Poo Laboratoare[1]

Contents

[1] http://www.google.ro/

Laborator3

Programare Orientata pe Obiecte: Laborator 3

1 Clase în Java

1.1 Definirea unei clase

```
class Masina
{
   Culoare c;
   int pret;
   public Masina(Culoare c)
   {
      this.c = c;
   }

   public Masina()
   {
      this(new Culoare(‘‘alb’’));
   }

   public void setPret(int pret)
   {
      this.pret = pret;
   }

   public static void main(String argv[])
   {
      Masina m = new Masina();
      m.setPret(20);
   }
}
```

1.2 Principii POO

1.2.1 Polimorfismul

După cum se observă în clasa de mai sus avem două funcții cu același nume, singura diferență fiind numărul de argumente. Clasa *Masina* poate fi instanțiată în ambele moduri:

- new Masina(new Culoare(“negru”)) : ceea ce înseamnă că se apeleaza constructorul care conține un argument de tip Culoare. Presupunem că există undeva definită clasa *Culoare* care primește în constructor un nume de culoare.
- new Masina() : care apelează constructorul cu un argument, și instanțiază o masină de culoare albă.

1.2.2 Supraîncărcarea

Operatorul '+' execută operații diferite în contexte diferite:

- int a = 2+3; // Suma a doi întregi
- String str = “acesta este ” + “un text”; // Concatenare de șiruri

Observație! Trebuie avut în vedere contextul static al funcției main. Dintr-un context static nu se pot apela functii nestatice, în schimb se pot creea obiecte ale oricărei clase, după cum se observă și în exemplul de mai sus, unde am creeat un obiect de tip mașină și i-am setat prețul.

2 Probleme de laborator

2.1 Problema 1

(2 puncte) Să se definească o clasă "Complex" pentru operații cu numere complexe și să se testeze metodele implementate. Clasa va avea doi constructori astfel:

- cu două argumente(partea reală și imaginară)
- fără argumente

Metodele necesare sunt: adunare, înmulțire, ridicare la putere naturală, "equals" și "toString" (șir de forma (re,im)).

2.2 Problema 2

(2 puncte) Să se definească o clasă "Stivă" pentru stive de numere întregi, reprezentate prin vectori intrinseci.

Datele clasei:

- Un vector de întregi
- Indicele elementului din vârful stivei (ultimul introdus)

Metodele clasei:

- Constructor fără parametri (dimensiunea implicită a stivei = 100)
- Constructor cu parametru dimensiunea stivei
- void push(int) : pune un intreg dat pe stiva
- int pop() : scoate elementul din vârful stivei
- boolean isEmpty(): verifică dacă stiva este goală

Considerați cazurile de stivă plină (la *push*) și stivă goală (la *pop*). Metoda statică **main** pentru verificarea operațiilor cu o stivă va fi inclusă în clasa **Stiva**.

2.3 Problema 3

(2 puncte) Să se definească o clasă *Matrix* pentru operații uzuale cu matrice pătratice de numere reale: adunare, înmulțire, "toString". Să se scrie un program pentru ridicarea la o putere întreagă a unei matrice pătratice.

2.4 Problema 4

(2 puncte) Să se definească o clasă *IntSet* pentru o mulțime de numere întregi, care conține un vector intrinsec de întregi.

Metodele clasei:

- Constructor cu argument dimensiune multime
- void add(int) : adăugare element, dacă nu există
- boolean contains(int) : verifică dacă un număr dat se află sau nu în mulțime

- String toString() : mulțimea de elemente transformată în șir de caractere

Să se scrie un program care crează o mulțime folosind clasa IntSet și testează operațiile de mai sus prin adăugări succesive și afișări.

2.5 Problema 5

(2 puncte) Să se definească o clasă *IntSet* pentru o mulțime de întregi, cu metodele *add*, *remove*, *contains* și "toString" care apelează metode din clasa *BitSet*(care reprezintă un vector de biți; astfel bitul *i* este true daca *i* aparține mulțimii). Clasa *IntSet* conține o variabilă de tip *BitSet* și apelează metode ale acestei clase.

Metode din clasa *BitSet*

- boolean get(int bitIndex) : întoarce true sau false dacă bitul cu indexul *bitIndex* a fost setat
- void set(int bitIndex, boolean value) : setează bitul de la indexul *bitIndex* la valoarea *value*

Pentru mai multe detalii despre clasa BitSet consultați API-ul Java.

2.6 Problema 6 (bonus)

(1 punct) Să se definească o clasă *HSet* pentru o mulțime de obiecte realizată ca tabel de dispersie, folosind clasa *Hashtable* (pachetul *java.util*). Metode care vor fi implementate: *add*, *remove*, *contains*, *toString*. Clasa *HSet* conține o variabilă de tip *Hashtable* și apelează metodele acesteia.

2.7 Problema 7 (bonus)

(2 puncte) Să se definească o clasă *Graph* pentru grafuri orientate, în care arcele nu au asociat un cost, iar nodurile sunt numerotate de la 1.

Datele clasei (private):

- O matrice cu componente de tip *boolean* (matricea de adiacențe)
- Numarul de noduri

Metodele clasei:

- Constructor cu parametru întreg (numărul de noduri din graf)
- int size() : are ca rezultat numărul de noduri din graf
- void addArc(int v, int w) : adaugă un arc la graf (între v si w)
- boolean isArc(int v, int w) : verifică dacă există arc intre nodurile v si w
- void print() : afișarea matricei de adiacențe (cu 1 si 0)
- void dfs(int v, boolean vazut[]) : vizitare în adâncime din nodul v.

Pentru verificare se va crea un graf prin adăugări succesive de arce. Se va afișa matricea de adiacențe a grafului și se va afisa ordinea nodurilor la explorarea în adâncime din fiecare nod al grafului:

```
public void dfs(int v, boolean vazut[]) {
   int w;
   vazut[v] = true;
   for (w = 1; w <= n; w++)
      if (isArc(v, w) && !vazut[w]) {
         System.out.println (v + ``-'' + w);
         dfs (w,vazut);
      }
}
```