POO-Rezumat

Programarea orientată obiect – POO (programe cu noi tipuri ce integrează atât datele, cât şi metodele asociate creării, prelucrării şi distrugerii acestor date).

Principiile fundamentale in OOP:

1. Abstractizarea - principiu care permite identificarea caracteristicilor şi comportamentului obiectelor ce ţin nemijlocit de domeniul problemei. Rezultatul este un model. În urma abstractizării, entităţilor din domeniul problemei se definesc prin clase.

2. Incapsularea – numită şi ascunderea de informaţii, este caracterizată prin 2 aspecte:

a. Gruparea comportamentelor şi caracteristicilor într-un tip abstract de date

b. Definirea nivelului de acces la datele unui obiect

Caracteristici incapsulare: Câmpurile sunt mereu private, Constructorii sunt mai mereu publici, membrii interfeţei sunt mereu publici, dar nu explicit, metodele non-interfaţă sunt private/protected.

Beneficiile incapsularii: Asigură faptul că schimbarile structurale rămân local. Schimbările interne ale clasei nu afectează codul extern, iar prin ascunderea detaliilor de implementare, se reduce complexitatea.

3. Moştenirea – organizează şi facilitează polimorfismul şi încapsularea permiţând definirea si crearea unor clase specializate plecând de la clase (generale) care sunt deja definite - acestea pot împărtăşi (şi extinde) comportamentul lor fără a fi nevoie de redefinirea aceluiaşi comportament.

Derivarea claselor (moştenire): Prin utilizarea moştenirii se poate defini o clasă generală care defineşte trăsături comune la un ansamblu de obiecte. O clasă abstractă (**abstract**) trebuie obligatoriu derivată, deoarece direct din ea nu se pot obţine obiecte prin operaţia de instanţiere, în timp ce o clasă sigilată (**sealed**) nu mai poate fi derivată (e un fel de terminal în ierarhia claselor).

Interfeţele sunt foarte importante în programarea orientată pe obiecte, deoarece permit utilizarea polimorfismului într-un sens mai extins. O **interfaţă** este o componentă a aplicaţiei, asemănătoare unei clase, care declară prin membrii săi (metode, proprietăţi, evenimente şi indexatori) un „comportament" unitar aplicabil mai multor clase, comportament care nu se poate defini prin ierarhia de clase a aplicaţiei.

Virtual, override: O clasă declarată virtuală implică faptul că o metodă implementată în ea poate fi redefinită în clasele derivate. În clasele derivate se va folosi cuvântul cheie **override** pentru redefinirea metodei virtuale sau abstracte din clasa de bază.

4. Polimorfismul - posibilitatea mai multor obiecte dintr-o ierarhie de clase de a utiliza denumiri de metode cu acelaşi nume dar, cu un comportament diferit.

Polimorfismul, care trebuie să fie abstract, se manifestă în lucrul cu obiecte din clase aparţinând unei ierarhii de clase, unde, prin redefinirea unor date sau metode, se obţin membri diferiţi având însă acelaşi nume.

Interfete vs Clase abstracte:

-Interfetele, spre deosebire de Clasele abstracte nu contin metode cu implementare, toate metodele acestora sunt abstracte, membrii sunt publici si nu definesc constructori, constante si campuri.

-Metodele Abstracte sunt si virtuale, sunt facute pentru a fi modificate mai tarziu. Metodele abstracte si membrii intrfetelor sunt virtuali puri.

-Se pot suprascrie doar metodele virtuale, abstracte sau averride.

Clasele: Definesc structuri specifice pentru obiecte. **Obiecte:** Instanţe particulare ale unei clase. **This**, reprezintă o referiţă la obiectul curent.

Altfel spus clasele modelează obiectele din lumea reală şi defineşte:

- **date (variabile): constante, campuri,**
- **funcţii:** *Constructori, Destructori, Metode, Proprietăţi, Evenimente, Indexatori, Operatori*

Atât datele cât şi funcţiile pot avea ca modificatori de acces (restrictionează accesul la membrii, suportă principiul incapsulării):

- **public -** Membrul este accesibil de oriunde;
- **internal -** Membrul este accesibil doar în assembly-ul curent;
- **protected -** Membrul este accesibil oricărui membru al clasei care-l conţine şi a claselor derivate;
- **private-** Modificator implicit. Accesibil, permis doar pentru clasa care-l conţine;

Constructorii sunt funcţii care folosesc la iniţializarea unei instanţe a clasei. Constructorii au acelaşi nume cu al clasei. Constructorul poate avea un modificator de acces şi nu returnează nimic. *Observaţie***:** Constructorii nu pot fi moşteniţi.

Metoda este un membru al unei clase care implementează o acţiune. Metoda poate admite parametri şi returna valori. Tipul returnat de către o metodă poate fi unul predefinit ***(int, bool*** etc.) sau de tip obiect (**class**). În cazul în care metoda nu returnează nimic, tipul este ***void***. Metodele pot fi supradefinite (supraîncărcate), adică se pot defini mai multe metode, care să poarte acelaşi nume, dar să difere prin numărul şi tipul de parametri. Valoarea returnată de către o metodă nu poate să fie luată în considerare în cazul supradefinirii.

Proprietatea este un membru al clasei care ne permite să accedem sau să modificăm caracteristicile unui obiect sau al clasei. Accesorul **get (***tipul valorii returnate***)** corespunde unei metode fără parametri, care returnează o valoare de tipul proprietăţii. Accesorul **set***(tipul valorii atribuite)* corespunde unei metode cu un singur parametru, de tipul proprietăţii şi tip de retur **void**.