Laborator 4 &5 – Bazele Programarii Orientate pe Obiect

Consideratii teoretice.

1. Tablouri

Tablourile reprezinta sunt structuri de lungime fixa, care pastreaza valori de acelasi tip. In limbajul Java tablourile extind clasa java.lang.Object. Valorile retinute in elementele unui tablou pot fi date primitive sau referinte. Fiecare element al unui tablou poate fi accesate prin intermediul unui indice numeric.

Un tablou este **declarat** similar cu declararea o variabila obisnuita. Declararea unui tablou se poate face in doua moduri:

```
String [] names;
int ages [];
```

In declarerea unui tablou trebuie sa specificam tipul de date pe care le stocam si un identificator valid. Spre deosebire de alte limbaje, in java nu se specifica dimensiunea tabloului în declaraţie (ex. int[10] scores).

Dupa declarearea unui tablou poate fi alocata memorie pentru elementele acestuia. **Instantierea** unui tablou se realizeaza prin intermediul operatorului **new**.

```
names= new String[10];
ages= new int[20];
```

Declararea si instantierea tabloului se poate face printr-o singura linie de cod ca in exemplu de mai jos:

```
String [] names= new String[10]
int ages []={1,2,3}
```

Accesul la elementele unui tablou se face prin intermediul indexului. Primul element dintr-un tablou are asociat indexul zero.

```
System.out.printl("prima valoare stocata in tabloul ages este "+ ages[0])
```

```
>>output : prima valoare stocata in tabloul ages este 1
```

Determinarea dimensiunii unui tablou se face cu ajutorul atribut **length** care specifica numarul de elemente din tablou.

Exemplu de afisare a elementelor unui tablou folosind proprietatea lenght:

```
class Names {
 public static void main(String[] args) {
   String names[] = { "ana", "ada", "tudor", "george", "natalia", "victor" };
            for (String name : names) {
                        System.out.println(name);
                        }
            }
}
```

In java, **tablouri multidimensionale** se declara specificand numarul de linii si de coloane:

tip numeTablou[ ] [ ] = new tip [nrLinii] [nrColoane];

Parcurgerea unui tablou multidimensional se realizeaza folosind cicluri imbricate.

Exemplu de parcurgere a elementelor unei matrici:

```
class Matrice {
 public static void main(String[] args) {
   char[][] matrice = { { 'X', '0', 'X' }, { '0', '0', 'X' }, {'X', '0', '0' } };
    for (int i = 0; i < matrice.length; i++) {
          for (int j = 0; j < matrice[i].length; j++) {
                    System.out.println("matrice[" + i + "," + j + "]=" + matrice[i][j] + "\t");
                    }
         }
    }
}
```

>>output

```
matrice[0,0]=X    matrice[0,1]=0     matrice[0,2]=X
matrice[1,0]=0   matrice[1,1]=0     matrice[1,2]=X
matrice[2,0]=X     matrice[2,1]=0   matrice[2,2]=0
```

Pentru a parcurge o matrice sunt necesare doua bucle for. Bucla interioara se executa in intregime pentru fiecare iteratie a buclei exterioare.

2. **Mostenirea**

Mostenirea este procesul prin care o clasa numita clasa copil (sau subclasa) extinde o alta clasa numita clasa parinte (sau superclasa). Prin moştenirea clasa copil dobândeşte metodele şi câmpurile clasei parinte. Moştenirea ne permite să refolosim codul deja existent şi sa ne definim structure ierahice intre clase. Mostenirea se specifica prin cuvantul cheie **extends** care apare in declaratia clasei copil urmat de numele clasei parinte.

Exemplu de relatie de mostenire intre doua clase:

```
class Persoana{
 String nume;
 Persoana(String nume)
   {
     this.nume=nume;
    }
 public String toString()
   {
    return „nume : „+ nume;
   }
}

Class Profesor extends Persoana{
 String materie;
 Profesor(String nume, String m){
```

```
   super(nume);
   this.materie=m;
  }
  public String toString(){
    return super.toString()+", materie"+ materie;
}

}
```

3. Clase abstracte

Clasele abstracte se utilizeaza in contextul relatiei de mostenire. O clasa abstracta poate fi utilizata ca si superclasa pentru alte clase care extind clasa abstracta. Clasele abstracte sunt declarate prin intermediul cuvantului cheie abstract care apare in declaratia clasei. O clasa abtracta nu poate fi instantiata. O clasa abstracta poate contine atribute, care descriu caracteristicile clasei, metode implementate si metode care nu sunt implementate. Metodele care nu sunt implementate sunt declaratie si au in declaratia lor cuvantul cheie abstracte. Clasa copil derivata dintr-o clasa parinte abstracta trebuie sa implementeze metodele abstracte declarate in clasa parinte.

Examplu de clasa abstracta:

```
public abstract class FiguraGeometrice{
String nume;
FiguraGeometrica(String n){
 this.nume=n;
}
public String toString(){
  return „nume"+nume;
}
public abstract double calculArie();

}

Public class Cerc extends FiguraGeometrica{
  double raza;
  Cerc(String nume, double raza){
    super(nume);
    this.raza=raza;
}
 public double arie(){
    return Math.Pi*Math.pow(raza, 2);

}
}
```

Nota. Temele for fi implementate pe durata laboratoarelor 4 si 5. Ultima tema va fi implementata doar de catre studentii care nu au ales ca si materie optional Tehnici de Programare.

1. Implementati o aplicatie java pentru gestiunea unei librari.
   - Creați clasa **Book** cu următoarele atribute: title, author, publisher, pageCount, number of copies, price. Definiti constructorii necesari in clase, o metoda public **toString**().
   - Creați clasa **Bookstore**, pentru a testa viitoarele funcționalități ale librariei. Completați această clasă, cu urmatoarele metode: metode de **adaugare** a unei carti in **BookStore**, metoda de stergere a unei carti, metode de cautare (dupa autor, dupa publisher), etc.
   - Pentru crearea obiectelor de tip carte și setarea atributelor introducând date de la tastatură folositi clasa Scanner: Verificați ca numărul de pagini introdus să fie diferit de zero.
   - Creați o clasă nouă, **BookstoreCheck**, ce va conține două metode.
     - Prima metodă va verifica dacă o carte este în dublu exemplar, caz în care va întoarce adevărat.
     - A doua metodă va verifica care carte este mai groasă decât altă, și va întoarce cartea mai groasă.
   - Testați aceste doua metode.
   - Afișați informația despre o carte în felul următor:

```
BOOK_TITLE: [insert_book_title], BOOK_AUTHOR: [insert_book_author],
BOOK_PUBLISHER: [insert_book_publisher]
```

2. a) Definiti clasa **Persoana** cu campul *nume* de tip **String**, si o metoda **toString**() care va returna numele persoanei. Clasa va mai contine:
   - Un constructor fara parametri
   - Un constructor ce va initializa numele.

   Din clasa **Persoana** derivati clasele **Profesor** si **Student**:
   - Clasa **Profesor** va avea un camp *materie* de tip **String**.
   - Clasa **Student** va avea un camp *nota* de tip **int**.
   - In clasele **Profesor** si **Student** se vor defini:
     - Constructori fara parametri
     - Constructori care permit initializarea campurilor claselor.
   - Instantiati clasele **Profesor** si **Student**, si apelati metoda **toString**() pentru fiecare instanta.

   b) Adaugati metode **toString**() in cele doua clase derivate, care sa returneze tipul obiectului, numele si membrul specific. De exemplu:
   - pentru clasa **Profesor**, se va afisa: "Profesor: Ionescu, POO"
   - pentru clasa **Student**, se va afisa: "Student: Popescu, 5"
   - Modificati implementarea **toString**() din clasele derivate astfel incat aceasta sa utilizeze implementarea metodei **toString**() din clasa de baza.

   b) Adaugati o metoda **equals** in clasa **Student**. Justificati criteriul de echivalenta ales.

   c) Creati un vector de obiecte **Persoana** si populati-l cu obiecte de tip **Profesor** si **Student**. Parcurgeti acest vector si apelati metoda **toString**() pentru elementele sale.

3. Implementaţi un program care să simuleze lucrul cu un ghiozdan cu rechizite. Soluţia creată trebuie să conţină 5 clase:
   - O clasă abstractă **Rechizita** cu următoarele proprietăţi:
     - Un atribut etichetă (variabila care va descrie numele rechizitei) si un atribut pret.
     - O metodă abstractă getNume()
   - Două subclase **Manual** şi **Caiet** ce extind clasa abstractă rechizite. Pe langa atributele mostenite din clasa parinte, clasa **Caiet** va contine ca si atribute tipul caietului si numar de file, iar clasa **Manual** va contine ca atribute autorii manualului si editura. Ambele clase vor implementa metoda getNume() intr-un mod diferit. Spre exemplu, ambele vor returna valoarea etichetei din Rechizita, însă pentru a

face diferenţa de implementare pentru fiecare clasă în parte se va adăuga substringul —Caiet sau —Manual în numele etichetei.

- O clasă Ghiozdan care va conţine:
    - O listă de rechizite (implementată cu tipul de date tablou).
    - O metoda **add** pentru adăugarea unei Rechizite în lista de rechizite
    - O metoda **remove** pentru eliminarea rechizitei aflate in pozitie *poz* Metoda returneaza rechizita stearsa
    - O metoda **elementAt** care returneaza elemental aflat pe pozitia *poz*
    - O metoda **listItems** pentru listarea tuturor rechizitelor
    - O metoda **listManual** pentru listarea doar a manualelor din listă
    - O metoda **listCaiet** pentru listarea doar a caietelor din listă
    - O metoda **getNrRechizite** pentru afisarea numărului de rechizite
    - O metoda **getNrManuale** pentru a calcularea şi afisarea nr. de manuale din listă
    - O metoda **getNrCaiete** pentru a calcularea şi afisarea nr. de Caiete din listă
    - O metoda **equals** care compara daca doua ghiozdane au acelasi continut.
- O clasă TestGhiozdan cu metoda main în care se vor efectua următoarele operaţii:
    - Instanţierea unui obiect de tip Ghiozdan
    - Crearea mai multor obiecte de tip Caiet şi Manual, şi adăugarea lor în lista de rechizite
    - Afişarea nr. total de rechizite, nr. de caiete şi nr. de manuale
    - Listarea tuturor rechizitelor
    - Listarea doar a rechizitelor de tip Caiet
    - Listarea doar a rechizitelor de tip Manual

Nota: Pentru implementarea clasei **Ghiozdan** se va folosi un tablou.