https://github.com/alina-puscasu/pao_lab_2022

# Programare avansata pe obiecte – laborator 3

***Alina Puscasu*** **-** *alina.puscasu@endava.com*

## Mostenirea

- Modeleaza relatia “is-a”, indica faptul ca o clasa este derivata dintr-o clasa de baza
- Mai exista si relatii de tip “has-a”, indica faptul ca o clasa container are o referinta la alta clasa continuta in ea

## Polimorfism

- Se refera la proprietatea obiectelor de a avea mai multe forme. Orice obiect poate fi referit prin tipul sau real sau prin tipul superclasei (tipul declarat)

1. **Supraincarcare** (tipul parametric) - **overloading** - putem defini o metoda cu acelasi nume dar care difera prin nr/tipul de parametri
    a. Acest tip de polimorfism se mai numeste si **compile time polymorphism** sau **static binding** sau **early binding**

2. **Supradefinire/suprascriere** – **overriding** - clasele derivate definesc o metoda din clasa de baza (cu **aceeasi semnatura**!) careia ii schimba comportamentul
    a. Acest tip de polimorfism se mai numeste si **runtime polymorphism** sau **dynamic binding** sau **late binding**
    b. Nu putem suprascrie o metoda static/final sau privata
    c. Metoda suprascrisa nu poate un avea un modificator de access care sa ii permita o vizibilitate mai mica. De ex, daca in clasa parinte avem o metoda protected in copil aceasta poate deveni publica, dar nu privata.

## Abstract

- Cuvantul cheie **abstract**
- O clasa care contine cel putin o metoda abstracta trebuie sa fie abstracta
- O clasa abstracta nu poate fi instantiata
- Orice clasa care va extinde clasa abstracta trebuie sa ofere implementari pentru metodele abstracte, daca nu o face pentru toate metodele abstracte, este obligatoriu sa fie declarata si ea abstract

https://github.com/alina-puscasu/pao_lab_2022

## Exercitiu

Proiectati o clasa CandyBox, care va continue campurile: flavor (String), origin (String) cu modificatorii de access corespunzatori. Clasa va avea de asemenea:

a. Constructor fara parametri
b. Constructor care va initializa **toate** campurile
c. O metoda abstracta getVolume()
d. Suprascrierea metodei toString() – puteti folosi IDE-ul pentru generare automata

Din ea derivati clasele Merci, Lindt, Milka. Pentru un design diferit, cutiile au diverse forme:

a. Lindt va fi un paralelipiped si va contine length, width si height
b. Milka va fi un cilindru cu campurile radius si height
c. Merci va fi un cub si va contine campul length

Cele trei clase vor avea de asemenea:

a. Membrii mentionati mai sus private si getters, setters pentru ei (encapsulare)
b. Constructor fara parametri
c. Constructor care initializeaza toti membrii claselor (inclusiv membrii mosteniti)
d. Suprascrierea metodei getVolume()
e. Suprascrierea metodei toString() care sa returneze un mesaj de genul: “The ” + origin + “ “ + flavor + “ has volume “ + volume.

Oferiti posibilitatea de a verifica egalitatea obiectelor din fiecare tip (adaugati metoda equals() dupa cum este nevoie). Justificati criteriul de echivalenta ales.

Creati o clasa CandyBag care va contine un array cu cutii din fiecare tip.

Creati clasa Area care va contine un obiect de tip CandyBag, un camp number (int) si unul street (String). Clasa va contine:

a. Constructor fara parametri
b. Constructor care initializeaza atributele
c. O metoda printAddress() care va afisa adresa completa si continutul CandyBag (parcurgeti array-ul si apelati metoda toString() pentru elementele sale)

Creati clasa Shop pentru a crea 2 obiecte de tipul CandyBag. Un obiect va continue 3 cutii de Milka, celalalt va contine cate o cutie din fiecare tip de ciocolata (Milka, Lindt, Merci).

Aceste obiecte CandyBag sunt folosite apoi pentru a fi apoi trimise la diverse adrese: primul va fi trimis la strada “Bulevardul Iuliu Maniu” numarul 10, al doilea la adresa “Str Brasov” numarul 25. In clasa Shop dorim sa printam pentru fiecare dintre cele 2 obiecte atat adresa, cat si continutul (inclusiv volumul cutiilor).