu oop

Componentele unui program simplu bazat pe EDP

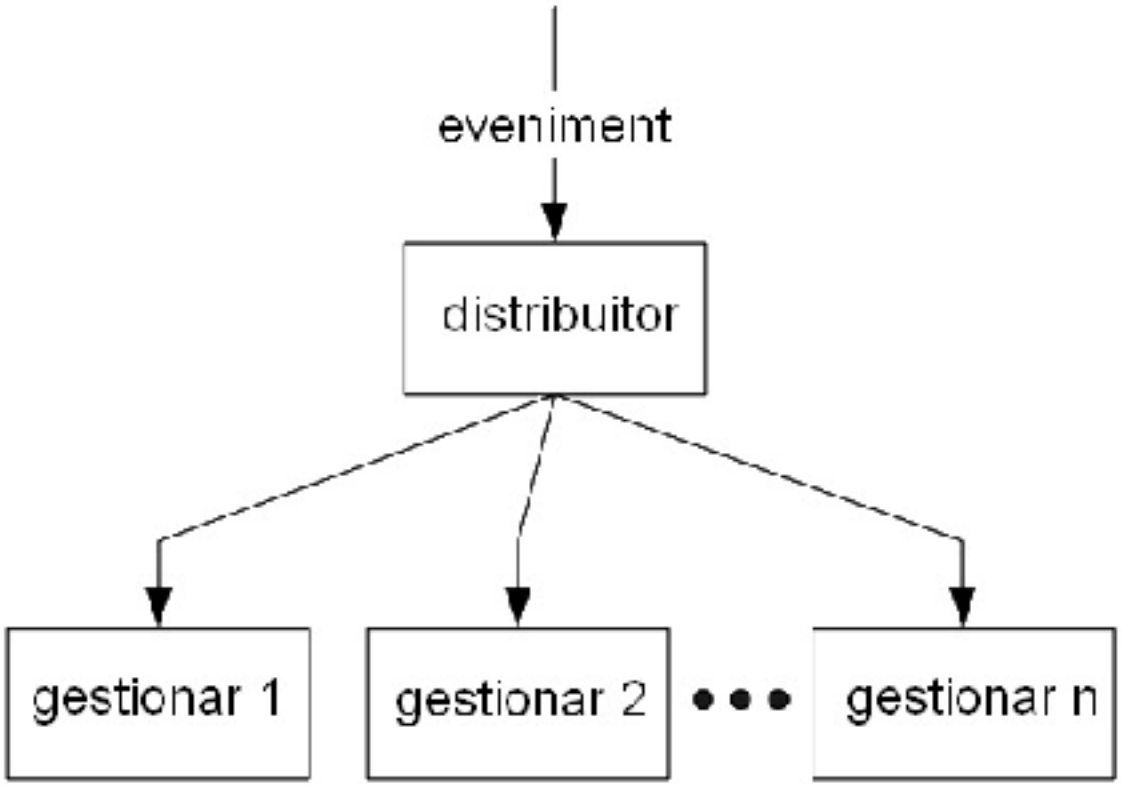
1. Generatoare de evenimente
2. Sursa evenimentelor
3. Bucla de evenimente
4. Gestionari de evenimente
5. Event mapper
6. Înregistrarea evenimentelor

Diagrama de secvență pentru EDP



Tratarea evenimentelor

Dispatcher/Distribuitor/Mapper:



Programarea orientată pe eveniment

EDP – aplicații consolă

```
#include <iostream>
using namespace std;
int value; // globală
int main() { // Inițializare
        char s = '+';
        value = 0;
        while(1) { // buclă tratare eveniment
                cout << "Selectați operația (+,-,q): \n";
                cin >> s; //aștept evenimentul de la utilizator
                switch(s)
                 { //event mapper
                 case '+': //event registration
                         add(); break; //event handler
                 case '-': //înregistrare eveniment
                         sub(); break; //gestionare eveniment
                 case 'q': //înregistrare eveniment
                         exit(1);  //gestionare eveniment
                  }
                }
        return(1);
        }
```

EDP – aplicații consolă

```
// gestionare pentru evenimente
void add ()
{ // gestionar eveniment "+"
int in;
cout << "Introduceți un întreg: \n";
cin >> in;
value += in;
cout << "Valoarea curentă este: " << value << "\n";
}
void sub ()
{ // "-" event handler
int in;
cout << "Introduceți un întreg: \n";
cin >> in;
value -= in;
cout << "Valoarea curentă este: " << value << "\n";
}
```

EDP – aplicații GUI

	Secvențial	EDP
Text	puțin	modest
GUI	inutil	majoritar

Proiectarea unei aplicații EDP - GUI

- Stabilirea interacțiunii vizuale și a evenimentelor: **Look & Feel**

GUI proiectată corect

- O interfață este bună dacă are următoarele caracteristici:
  - **Eleganța**
  - **Îl ghidează** pe looser
  - Oferă **informații ajutătoare**
  - Folosește o **ierarhie** de interfețe
  - Permite ca utilizatorul să facă **greșeli**

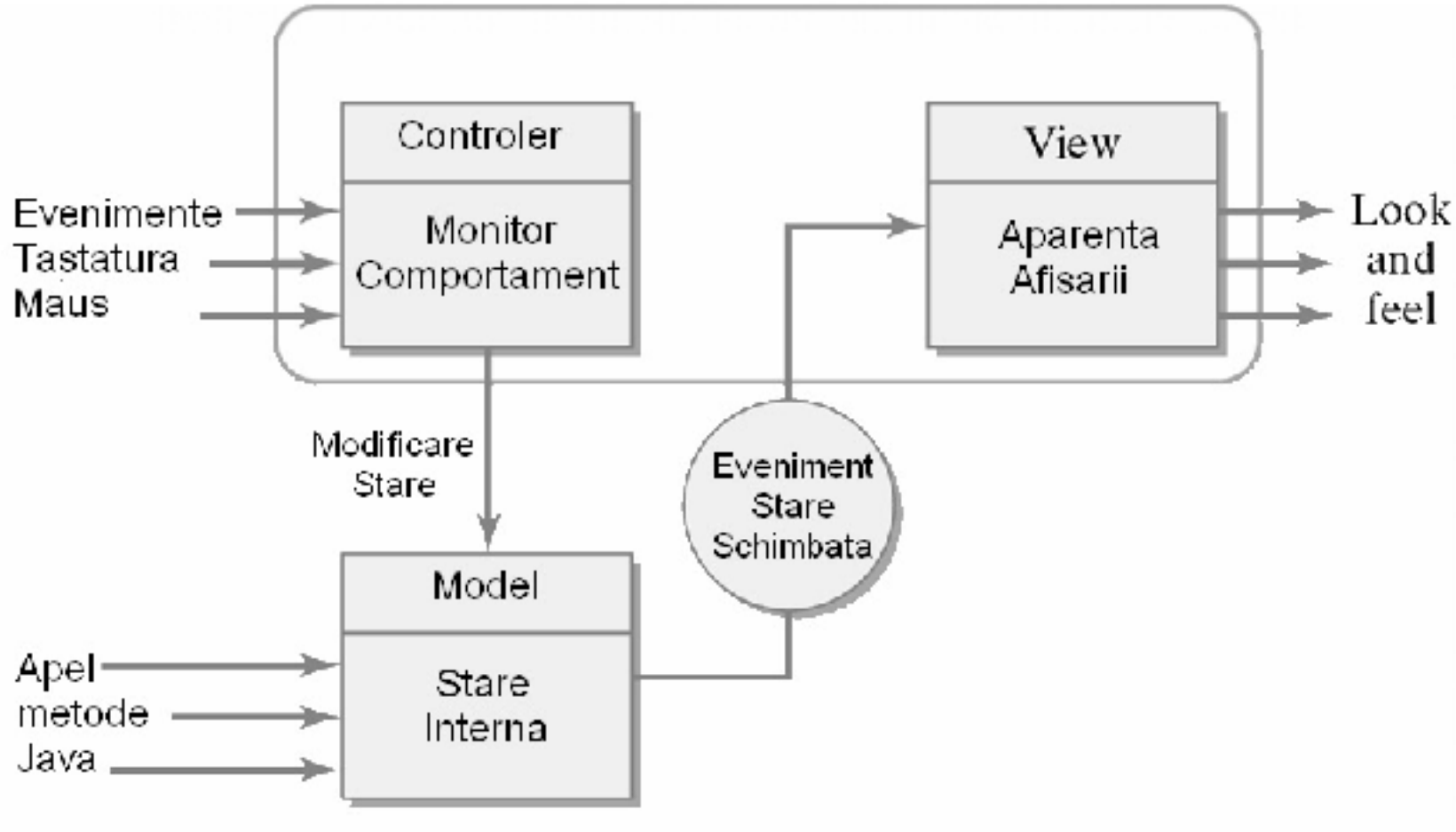
Pick Correlation

- Procesul de selecție a unei ferestre sau aplicații care trebuie sa trateze un eveniment oarecare (deoarece le aparține) se numește corelația de selecție (pick correlation)

Ce sunt widgets?

- Sunt obiectele din cadrul unui GUI orientat obiect.

Model-View-Controller (MVC)



Ce este SDL?

- SDL = Simple DirectMedia Layer
- unde=*www.libsdl.org*
- *alte surse*
  - http://tlahoda.github.io/sdlpp/index.html
  - https://wiki.libsdl.org/
  - https://docs.sdl.com/LiveContent/content/en-US/SDL%20Web-v5

Componente SDL

Componentă	Descriere	Echivalent DirectX
Video	Ascunde accesunl nativ la ecran	DirectDraw
Gestiunea evenimentelor	Ascunde accesul nativ la evenimente	DirectInput
Joystick	Ascunde accesul nativ la maneta de joc	DirectInput
Audio	Ascunde accesul nativ la placa audio	DirectSound
CD-ROM	Accesează direct CD audio	Fără Echivalent (F/E)
Fire de execuție	Funcții helper peste cele native de gestiune a firelor de execuție	F/E
Timere	Funcții helper peste cele native de gestiune a timerelor	F/E

Funcții pentru inițializare	**Componenta care va fi inițializată**
SDL_Init	inițializează una sau mai multe subsiteme SDL
SDL_InitSubSystem	inițializează numai un subsistem anume. POate fi tilizată numai după anterioara.
SDL_Quit	închide toate subsistemele SDL
SDL_Quit_SubSystem	închide numai un subsistem anume
SDL_WasInit	verifică și anunță care subsisteme sunt active
SDL_GetError	raportează ultima eroare internă generată de SDL

Identificator	**Componenta care va fi inițializată**
SDL_INIT_TIMER	acces la timere
SDL_INIT_AUDIO	acces la audio
SDL_INIT_VIDEO	acces la video
SDL_INIT_CDROM	acces la CDROM
SDL_INIT_JOYSTICK	acces la Joystick
SDL_INIT_EVERYTHING	activează întreg framework-ul

```
SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO); //exemplu de utilizare
```

Crearea unei ferestre

```
#include<SDL.h>
SDL_Window* g_pWindow = 0;
SDL_Renderer* g_pRenderer = 0;
int main(int argc, char* args[])
{
if(SDL_Init(SDL_INIT_EVERYTHING) >= 0)
   {  SDL_SetVideoMode(800, 600, 32, SDL_FULLSCREEN);
      g_pWindow = SDL_CreateWindow("Testing", SDL_WINDOWPOS_CENTERED,
                                   SDL_WINDOWPOS_CENTERED, 640, 480, SDL_WINDOW_SHOWN);
      if(g_pWindow != 0)
         { g_pRenderer = SDL_CreateRenderer(g_pWindow, -1, 0); }
   }
else
   { return 1; // eroare }
SDL_SetRenderDrawColor(g_pRenderer, 0, 0, 0, 255);
SDL_RenderClear(g_pRenderer);
SDL_RenderPresent(g_pRenderer);
// astept 5 secunde
SDL_Delay(5000);
// iesire }n mod corect
SDL_DestroyWindow(m_pWindow);
SDL_DestroyRenderer(m_pRenderer);
SDL SDL_Quit();
return 0;
}
```

Setări posibile pentru o fereastră SDL

Flag	Scop
SDL_WINDOW_FULLSCREEN	fereastra va ocupa tot ecranul
SDL_WINDOW_OPENGL	Fereastra poate fi folosită cu un context OpenGL
SDL_WINDOW_SHOWN	Fă vizibilă fereastra
SDL_WINDOW_HIDDEN	Ascunde Fereastra
SDL_WINDOW_BORDERLESS	Elimină marginea implicită a ferestrei
SDL_WINDOW_RESIZABLE	Permite redimensionarea ferestrei
SDL_WINDOW_MINIMIZED	Minimizează fereastra
SDL_WINDOW_MAXIMIZED	Maximizează fereastra
SDL_WINDOW_INPUT_GRABBED	Fereastra preia focus-ul
SDL_WINDOW_INPUT_FOCUS	fereastra are focus-ul de intrare
SDL_WINDOW_MOUSE_FOCUS	fereastra are focus de la șobolan
SDL_WINDOW_FOREIGN	fereastra nu a fost creată cu SDL

Gestiunea evenimentelor în SDL

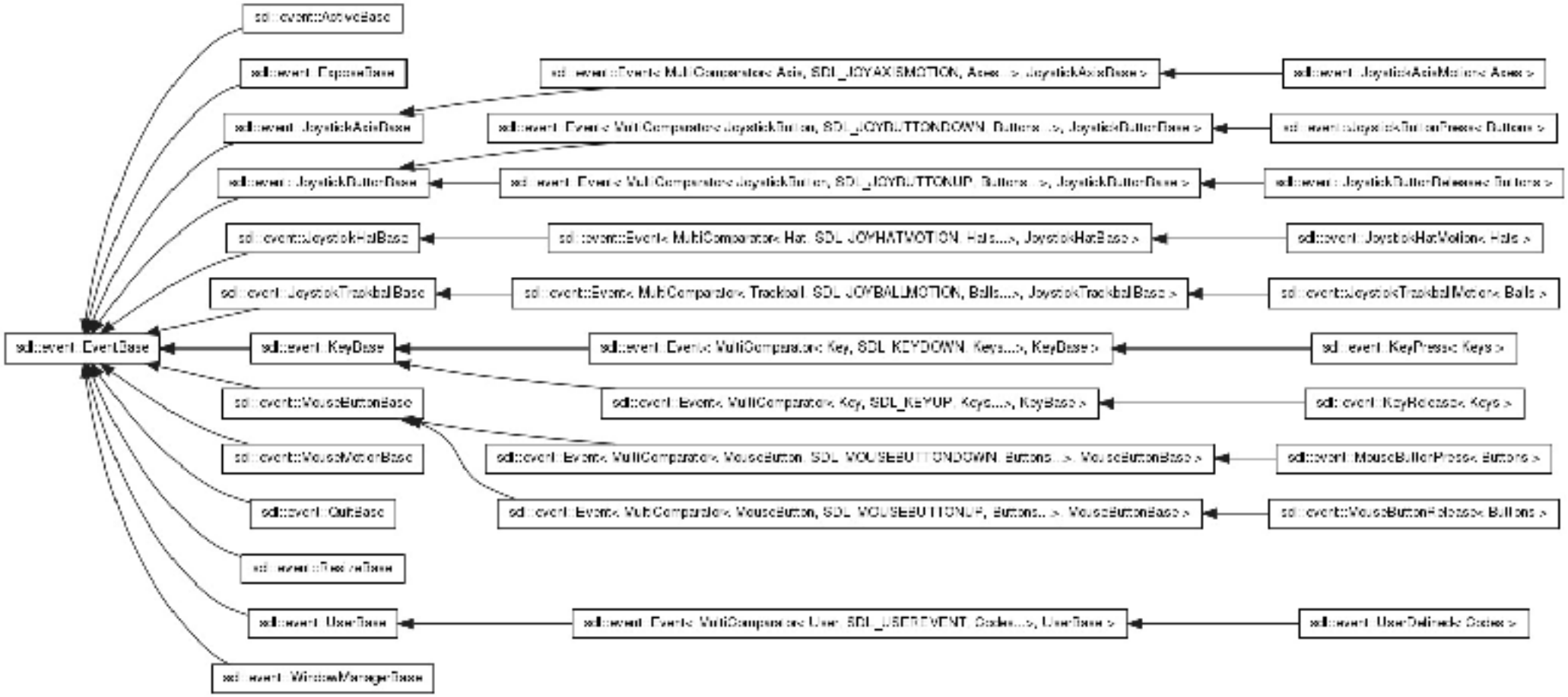
```
#include "SDL.h" class InputHandler {
public:
static InputHandler* Instanced
   {
   if(s_pInstance == 0)
        {  s_pInstance = new InputHandler(); }
    return s_pInstance;
   }
void update();
void clean();
private:
InputHandler();
~InputHandler() {}
static InputHandler* s_pInstance;
};
typedef InputHandler TheInputHandler;
```

Gestiunea șobolanului - evenimente

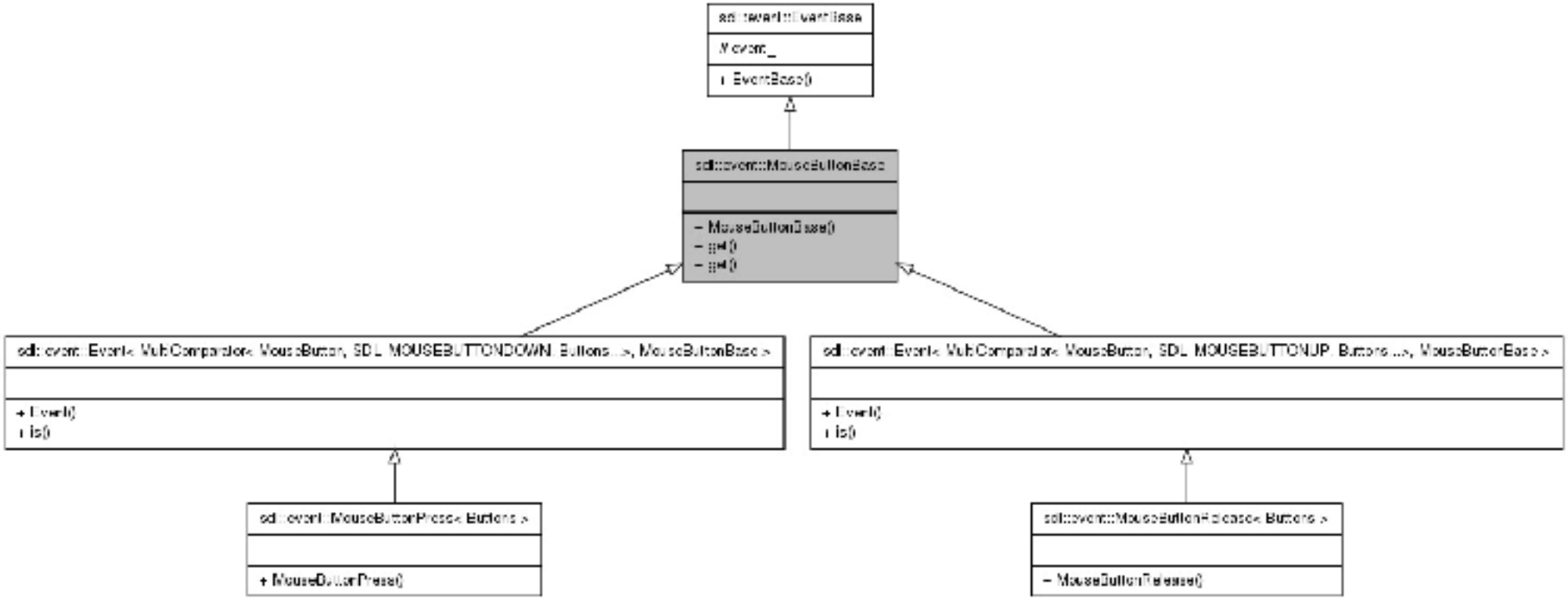
Tip Evenimen SDL	Scop
SDL_MouseButtonEvent	un buton apăsat/eliberat
SDL_MouseMotionEvent	s-a mișcat șobolanul
SDL_MouseWheelEvent	a fost mișcată rotița guzganului

Tip Eveniment SDL	Valoare posibilă
SDL MouseButtonEvent	SDL_MOUSEBUTTONDOWN sau SDL_MOUSEBUTTONUP
SDL_MouseMotionEvent	SDL_MOUSEMOTION
SDL_MouseWheelEvent	SDL_MOUSEWHEEL

Evenimente sdl



Șoricel - Clase



Gestiunea șobolanului - evenimente

```
std::vector<bool> m_mouseButtonStates;
for(int i = 0; i < 3; i++)
   { m_mouseButtonStates.push_back(false); }
```

apoi

```
enum mouse_buttons {
LEFT = 0,
MIDDLE = 1,
RIGHT = 2
};
```

Tratare evenimente urechi șobolan

```
if(event.type == SDL_MOUSEBUTTONDOWN)
{ if(event.button.button == SDL_BUTTON_LEFT)
   { m_mouseButtonStates[LEFT] = true; }
if(event.button.button == SDL_BUTTON_MIDDLE)
   { m_mouseButtonStates[MIDDLE] = true; }
if(event.button.button == SDL_BUTTON_RIGHT)
   { m_mouseButtonStates[RIGHT] = true; }  } //și ...
if(event.type == SDL_MOUSEBUTTONUP)
{
if(event.button.button == SDL_BUTTON_LEFT)
   { m_mouseButtonStates[LEFT] = false; }
if(event.button.button == SDL_BUTTON_MIDDLE)
   { m_mouseButtonStates[MIDDLE] = false; }
if(event.button.button == SDL_BUTTON_RIGHT)
   { m_mouseButtonStates[RIGHT] = false; }
} //si...
bool getMouseButtonState(int buttonNumber)
{ return m_mouseButtonStates[buttonNumber]; } //si...
if(TheInputHandler::Instance()->getMouseButtonState(LEFT))
{ //acțiune în program }
```

Tratarea fugii... după brânză

```
Vector2D* m_mousePosition;//apoi
Vector2D* getMousePosition()
{ return m_mousePosition; } //acum ...
if(event.type == SDL_MOUSEMOTION)
{
m_mousePosition->setX(event.motion.x);
m_mousePosition->setY(event.motion.y);
} //și
Vector2D* vec = TheInputHandler::Instance()->getMousePosition();
m_velocity = (*vec - m_position) / 100; // un calcul de viteză
```

Tratarea tastaturii

```
SDL_GetKeyboardState(int* numkeys) //și
Uint8* m_keystate; //iar apoi
m_keystates = SDL_GetKeyboardState(0);//ne mai...
bool InputHandler::isKeyDown(SDL_Scancode key)
{
if(m_keystates != 0)
  {
  if(m_keystates[key] == 1)
    { return true;}
  else
    {return false;}
  }
return false;
}// si...

if(TheInputHandler::Instance()->isKeyDown(SDL_SCANCODE_RIGHT))
{ m_velocity.setX(2); }
```

Exemplu combinat

```
cvoid InputHandler::update()
{
SDL_Event event; while(SDL_PollEvent(&event))
 {switch (event.type)
  { case SDL_QUIT: TheGame::Instance()->quit(); break;
    case SDL_MOUSEMOTION: onMouseMove(event); break;
    case SDL_MOUSEBUTTONDOWN: onMouseButtonDown(event); break;
    case SDL_MOUSEBUTTONUP: onMouseButtonUp(event); break;
    case SDL_KEYDOWN: onKeyDown(); break;
    case SDL_KEYUP: onKeyUp(); break;
    default: break;}   }
} //si ...
void InputHandler::onMouseButtonDown(SDL_Event& event)
{  if(event.button.button == SDL_BUTTON_LEFT)
      { m_mouseButtonStates[LEFT] = true; }
    if(event.button.button == SDL_BUTTON_MIDDLE)
      { m_mouseButtonStates[MIDDLE] = true; }
    if(event.button.button == SDL_BUTTON_RIGHT)
      { m_mouseButtonStates[RIGHT] = true; }   }
```

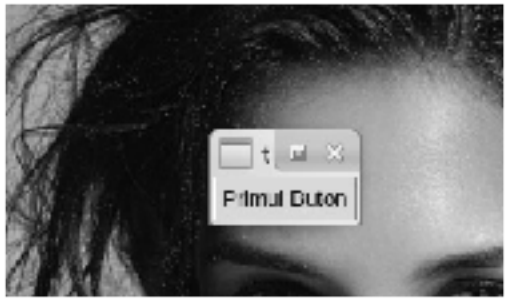
Grafică în Python - TKInter

Primul Buton

```
from tkinter import *
from tkinter import ttk
baza = Tk()
ttk.Button(baza, text="Primul Buton").grid()
baza.mainloop()
```

Pt disperati

instalați pycharm exact ca și intelij
apoi din terminal
sudo apt-get install python3-tk
repornește pycharm
execuția și restul ca la intelij
rezultatul --> pe fruntea fetei

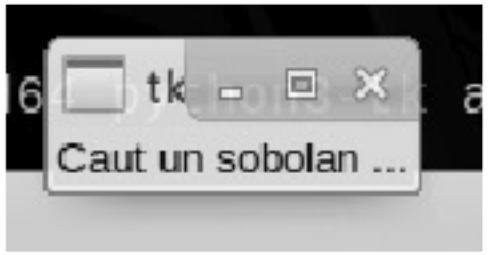


Primul Widget

```
from tkinter import *
from tkinter import ttk

root = Tk()
content = ttk.Frame(root)
button = ttk.Button(content)
root.mainloop()
```

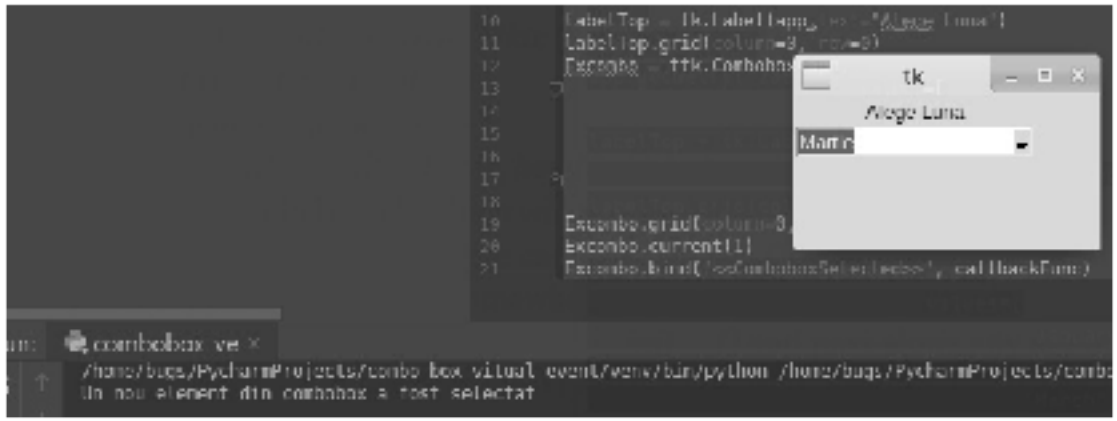
Asocierea unor evenimente

```
from tkinter import *
from tkinter import ttk

root = Tk()

l =ttk.Label(root, text="Caut un sobolan ...")
l.grid()
l.bind('<Enter>', lambda e: l.configure(text='Sobolanul este in interior'))
l.bind('<Leave>', lambda e: l.configure(text='Sobolanul a fugit din zona'))
l.bind('<1>', lambda e: l.configure(text='Sobolanul a miscat din urechea stanga'))
l.bind('<Double-1>', lambda e: l.configure(text='L-am tras de doua ori la rand de
urechi pe sobolan'))
l.bind('<B3-Motion>', lambda e: l.configure(text='Urechea dreapta a fost folosita
pentru o incercare de mutare la %d,%d' %(e.x, e.y)))
root.mainloop()
```



Evenimente virtuale

```
import tkinter as tk
from tkinter import ttk
def callbackFunc(event):
    print("Un nou element din combobox a fost selectat")
app = tk.Tk()
app.geometry('200x100')
labelTop = tk.Label(app,text="Alege Luna")
labelTop.grid(column=0, row=0)
Excombo = ttk.Combobox(app,
                       values=[
                           "Ianuarie",
                           "Februarie",
                           "Martie",
                           "Aprilie"])
Excombo.grid(column=0, row=1)
Excombo.current(1)
Excombo.bind("<<ComboboxSelected>>", callbackFunc)
app.mainloop()
```

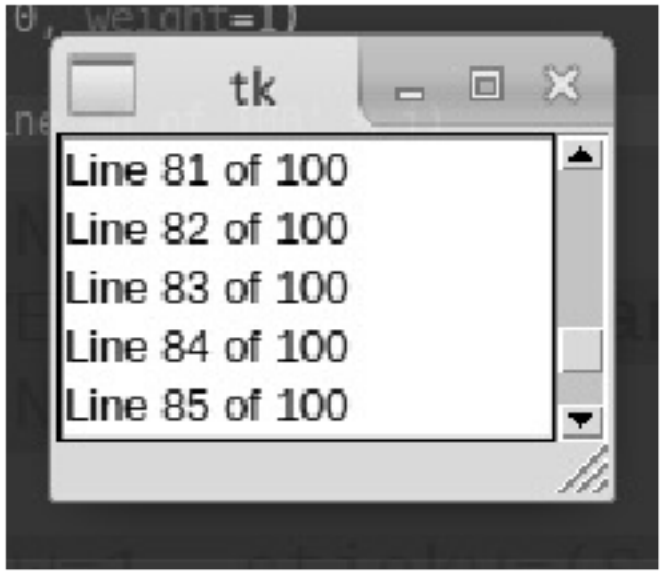


Organizarea matricială a widgets

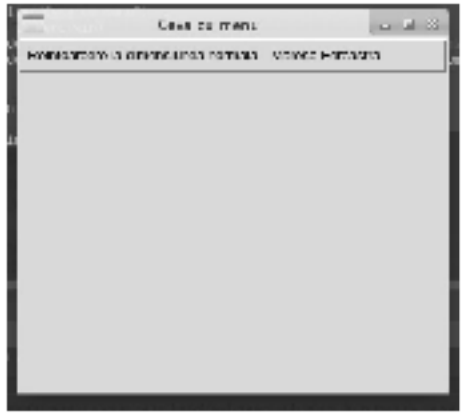
```
from tkinter import *
from tkinter import ttk
root = Tk()
content = ttk.Frame(root)
frame = ttk.Frame(content, borderwidth=5, relief="sunken", width=200,height=100)
namelbl = ttk.Label(content, text="Name")
name = ttk.Entry(content)
onevar = BooleanVar()
twovar = BooleanVar()
threevar = BooleanVar()
onevar.set(True)
twovar.set(False)
threevar.set(True)
one = ttk.Checkbutton(content, text="One", variable=onevar, onvalue=True)
two = ttk.Checkbutton(content, text="Two", variable=twovar, onvalue=True)
three = ttk.Checkbutton(content, text="Three", variable=threevar, onvalue=True)
ok = ttk.Button(content, text="Okay")
cancel = ttk.Button(content, text="Cancel")
content.grid(column=0, row=0)
frame.grid(column=0, row=0, columnspan=3, rowspan=2)
namelbl.grid(column=3, row=0, columnspan=2)
name.grid(column=3, row=1, columnspan=2)
one.grid(column=0, row=3)
two.grid(column=1, row=3)
three.grid(column=2, row=3)
ok.grid(column=3, row=3)
cancel.grid(column=4, row=3)
root.mainloop()
```

O bară pentru derulare

```
from tkinter import *
from tkinter import ttk
root = Tk()
l = Listbox(root, height=5)
l.grid(column=0, row=0, sticky=(N,W,E,S))
s = ttk.Scrollbar(root, orient=VERTICAL, command=l.yview)
s.grid(column=1, row=0, sticky=(N,S))
l['yscrollcommand'] = s.set
ttk.Sizegrip().grid(column=1, row=1, sticky=(S,E))
root.grid_columnconfigure(0, weight=1)
root.grid_rowconfigure(0, weight=1)
for i in range(1,101):
    l.insert('end', 'Line %d of 100' % i)
root.mainloop()
```



Creare Menu-uri simple cu comenzi directe



```
import tkinter as tk

win = tk.Tk()

win.geometry('400x300')

win.title("Ceva cu menu")

menu = tk.Menu(win)

menu.add_command(label='Reintoarcere la dimensiunea normala',
command=lambda: win.geometry('400x300') + win.title("Dimensiune
400x300"))

menu.add_command(label='Maresc Fereastra', command=lambda:
win.geometry('600x600') + win.title("Dimensiune 600x600"))

win.configure(menu=menu)

win.mainloop()
```