Programare orientată pe obiecte

Laboratorul 3

Mihai Nan
mihai.nan.cti@gmail.com

Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2015 - 2016

1 Clase si Obiecte in Java

1.1 Introducere

Presupunem ca dorim ***sa descriem***, uzitand un limbaj de programare, un obiect carte. In general, o carte poate fi caracterizata prin titlu, autor si editura. Cum am putea realiza aceasta descriere formala?

Daca descriem acest obiect, tip abstract de date, intr-un limbaj de programare structural, spre exemplu limbajul C, atunci vom crea, ca mai jos, o structura Carte impreuna cu o serie de functii cuplate de aceasta structura. Cuplajul este realizat prin faptul ca orice functie care opereaza asupra unei carti contine in lista sa de parametri o variabila de tip Carte.

Cod sursa C

```c
typedef struct carte {
  char *titlu, *autor;
  int nr_pagini;
}*Carte;

void initializare(Carte this, char* titlu, char* autor,
    int nr_pagini) {
  this->titlu = strdup(titlu);
  this->autor = strdup(autor);
  this->nr_pagini = nr_pagini;
}

void afisare(Carte this) {
  printf("%s, %s - %d\n", this->autor, this->titlu,
      this->nr_pagini);
}
```

Daca modelam acest obiect intr-un limbaj orientat pe obiecte (in acest caz, Java), atunci vom crea o ***clasa*** Carte ca mai jos.

Se poate observa cu usurinta, in cadrul exemplului de mai jos, ca atat datele cat si ***metodele*** (functiile) care opereaza asupra acestora se gasesc in interiorul aceleiasi entitati, numita ***clasa***. Evident, in codul din exemplu sunt folosite concepte care nu au fost inca explicate, dar cunoasterea si intelegerea reprezinta scopul principal al acestui laborator.

Cod sursa Java

```java
class Carte {
    String nume, autor;
    int nr_pagini;

    public Carte(String nume, String autor, int nr_pagini) {
        this.nume = nume;
        this.autor = autor;
        this.nr_pagini = nr_pagini;
    }

    public Carte() {
        this("Enigma Otiliei", "George Calinescu", 423);
    }

    public String toString() {
        String result = "";
        result += this.autor + ", " + this.nume;
        result += " - " + this.nr_pagini;
        return result;
    }

    public static void main(String args[]) {
        Carte carte;
        carte = new Carte("Poezii", "Mihai Eminescu", 256);
        System.out.println(carte.toString());
    }
}
```

1.2 Clase si Obiecte

1.2.1 Ce este un obiect? Ce este o clasa?

Atunci cand un producator creaza un produs, mai intai acesta specifica toate caracteristicile produsului intr-un document de specificatii, iar pe baza acelui document se creaza fizic produsul. De exemplu, calculatorul este un produs creat pe baza unui astfel de document de specificatii. La fel stau lucrurile si intr-un program orientat pe obiecte: mai intai se creaza ***clasa*** obiectului (documentul de specificatii) care inglobeaza toate caracteristicile unui ***obiect*** (instanta a clasei), dupa care, pe baza acesteia, se creaza (instantiaza) obiectul in memorie.

In general, putem spune ca o clasa furnizeaza un sablon ce spcifica datele si operatiile ce apartin obiectelor create pe baza sablonului - in documentul de specificatii pentru un calculator se mentioneaza ca acesta are un monitor si o serie de periferice.

⚠ IMPORTANT !

⚠ Programarea orientata pe obiecte este o metoda de implementare a programelor in care acestea sunt organizate ca si colectii de obiecte care coorpereaza intre ele, fiecare obiect reprezentand instanta unei clase.

1.2.2 Definirea unei clase

Din cele de mai sus deducem ca o clasa descrie un obiect, in general, un nou tip de data. Intr-o ***clasa*** gasim ***date*** si ***metode*** ce opereaza asupra datelor respective.

Pentru a defini o clasa, trebuie folosit cuvantul cheie ***class*** urmat de numele clasei.

⚠ IMPORTANT !

⚠ O ***metoda*** nu poate fi definita in afara unei ***clase***.

Datele **nume**, **autor**, **nr_pagini** definite in clasa **Carte** se numesc ***atribute***, ***date-membru***, ***variabile-membru*** sau ***campuri***, iar operatiile **toString** si **main** se numesc ***metode***.

Fiecare clasa are un set de ***constructori*** care se ocupa cu ***instantierea*** (initializarea) obiectelor nou create. De exemplu, clasa **Carte** are doi constructori: unul cu trei parametri si unul fara parametri care il apeleaza pe cel cu trei parametri.

1.2.3 Crearea unui obiect

Spuneam mai sus ca un obiect reprezinta o instanta a unei clasa. In Java, instantierea sau crearea unui obiect se face dinamic, folosind cuvantul cheie ***new*** si are ca efect crearea efectiva a obiectului cu alocarea spatiului de memorie corespunzator.

Asa cum fiecare calculator construit pe baza documentului de specificatii are propriile componente, fiecare obiect de tip **Calculator** are propriile sale atribute.

Initializarea se realizeaza prin intermediul constructorilor clasei respective. Initializarea este, de fapt, parte integranta a procesului de instantiere, in sensul ca imediat dupa alocarea memoriei ca efect al operatorului ***new*** este apelat constructorul specificat. Parantezele rotunde dupa numele clasei indica faptul ca acolo este de fapt un apel la unul din constructorii clasei si nu simpla specificare a numelui clasei.

In Java, este posibila si crearea unor ***obiecte anonime***, care servesc doar pentru initializarea altor obiecte, caz in care etapa de declarare a referintei obiectului

nu mai este prezenta.

Cod sursa Java

```java
class Point {
    int x, y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

class Dimension {
    int width, height;

    public Dimension(int width, int height) {
        this.width = width;
        this.height = height;
    }
}

class Rectangle {
    Point p;
    Dimension d;

    public Rectangle(Point p, Dimension d) {
        this.p = p;
        this.d = d;
    }

    public static void main(String args[]) {
        Rectangle patrat = new Rectangle(new Point(0, 0),
            new Dimension(10, 10));
    }
}
```

⚠ IMPORTANT !

⚠ Declararea unui obiect nu implica alocarea de spatiu de memorie pentru acel obiect. Alocarea memoriei se face doar la apelul operatorului ***new***.

1.3 Referinte la obiecte

In sectiunea anterioara, am vazut cum se defineste o clasa si cum se creaza un obiect. In aceasta sectiune vom vedea cum putem executa operatiile furnizate de obiecte. Pentru a putea avea acces la operatiile furnizate de catre un obiect, trebuie sa detinem o ***referinta*** spre acel obiect.

Odata un obiect creat, el poate fi folosit in urmatoarele sensuri: aflarea unor informatii despre obiect, schimbarea starii sale sau executarea unor actiuni. Aceste lucruri se realizeaza prin aflarea sau schimbarea valorilor variabilelor sale, respectiv prin apelarea metodelor sale.

Declarea unei referinte numite ***carte*** spre un obiect de tip ***Carte*** se face in felul urmator: ***Carte carte;***.

⚠ IMPORTANT !

⚠ Faptul ca avem la un moment dat o referinta nu implica si existenta unui obiect indicat de acea referinta. Pana in momentul in care referintei nu i se ataseaza un obiect, aceasta nu poate fi folosita.

Observatie

Valoarea ***null***, ce inseamna ***niciun obiect referit***, nu este atribuita automat tuturor variabilelor referinta la declararea lor. Regula este urmatoarea: daca referinta este un membru al unei clase si ea nu este initializata in niciun fel, la instantierea unui obiect al clasei respective referinta va primit implicit valoarea ***null***. Daca insa referinta este o variabila locala ce apartine unei metode, initializarea implicita nu mai functioneaza. De aceea, se recomanda ca programatorul sa realizeze **intotdeauna** o initializare explicita a obiectelor.

Dupa cum am observat in exemplul oferit in prima sectiune, apelul metode ***toString*** nu este ***toString(carte)***, ci ***carte.toString()*** intrucat metoda ***toString*** apartine obiectului referit de ***carte*** - se apeleaza metoda ***toString*** pentru obiectul referit de variabila ***carte*** din fata lui.

Pentru o intelegere mai buna a conceptului de referinta a unui obiect, consideram exemplul de mai jos in care cream doua obiect de tip ***Carte*** precum si trei referinte spre acest tip de obiecte.

Fiecare dintre obiectele ***Carte*** are alocata o zona proprie de memorie, in care sunt stocate valorile campurilor ***nume***, ***autor***, ***nr_pagini***. Ultima referinta definita in exemplul de mai jos, **c3**, va referi si ea exact acelasi obiect ca si **c2**, adica al doilea obiect creat.

⚠ IMPORTANT !

⚠ In cazul unui program, putem avea acces la serviciile puse la dispozitie de un obiect prin intermediul mai multor referinte.

Cod sursa Java

```
class Test {
    public static void main(String args[]) {
        Carte c1 = new Carte("Poezii", "Mihai Eminescu", 326);
        Carte c2 = new Carte("Camil Petrescu", "George
    Calinescu", 426);
        Carte c3 = c2;
        c3.autor = "George Calinescu";
        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
    }
}
```

Atribuirea ***c3 = c2*** nu a facut altceva decat sa ataseze referintei ***c3*** obiectul avand aceeasi identitate ca si cel referit de ***c2***, adica obiectul secund creat.

1.4 Componenta unei clase

Clasele, asa cum am vazut deja, sunt definite folosind cuvantul cheie ***class***. In urmatoarele sectiuni, vom vorbi despre diferite categorii de membri care pot apare in interiorul unei clase.

1.4.1 Constructori

In multe cazuri, atunci cand instantiem un obiect, ar fi folositor ca obiectul sa aiba anumite atribute initializate.

Initializarea atributelor unui obiect se poate face in mod automat, la crearea obiectului, prin intermediul unui ***constructor***. Principalele caracteristici ale unui constructor sunt:

- un constructor are acelasi nume ca si clasa in care este declarat;
- un constructor nu are tip returnat;
- un constructor se apeleaza automat la crearea unui obiect;
- un constructor se executa la crearea obiectului si numai atunci.

⚠ IMPORTANT !

⚠ Daca programatorul nu prevede intr-o clasa niciun constructor, atunci compilatorul va genera pentru clasa respectiva un constructor implicit fara niciun argument si al carui corp de instructiuni este vid.

⚠ IMPORTANT !

⚠ Daca programatorul include intr-o clasa cel putin un constructor, compilatorul nu va mai genera constructorul implicit.

1.4.2 Membri statici

Atunci cand definim o clasa, specificam felul in care obiectele de tipul acelei clase arata si se comporta. Dar pana la crearea efectiva a unui obiect folosind ***new*** nu se aloca nicio zona de memorie pentru atributele definite in cadrul clasei, iar la crearea unui obiect se aloca acestuia memoria necesara pentru fiecare atribut existent in clasa instantiata. Tot pana la crearea efectiva a unui obiect nu putem beneficia de serviciile definite in cadrul unei clase. Ei bine, exista si o exceptie de la regula prezentata anterior - ***membrii statici*** (atribute si metode) ai unei clase. Acesti membri ai unei clase pot fi folositi direct prin intermediul numelui clasei, fara a detine instante a respectivei clase.

⚠ IMPORTANT !

⚠ Un membru static al unei clase caracterizeaza clasa in interiorul careia este definit precum si toate obiectele clasei respective.

Un membru al unei clase (atribut sau metoda) este static daca el este precedat de cuvantul cheie ***static***.

Din interiorul unei metode statice pot fi accesati doar alti membri statici ai clasei in care este definita metoda, accesarea membrilor nestatici ai clasei producand o eroare de compilare.

Trebuie avut in vedere contextul static al metodei ***main***. Dintr-un context static nu se pot apela functii nestatice, in schimb, se pot crea obiecte ale oricarei clase.

1.5 Principii POO

Observatie

Mai multe functii pot avea acelasi nume in acelasi domeniu de definitie, daca se pot diferentia prin numarul sau tipul argumentelor de apel.

1.5.1 Supraincarcarea

In Java, se pot gasi doua sau mai multe metode, in cadrul aceleiasi clase, care sa aiba acelasi nume, atata timp cat parametrii lor sunt diferiti. In acest caz, se spune ca metoda este supraincarcata, iar procedeul se numeste supraincarcarea metodelor.

Pentru o mai buna intelegere a acestui principiu POO, se va oferi, in continuare, un exemplu pentru o metoda care determina maximul.

Cod sursa Java

```
class Test {
    public int maxim(int a, int b) {
        if(a > b) {
            return a;
        } else {
            return b;
        }
    }

    public int maxim(String s1, String s2) {
        if(s1.compareTo(s2) < 0) {
            return 2;
        } else {
            return 1;
        }
    }

    public int maxim(int a, int b, int c) {
        if(maxim(a, b) < c) {
            return c;
        } else {
            return maxim(a, b);
        }
    }
}
```

Un alt exemplu elocvent, pentru acest prinicpiu POO, este operatorul + care executa operatii diferite in cotexte diferite.

2 Probleme laborator

2.1 Probleme standard

Problema 1 - 2 puncte

Definiti o clasa ***Complex*** care modeleaza lucrul cu numere complexe. Membrii acestei clase sunt:

- doua atribuite de tip ***double*** pentru partile reala, respectiv imaginara ale unui numar complex;
- un constructorul cu doi parametri de tip ***double***, pentru setarea celor doua parti ale numarului (reala si imaginara);
- un constructor fara parametri care apeleaza constructorul anterior;
- o metoda, cu un **singur** parametru, de calcul a sumei a doua numere complexe;
- o metoda ***toString*** uzitata pentru afisarea pe ecran a valorii numarului complex;
- o metoda ***equals***, cu un parametru de tip ***Object***, care va returna ***true*** daca numerele comparate sunt egale, respectiv ***false*** in sens contrar;
- o metoda ***main*** pentru testarea functionalitatii clasei.

`Consultati exemplul prezentat la curs!`

Problema 2 - 2 puncte

Un sertar este caracterizat de latime, lungime si inaltime, toate 3 reprezentate prin numere intregi. Un birou are doua sertare si, evident, este caracterizat de latime, lungime si inaltime, de asemenea, fiind numere intregi.

Creati clasele ***Sertar*** si ***Birou*** corespunzatoare specificatiilor de mai sus. Creati pentru fiecare clasa constructorul potrivit astfel incat caracteristicile instantelor sa fie setate la crearea acestora.

Clasa ***Sertar*** contine o metoda ***toString*** al carei apel va returna un ***String*** sub forma *"Sertar " + l + L + H*, unde *l, L, H* sunt valorile corespunzatoare latimii, lungimii si inaltimitii sertarului. Clasa ***Birou*** contine o metoda ***toString*** care ajuta la afisarea tuturor componentelor biroului (va apela metoda ***toString*** pentru fiecare sertar). Creati, intr-o metoda ***main***, un obiect de tip ***Birou***, uzitand obiecte anonime in cadrul instantierii.

```
Patrat p = new Patrat(new Point(0, 0), new Dimension(5, 5));
```

Problema 3 - 2 puncte

Sa se defineasca o clasa ***Stiva*** pentru stive de numere intregi, reprezentate prin vectori intrinseci.

Datele clasei:

- un vector de intregi;
- indicele elementului din varful stivei (ultimul introdus).

Constructorii clasei:

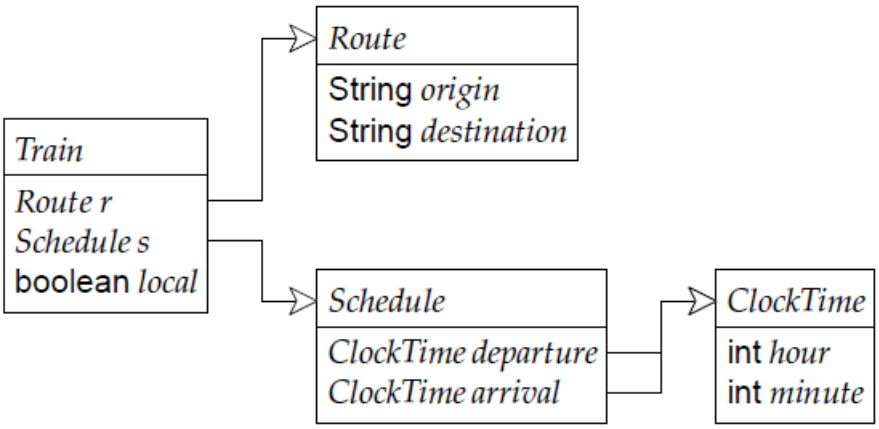
- constructor fara parametri (dimensiunea implicita a stivei fiind 100 de elemente);
- constructor cu un parametru intreg, reprezentand dimensiunea stivei.

Metodele clasei:

- ***void push(int)*** = adauga un intreg in stiva;
- ***int pop()*** = scoate elementul din varful stivei si il returneaza;
- ***boolean isEmpty()*** = verifica daca stiva este goala;
- ***String toString()*** = afiseaza continutul stivei;
- ***void main(String[])*** = metoda statica pentru verificarea operatiilor cu o stiva.

Problema 4 - 2 puncte

Sa se implementeze ierarhia de clase descrisa prin urmatoarea diagrama:



Clasa ***Schedule*** contine o metoda care calculeaza durata calatoriei in minute. Stiind algoritmul de calcul al pretului unui bilet de calatorie, sa se implementeze o metoda, in clasa ***Train***, care calculeaza pretul unui bilet. Valoarea unui bilet este egala cu ***X * durata_calatoriei***, unde ***X*** este egal cu **1** pentru

cursele interne si **2** pentru cursele internationale. Clasa ***Route*** va contine un constructor cu 2 parametri si o metoda care primeste ca parametru un obiect de tip ***Route*** si verifica daca sunt de tip tur - retur cele doua rute, rezultatul fiind de tip ***boolean***. Clasa ***ClockTime*** contine o metoda, cu un parametru de tip ***ClockTime***, si compara doua momente de timp, rezultatul fiind un ***int***.

Adaugati o metoda statica ***main*** pentru testarea claselor implementate, utilizand exemplele oferite. Definiti un constructor potrivit pentru instantierea unui obiect de tip ***Train***, in care sa apelati constructorii definiti in clasele ***Route***, ***Schedule*** si ***ClockTime***.

```
[local] [origin (departure)] -> [destination (arrival)]
true Bucuresti Nord (9:35) -> Constanta (12:02)
true Bucuresti Nord (5:45) -> Iasi (12:49)
false Bucuresti Nord (23:45) -> Sofia (17:00)
```

Problema 5 - **1 punct**

Definiti o clasa executabila ***Numar*** care are ca membru un numar intreg si contine metodele descrise mai jos. Implementati metodele astfel incat fiecare metoda sa efectueze o singura adunare.

Instantiati un obiect de tip ***Numar*** in metoda statica ***main*** si apelati metodele implementate.

Ce principiu POO este evidentiat in acest exercitiu?

```
//returneaza suma dintre nr (membrul clasei) si a
public int suma(int a);
//returneaza suma dintre nr, a si b
public int suma(int a, int b);
//returneaza suma dintre nr, a, b si c
public int suma(int a, int b, int c);
//returneaza suma dintre nr, a, b, c si d
public int suma(int a, int b, int c, int d);
```

Problema 6 - **1 punct**

Implementati clasa ***Punct*** care defineste un punct din spatiul 2D.

Datele clasei (private):

- doua nr. intregi reprezentand cele doua coordonate ale punctului.

Conctructorul clasei:

- un constructor fara parametri care instantiaza punctul O(0, 0).

Metodele clasei

- ***int getX()*** = intoarce abscisa punctului;
- ***void setX(int x)*** = seteaza abscisa punctului;

- ***int getY()*** = intoarce ordonata punctului;
- ***void setY(int y)*** = seteaza ordonata punctului;
- ***String toString()*** = returneaza un String de forma *(x, y)*;
- ***double distance(int, int)*** = calculeaza distanta dintre 2 puncte;
- ***double distance(Punct p1)*** = calculeaza distanta dintre 2 puncte.

Creati o clasa executabila ***Test***, in acelasi pachet cu clasa ***Punct***, care calculeaza distanta dintre punctele **A(1, 2)** si **B(-1, 3)**.

Puteti accesa datele clasei ***Punct*** in metoda ***main*** din clasa ***Test***?

2.2 Problema bonus

Problema 7 - 2 puncte

Sa se defineasca o clasa ***Graph*** care sa descrie un graf ponderat orientat care are nodurile numerotate de la 0.

Datele clasei (private):

- o matrice cu componente de tip int (matricea costurilor);
- o constanta (***Infinit***) avand valoarea 9500;
- numarul de noduri.

Constructorul clasei:

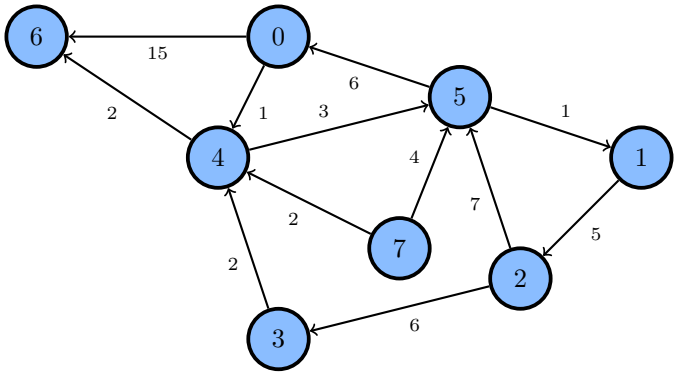
- constructor cu un parametru intreg (numarul de noduri din graf)

Metodele clasei:

- ***int getSize()*** = are ca rezultat numarul de noduri din graf;
- ***void addArc(int v, int w, int cost)*** = adauga un arc la graf (intre v si w, avand costul cost);
- ***boolean isArc(int v, int w)*** = verifica daca exista arc intre v si w in graf;
- ***toString()*** = afisarea grafului (se va alege o varianta intuitiva de afisare a grafului);
- ***int[][] floydWarshall()*** = implementarea algoritmului *Floyd - Warshall* pentru determinarea drumurilor de cost minim in graf;
- ***void main(String[])*** = metoda statica main pentru testarea functionalitatii clasei implementate.

Pentru verificare, se va crea graful din figura de mai jos. Se va afisa graful, uzitand metoda ***toString*** implementata, si se va determina distanta minima dintre doua noduri folosind algoritmul *Floyd - Warshall*.

```
public static final double pi = 3.14;
```



Cod sursa Java

```
public int[][] floydWarshall() {
  int result[][];
  result = new int[this.nrVarfuri][this.nrVarfuri];
  int k, i, j;
  for(i = 1; i <= this.nrVarfuri; i++) {
    for(j = 1; j <= this.nrVarfuri; j++) {
      if(i == j) {
        result[i][j] = 0;
      } else if(this.isArc(i, j)) {
        result[i][j] = this.matrice[i][j];
      } else {
        result[i][j] = Infinit;
      }
    }
  }
  for(k = 1; k <= this.nrVarfuri; k++) {
    for(i = 1; i <= this.nrVarfuri; i++) {
      for(j = 1; j <= this.nrVarfuri; j++) {
        int dist;
        dist = result[i][k] + result[k][j];
        if(result[i][j] > dist) {
          result[i][j] = dist;
        }
      }
    }
  }
  return result;
}
```

3 Interviu

Observatie

Aceasta sectiune este una optionala si incearca sa va familiarizeze cu o serie de intrebari ce pot fi adresate in cadrul unui interviu tehnic. De asemenea, aceasta sectiune poate fi utila si in pregatirea pentru examenul final de la aceasta disciplina.

Intrebari interviu

1. Care este diferenta dintre ***declararea*** unei variabile si ***definirea*** variabilei?
2. Ce reprezinta un ***obiect anonim***?
3. Explicati termenul de ***clasa a obiectului***.
4. Explicati si exemplificati termenul de ***constructor*** in Java.
5. De ce este util sa se ***supraincarce constructorii***?
6. Ce se intampla la apelul unei metode in Java?
7. Cum se transmit parametri unei metode in Java?
8. In ce situatii se obtine un constructor implicit?
9. Care este diferenta dintre o ***metoda polimorfica*** si o ***metoda supraincarcata***?

Feedback

Pentru imbunatatirea constanta a acestui laborator, va rog sa completati formularul de feedback disponibil aici.

De asemenea, va rog sa semnalati orice greseala / neclaritate depistata in laborator pentru a o corecta.

Va multumesc anticipat!