****

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**Calancea Catalin**

**MI-222**

**Raport**

**pentru lucrarea de laborator Nr.6**

***la cursul de “Programarea Orientată pe Obiect”***

Verificat:

lector universitar

Buldumac Oleg

**Chișinău – 20****23**

**CUPRINS**

[1.INTRODUCERE 3](#_Toc149568596)

[2.SARCINA 4](#_Toc149568597)

[3.CONCLUZIA 5](#_Toc149568598)

[4.WEBOGRAFIE 6](#_Toc149568599)

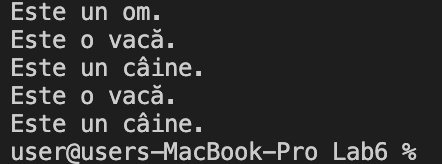
[5.ANEXĂ 7](#_Toc149568600)

# 1.INTRODUCERE

# Polimorfismul reprezintă capacitatea unui obiect de a se comporta în moduri multiple, în funcție de contextul în care este utilizat. Vom analiza modalitățile prin care polimorfismul poate aduce flexibilitate și extensibilitate în proiectarea aplicațiilor.Principiul legăturii întârziate sau întârzierea legăturii este un concept fundamental în POO, referindu-se la momentul în care se stabilește legătura între un nume și o implementare. Vom explora beneficiile legăturii întârziate și modul în care aceasta contribuie la creșterea modularității.Funcțiile virtuale permit suprascrierea metodelor în clasele derivate, facilitând astfel implementarea polimorfismului. Vom analiza utilizarea funcțiilor virtuale pentru a obține comportamente specifice în clasele moștenite. Polimorfismul ad-hoc se referă la capacitatea de a trata obiecte diferite într-un mod uniform, indiferent de tipul lor. Vom explora modalitățile în care acest tip de polimorfism aduce un plus de flexibilitate în proiectare și implementare.Vom implementa și analiza funcții virtuale în cadrul unor scenarii practice, evaluând modul în care acestea pot influența structura și comportamentul sistemelor software.Clasele abstracte oferă un nivel suplimentar de abstractizare, permițând definirea de interfețe și specificații fără a furniza implementări concrete. Vom explora modalitățile în care clasele abstracte pot servi drept schelete pentru alte clase derivate.

# 2.SARCINA

Creaţi clasa abstractă de bază Mammal – mamifere cu funcţia virtuală - descrierea. Animal şi Human. Pentru animal determinaţi clasele derivate Dog – câinele şi Cow – vaca, în care funcţia se predefineşte.



# 

# 3.CONCLUZIA

În urma explorării și implementării conceptelor cheie ale programării orientate pe obiect în cadrul acestei lucrări de laborator, putem trage mai multe concluzii relevante pentru dezvoltarea software-ului modular și eficient. Principalele aspecte pe care le-am abordat includ studierea polimorfismului, a principiilor legăturii întârziate, a funcțiilor virtuale, a polimorfismului ad-hoc, a realizării funcțiilor virtuale și a claselor abstracte.

Polimorfismul, atunci când este implementat corespunzător, aduce o flexibilitate semnificativă în dezvoltarea aplicațiilor. Capacitatea de a manipula obiecte într-un mod uniform, indiferent de tipul lor concret, deschide uși pentru extindere și adaptabilitate în proiectare.

Implementarea legăturii întârziate aduce beneficii semnificative în ceea ce privește modularitatea. Acest principiu permite o decuplare mai mare între modulele software, facilitând astfel întreținerea și extinderea ulterioară a codului.

Funcțiile virtuale oferă posibilitatea de a suprascrie comportamentul în clasele derivate, oferind astfel o modalitate puternică de implementare a polimorfismului. Această abordare permite adaptarea flexibilă a funcționalității la nevoile specifice ale fiecărui tip de obiect.

Clasele Abstracte și Abstracția de Interfețe:

Clasele abstracte furnizează un nivel suplimentar de abstractizare, permițând definirea de interfețe fără a specifica implementări concrete. Această abordare susține proiectarea modulară și se integrează bine în contextul dezvoltării bazate pe interfețe clare și specificații.

În final, această lucrare de laborator nu doar a oferit o înțelegere practică a conceptelor de bază ale programării orientate pe obiect, ci și a evidențiat importanța aplicării acestora în dezvoltarea software-ului modern. Cunoștințele acumulate vor servi ca temelie solidă pentru abordarea eficientă a problemelor complexe și pentru dezvoltarea de soluții software robuste și ușor de întreținut.

# 4.BIBLIOGRAFIE

1. Schildt, H. (2011). "C++: The Complete Reference." McGraw-Hill.
2. Eckel, B. (2003). "Thinking in C++." Prentice Hall.
3. Meyer, B. (1997). "Object-Oriented Software Construction." Prentice Hall.
4. C++ Standard Committee. (2017). "Working Draft, Standard for Programming Language C++." Disponibil online: https://github.com/cplusplus/draft.

# 5.ANEXĂ

Codul sursa :

Sarcina

<https://github.com/KetSchnaider/Anul2/blob/main/POO/Lab6/lab6.cpp>