UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

ALGORITMI ŞI PROGRAMARE

Cursul 5

Principii de dezvoltare

Programare Orientată Obiect

Clase în Python

Ionescu Vlad
vlad.ionescu@ubbcluj.ro

Feedback săptămâna 4

- Întrebări
- Feedback
- Despre test

Organizarea aplicației – **SRP**

- În funcții, module și pachete
- **Responsabilitate**:
  - un motiv pentru a schimba ceva
  - Unei funcții: efectuarea unui singur calcul
  - Unui modul: responsabilitățile tuturor funcțiilor din modul

- **Principiul Singurei Responsabilități** (Single Responsibility Principle – SRP)
  - O funcție/un modul trebuie să aibă o singură responsabilitate: un singur motiv de schimbare

Organizarea aplicației – **SRP**

- Încălcarea SRP duce la:
  - Dificultăți de înțelegere și utilizare
  - Imposibilitatea sau îngreunarea testării
  - Imposibilitatea sau îngreunarea refolosirii
  - Dificultăți la întreținere și evoluție

Organizarea aplicației – **SoC**

- **Separation of Concerns**
  - Principiul separării grijilor
  - Aplicațiile trebuie împărțite în module ale căror funcționalități se suprapun cât mai puțin

- Discuție: SoC vs SRP

Organizarea aplicației – **Coupling**

- **Dependencies**
  - Dependențe
  - Când o entitate depinde de alta
    - La nivel de funcții: o funcție care apelează altă funcție
    - La nivel de module: o funcție dintr-un modul apelează o funcție din alt modul

- **Coupling** (cuplare)
  - Măsoară nivelul de legături dintre module (dependențele)
  - Dependențe multe => cuplare ridicată: dificil de întreținut, refolosit
  - Cuplare scăzută => modificările afectează o parte izolată a aplicației

Organizarea aplicației – **Cohesion**

- **Cohesion** (coeziune)
  - Măsoară gradul de relaționare dintre entități
  - Coeziune ridicată (high cohesion): entitățile implementează responsabilități înrudite
  - Coeziune scăzută (low cohesion): entitățile implementează responsabilități între care nu există o legătură conceptuală
  - O entitate puternic coezivă trebuie să realizeze o singură sarcină și să introducă puține dependențe

- Ne dorim să obținem **low coupling, high cohesion**!

Layered Architecture (arhitectura stratificată)

- Structurarea aplicației trebuie să aibă în vedere:
  - Minimizarea cuplării între module (modulele nu trebuie să cunoască detalii despre alte module, astfel schimbările ulterioare sunt mai ușor de implementat)
- Maximizarea coeziunii pentru module: conținutul unui modul să izoleze un concept bine definit
- Arhitectură stratificată
  - Un șablon arhitectural care permite dezvoltarea de sisteme flexibile
  - Componentele au un grad ridicat de independență
  - Fiecare strat comunică doar cu startul imediat următor (depinde doar de stratul imediat următor)
  - Fiecare strat are o interfață bine definită (se ascund detaliile), interfață folosită de stratul imediat superior

Layered Architecture (arhitectura stratificată)

- Nivel prezentare (User interface / Presentation)
  - Implementează interfața utilizator
- Nivel logic (Domain, Application Logic / Service)
  - Oferă funcții determinate de cazurile de utilizare
  - Implementeaza concepte din domeniul aplicației
- Infrastructură
  - Funcții/module/clase generale, utilitare
- Coordonatorul aplicației (Application coordinator, "Main")
  - Asamblează și pornește aplicația

Programarea Orientată Obiect (OOP)

- Este o paradigmă de programare
- Oferă o abstractizare puternică și flexibilă
- Programatorul poate exprima soluția într-un mod mai natural, apropiat de viața reală, nu de detaliile tehnice ale calculatorului
- Programul va lucra cu obiecte
- Obiectele vor interacționa pentru rezolvarea problemei
- Clase = tipuri de date
- Obiecte = instanțe ale unei clase

Clase – **Exemplu**

- Clasa:
  - Definește un nou tip de date (domeniu + operații)
  - Formată din:
    - Câmpuri (atribute): descriu caracteristici
    - Metode (operații): descriu comportament
- O clasă este un șablon pentru obiectele create pe baza ei.
- Introduce un nou namespace.

Obiecte – **Exemplu**

- Obiectele:
  - Au o stare: valorile câmpurilor
  - Folosind metode, le putem modifica starea
  - Sunt instanțe ale unei clase
  - În general, se comportă la fel ca celelalte obiecte din Python

Metode – **Exemplu**

- Sunt funcții care se definesc în interiorul unei clase
- Două tipuri:
  - Metode de instanță (instance methods): care se pot apela doar prin intermediul unui obiect al clasei
  - Metode statice (de clasă): care se pot apela fără un obiect al clasei
- Au ca prim parametru instanța curentă a clasei (instance methods)

Metode speciale

- Apelate automat în diverse contexte
- Supraîncărcarea operatorilor:
  - __str__ - reprezentarea ca string
  - __lt__, __le__, __gt__, __ge__ - comparații
  - __eq__ - egalitate
  - __add__, __mul__, etc. – operații matematice
  - __setitem__, __getitem__, __len__, __getslice__ - pentru obiecte tip colecții
  - Altele, vezi documentația

Questions and Answers

Q & A