Laborator 01

Exemplu 01 - Despre utilizarea metodei main

```java
package upg.laborator02.exemplu01;
//Numele pachetului de care apartine clasa (optional).

public class Principal {
    //Numele clasei.
    //Atentie la regulile de denumire in Java!!!

    public static void main(String[] args) {
        /*
        Metoda main. o numim metoda si nu o simpla functie, pentru
        ca este o functie membra a clasei deschise mai sus.
        Aici varianta "normala".
        In Java main cea "normala" este intotdeauna publica (ca sa poata fi accesata).
        In Java main cea "normala" returneaza intotdeauna void.
        In Java main cea "normala" este intotdeauna statica.
        */

        /*
        Scurta recapitulare de la POO:
        1. Metoda sau atribut static = poate fi accesat(a) / utilizat(a)
        si fara a avea vreun obiect din clasa respectiva.
        2. Metodele statice pot accesa si modifica atributele statice.
        3. Metodele statice nu pot fi suprascrise.
        4. Metodele statice pot fi supraincarcate.
        */

        /*Bizarerii ale metodei main in Java
        1. main, in particular, chiar daca este supraincarcata, se va executa
        automat, la pornirea programului, tot varianta "normala" (cea cu
        (String[] args)), deci supraincarcarea lui main, din acest punct de
        vedere, nu functioneaza.
        2. Daca totusi insist sa supraincarc main, pot utiliza celelalte
        variante ale lui main apelandu-le direct sau indirect din main-ul "normala".
        Ele nu se vor comporta totusi ca o functie "main" ci ca o functie oarecare.
        */
        System.out.println("Executia functiei main \"normale\"");
        //am terminat practic de executat functia main "normala" ...

        //o apelez si pe cealalta:
        main();
    }

    private static int main() {
        //o alta functie main care teoretic supraincarca functia main deja existenta
        //in realitate, este o functie oarecare, care poate fi folosita sau nu intr-o
        //alta functie, inclusiv in functia main "normala"
        System.out.println("Executia functiei main \"speciale\"");
```

```
        return 0;
    }
}
```

Exemplu 02 - Variabile. Intrari-ieșiri la consolă

```
package upg.laborator02.exemplu02;
//Numele pachetului de care apartine clasa (optional).

import java.util.Scanner;
//importarea unei biblioteci (uzual este un pachet sau o clasa)

public class Principal {
    //Numele clasei.
    //Atentie la regulile de denumire in Java!!!

    public static void main(String[] args) {
        //Metoda main ...
        //Scrierea la consola, fara trecere pe linie noua. Folosim iesirea standard
        //din clasa/biblioteca System, care face parte din pachetul java.lang, care
        //este automat importat in toate aplicatiile Java (deci nu trebuie sa il
        //import eu explicit). folosim iesirea standard System.out

        System.out.print("Am scris niste chestii aici. "
                        + "Urmatoarea afisare va continua pe acelasi rand. ");
        //Daca am scris o linie prea lunga si ma incurca, pot da enter in mijlocul
        //unui string. Eclipse, in mod foarte amabil, imi va rupe stringul in doua
        //stringuri si le va concatena cu operatorul +.
        //Scrierea la consola, cu trecere pe linie noua:
        System.out.println("Incep de unde am ramas. Trec pe linie noua");
         System.out.println("Am ajuns pe linie noua ... sa afisam valorile unor
variabile:");

        //Diverse declaratii de variabila. Un double, un int, un string.
        double x = 12.73;
        int y = -95;
        String z="Un sir de caractere.";

        /*
        Tipurile de date primitive (care nu sunt alcatuite din alte tipuri) in Java sunt:
        boolean, byte, char, short, int, long, float, double.
         String NU este un tip de date primitiv. Este o clasa bazata pe tipul de date
primitiv
        char.
        */
        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
        //in print si println pot sa fac si combinatii de diverse tipuri de date:
        System.out.println("Un intreg " + y + " afisat langa un double " + x);

        /*
```

```
        Biblioteca System furnizeaza o gramada de facilitati interesante, printre care
        si care si iesirea standard pe care am vazut-o mai sus (System.out), intrarea
        standard (System.in) si iesirea standard pentru mesajele de eroare (System.err).
        Aceste sunt in mod normal "legate" la consola dar pot fi redirectate.
        */
        //System.in poate fi utiliza si separat dar, cel putin la inceput, se
        //utilizeaza usor impreuna cu un obiect din clasa Scanner. Pentru utilizarea
        //clasei Scanner trebuie importat pachetul java.util.Scanner (vezi la
        //inceputul acestui program)

        Scanner intrare=new Scanner(System.in);

        //Odata definit un obiect din clasa Scanner bazat pe un strem de intrare
        //oarecare. Acest obiect poate fi folosit pentru a citi orice tip primitiv de
        //date si unele tipuri derivate. Da un clic pe consola ca sa fii sigur ca
        //introduci date la consola si nu scrii valori aiurea prin codul tau :) :

        System.out.println("Introdu o valoare de tip double: ");
        //citesc in variabila x o valoare double (folosind obiectul intrare din clasa
        //Scanner)
        x=intrare.nextDouble();

        System.out.println("Introdu o valoare de tip int: ");
        //citesc in variabila y o valoare int (folosind obiectul intrare din clasa
        //Scanner)
        y=intrare.nextInt();

        System.out.println("Introdu un sir de caractere: ");
        //citesc in variabila z un sir de caractere (folosind obiectul intrare din
        //clasa Scanner)
        z=intrare.next();

        System.out.println("Am citit: ");
        System.out.println(x);
        System.out.println(y);
        System.out.println(z);

        //Executia se termina automat odata ce s-a terminat tot ce era in main si nu
        //te anunta nimeni de asta. E folositor cateodata sa bagi un mesaj in acest
        //sens:
        System.out.println("Executia programului s-a terminat!");
    }
}
```

Exemplu 03 - Rezolvarea unei probleme simple cu date numerice, fără tratarea excepțiilor

```
package upg.laborator02.exemplu03;
//Numele pachetului de care apartine clasa (optional).

import java.util.Scanner;
//importarea unei biblioteci (uzual este un pachet sau o clasa)
```

```
public class Principal {
    //Numele clasei.

    public static void main(String[] args) {
        //Metoda main ...
        //incerc sa rezolva o problema simpla. ecuatia de gradul 1
        double a,b,x;
        //declar variabilele corespunzatoare datelor de intrare si de iesire
        Scanner intrare=new Scanner(System.in);
        //creez un nou obiect din clasa Scanner pentru citirea de la consola
        //nu uitati sa dati clic pe consola pentru a introduce date
        System.out.println("Introduceti coeficientii ecuatiei ax+b=0");
        System.out.println("a=");
        a=intrare.nextDouble();
        System.out.println("b=");
        b=intrare.nextDouble();
        x=-b/a;
        System.out.println("Solutia ecuatiei este x=" + x);
        System.out.println("Executia programului s-a terminat!");
    }
}
```

Exemplu 04 - Rezolvarea unei probleme simple cu date numerice, cu tratarea excepțiilor

```
package upg.laborator02.exemplu04;
//Numele pachetului de care apartine clasa (optional).

import java.util.InputMismatchException;
import java.util.Scanner;
//importarea unor biblioteci (uzual este un pachet sau o clasa)

public class Principal {
    //Numele clasei.

    public static void main(String[] args) {
        //Metoda main ...
        //incerc sa rezolva o problema simpla. ecuatia de gradul 1
        double a=0,b=0,x=0;
        //declar variabilele corespunzatoare datelor de intrare si de iesire
        Scanner intrare=new Scanner(System.in);
        //creez un nou obiect din clasa Scanner pentru citirea de la consola
        //nu uitati sa dati clic pe consola pentru a introduce date
        System.out.println("Introduceti coeficientii ecuatiei ax+b=0");
        System.out.println("a=");

        /*
        Imi doresc sa ma sigur ca programul nu da eroare in cazul in care utilizatorul nu
        introduce o valoare numerica, asa ca ma apuc sa fac tratarea exceptiilor. In cazul
        de fata va fi o exceptie de tip InputMismatchException. Pot afla cu ce tip de
exceptie
```

```
	voi lucra provocand pur si simplu aparitia exceptiei respective (de exemplu aici, pot
	tasta "mere" in loc de o valoare numerica pentru a sau pentru b).
	Pot si sa tratez la gramada toate exceptiile, de orie tip ar fi, folosind tipul generic
	Exception.
	Blocul de tratare a exceptiilor  comporta absolut la fel ca in C# sau C++:
	try{
		instructiunea sau instructiunile care ar putea genera o exceptie
	}
	catch(tipul_exceptiei variabila_in_care_stochez_informatii_despre_exceptie){
			ce fac in cazul aparitiei exceptiei
	}
	*/

	try {
	a=intrare.nextDouble();
	}
	catch (InputMismatchException exceptie) {
		System.err.println("Valoarea introdusa pentru a nu poate fi folosita ca si valoare double!");
		System.err.println("A aparut o exceptie de tip: "+exceptie.getMessage());
		//System.err este iesirea standard pentru mesaje de eroare
		//mesajele vor aparea tot la consola, doar ca vor fi scrise cu rosu
		System.err.println("care este cauzata de: "+exceptie.getCause());
		System.err.println("din clasa: "+exceptie.getClass());
		System.err.println("Mai multe informatii despre aceasta exceptie: ");
		exceptie.printStackTrace();
		//pot face diverse chestii ca sa ma lamuresc care a fost problema
		System.err.println("Executia se opreste aici!");
		//incerc sa fiu politicos
		return;
		//oprirea fortata a executiei
	}
	System.out.println("b=");
	try {
	b=intrare.nextDouble();
	}
	catch (InputMismatchException exceptie) {
		System.err.println("Valoarea introdusa pentru b nu poate fi folosita ca si valoare double!");
		System.err.println("A aparut o exceptie de tip: "+exceptie.getMessage());
		System.err.println("care este cauzata de: "+exceptie.getCause());
		System.err.println("din clasa: "+exceptie.getClass());
		System.err.println("Mai multe informatii despre aceasta exceptie: ");
		exceptie.printStackTrace();
		//pot face diverse chestii ca sa ma lamuresc care a fost problema
		System.err.println("Executia se opreste aici!");
		//incerc sa fiu politicos
		return;
		//oprirea fortata a executiei
	}
	//daca ambele citiri s-au facut cu succes si am ajuns pana aici, il pot calcula pe x
```

```
        x=-b/a;
        System.out.println("Solutia ecuatiei este x=" + x);
        System.out.println("Executia programului s-a terminat!");
    }
}
```

Exemplu 05 - Rezolvarea unei probleme simple cu date numerice, cu tratarea excepțiilor, cu generarea unor excepții proprii

```
package upg.laborator02.exemplu05;
//Numele pachetului de care apartine clasa (optional).

import java.util.InputMismatchException;
import java.util.Scanner;
//importarea unor biblioteci (uzual este un pachet sau o clasa)

public class Principal {
    //Numele clasei.

   public static void main(String[] args) {
        //Metoda main ...
        //incerc sa rezolva o problema simpla. ecuatia de gradul 1
        double a=0,b=0,x=0;
        //declar variabilele corespunzatoare datelor de intrare si de iesire
        Scanner intrare=new Scanner(System.in);
        //creez un nou obiect din clasa Scanner pentru citirea de la consola
        //nu uitati sa dati clic pe consola pentru a introduce date
        System.out.println("Introduceti coeficientii ecuatiei ax+b=0");
        System.out.println("a=");

        /*
        Imi doresc sa ma sigur ca programul nu da eroare in cazul in care utilizatorul nu
        introduce o valoare numerica, asa ca ma apuc sa fac tratarea exceptiilor. In cazul
        de fata va fi o exceptie de tip InputMismatchException. Pot afla cu ce tip de
exceptie
        voi lucra provocand pur si simplu aparitia exceptiei respective (de exemplu aici,
pot
        tasta "mere" in loc de o valoare numerica pentru a sau pentru b).
        Pot si sa tratez la gramada toate exceptiile, de orie tip ar fi, folosind tipul
        Generic Exception.
        Blocul de tratare a exceptiilor  comporta absolut la fel ca in C# sau C++:
        try{
               instructiunea sau instructiunile care ar putea genera o exceptie
        }
        catch(tipul_exceptiei variabila_in_care_stochez_informatii_despre_exceptie){
                      ce fac in cazul aparitiei exceptiei
        }
        */

        try {
        a=intrare.nextDouble();
        }
        catch (InputMismatchException exceptie) {
```

```
                System.err.println("Valoarea introdusa pentru a nu poate fi folosita ca si
valoare double!");
                System.err.println("A aparut o exceptie de tip: "+exceptie.getMessage());
                System.err.println("care este cauzata de: "+exceptie.getCause());
                System.err.println("din clasa: "+exceptie.getClass());
                System.err.println("Mai multe informatii despre aceasta exceptie: ");
                exceptie.printStackTrace();
                //pot face diverse chestii ca sa ma lamuresc care a fost problema
                System.err.println("Executia se opreste aici!");
                //incerc sa fiu politicos
                return;
                //oprirea fortata a executiei
        }
        System.out.println("b=");
        try {
        b=intrare.nextDouble();
        }
        catch (InputMismatchException exceptie) {
                System.err.println("Valoarea introdusa pentru b nu poate fi folosita ca si
valoare double!");
                System.err.println("A aparut o exceptie de tip: "+exceptie.getMessage());
                System.err.println("care este cauzata de: "+exceptie.getCause());
                System.err.println("din clasa: "+exceptie.getClass());
                System.err.println("Mai multe informatii despre aceasta exceptie: ");
                exceptie.printStackTrace();
                //pot face diverse chestii ca sa ma lamuresc care a fost problema
                System.err.println("Executia se opreste aici!");
                //incerc sa fiu politicos
                return;
                //oprirea fortata a executiei
        }

        //facem cunostinta cu exceptiile generate chiar de noi
        try {
                        if(a==0)
                                throw new ExceptieRezultatInfinit();
                                //cu throw generez chiar eu o exceptie de ce tip vreau
                                //poate sa fie un tip de exceptie pre-existent sau pot sa imi
                                creez propriul tip
                                //in acest caz chiar mi-am creat propriul tip de exceptie (a
se
                                vedea mai jos)
        }
        catch (ExceptieRezultatInfinit exceptie) {
                System.err.println("Valoarea introdusa pentru a este 0!");
                System.err.println("A aparut o exceptie de tip: "+exceptie.getMessage());
                System.err.println("care este cauzata de: "+exceptie.getCause());
                System.err.println("din clasa: "+exceptie.getClass());
                System.err.println("Mai multe informatii despre aceasta exceptie: ");
                exceptie.printStackTrace();
                //pot face diverse chestii ca sa ma lamuresc care a fost problema
                System.err.println("Executia se opreste aici!");
                //incerc sa fiu politicos
                return;
```

```
                //oprirea fortata a executiei
                }
        //daca ambele citiri s-au facut cu succes si a nu a fost 0 si am ajuns pana aici,
il pot calcula pe x
        x=-b/a;
        System.out.println("Solutia ecuatiei este x=" + x);
        System.out.println("Executia programului s-a terminat!");
    }
}

class ExceptieRezultatInfinit extends Exception{
        //Tipul meu de exceptie. Mosteneste clasa generala Exception
        public String toString(){
        //Nu face nimic special, doar returneaza un mesaj, folosind metoda toString().
        return ("S-a efectuat o impartire la 0 si rezultatul a fost o valoare infinita!") ;
        }
}
```

Exemplu 06 - Lucrul cu clase proprii. Clasele sunt puse în același fișier (neuzual pentru Java, dar acceptabil pentru o cantitate mica de cod)

```
package upg.laborator02.exemplu06;
//Numele pachetului de care apartine clasa (optional).

import java.util.Scanner;
//importarea unor biblioteci (uzual este un pachet sau o clasa)

public class Principal {
    //Numele clasei.

    public static void main(String[] args) {
        //Metoda main ...
        Student student1 = new Student("Ionescu", "Eugen", "1234", "50320");
        //un nou obiect de tip Student care utilizeaza constructorul explicit
        //acest obiect are deja valori pentru atribute asa ca poate fi utilizat imediat
        System.out.println(student1);
        //cand afisez un obiect, metoda toString() corespunzatoare clasei din care face
parte
        //obiectul este apelata automat (a se vedea mai jos, la sfarsitul clasei Student)

        Student student2 = new Student();
        //un nou obiect de tip Student care utilizeaza constructorul implicit
        //acest obiect nu are valori pentru atribute asa ca trebuie sa le acord prin seteri
        //daca as incerca sa afisez continutul obiectului acum, mi-ar returna valori nule
        System.out.println(student2);

        student2.setNume("Georgescu");
        student2.setPrenume("Marioara");
        student2.setGrupa("99999");
        student2.setNrMatricol("9876");
        //acum atributele obiectului au valori asa ca pot face afisarea:
```

```
        System.out.println(student2);
    }
}

class Student{
        //O clasa definita de mine
        private String nume, prenume, nrMatricol, grupa;

        public Student() {
                //constructorul implicit (fara parametri)

        }

        public Student(String nume, String prenume, String nrMatricol, String grupa) {
                //un constructor explicit (cu parametri)
                this.nume=nume;
                this.prenume=prenume;
                this.nrMatricol=nrMatricol;
                this.grupa=grupa;
        }

        //seteri si geteri pentru toate atributele:
        public String getNume(){
                return nume;
        }

        public void setNume(String nume) {
                this.nume=nume;
        }

        public String getPrenume(){
                return prenume;
        }

        public void setPrenume(String prenume) {
                this.prenume=prenume;
        }

        public String getNrMatricol(){
                return nrMatricol;
        }

        public void setNrMatricol(String nrMatricol) {
                this.nrMatricol=nrMatricol;
        }

        public String getGrupa(){
                return grupa;
        }

        public void setGrupa(String grupa) {
                this.grupa=grupa;
        }
```

```
        //o metoda generica folosita pentru afisarea continutului obiectului:
        //(aceasta functie este denumita traditional toString)
        public String toString(){
                return "Studentul " + nume + " " + prenume + " este de la grupa " + grupa +
"
                si are numarul matricol " + nrMatricol;
        }
}
```

Exemplu 07 - Lucrul cu clase proprii. Clasele sunt puse în fișiere diferite

Primul fișier (cel cu metoda main):

```
package upg.laborator02.exemplu07;
//Numele pachetului de care apartine clasa (optional).

import java.util.Scanner;
//importarea unor biblioteci (uzual este un pachet sau o clasa)

public class Principal {
    //Numele clasei.

    public static void main(String[] args) {
        //Metoda main ...
        Student student1 = new Student("Ionescu", "Eugen", "1234", "50320");
        //un nou obiect de tip Student care utilizeaza constructorul explicit
        //acest obiect are deja valori pentru atribute asa ca poate fi utilizat imediat
        System.out.println(student1);
        //cand  afisez  un  obiect,  metoda  toString()  corespunzatoare  clasei  din  care  face
parte
        //obiectul este apelata automat (a se vedea in celalalt fisier, la sfarsitul clasei
        //Student)

        Student student2 = new Student();
        //un nou obiect de tip Student care utilizeaza constructorul implicit
        //acest obiect nu are valori pentru atribute asa ca trebuie sa le acord prin seteri
        //daca as incerca sa afisez continutul obiectului acum, mi-ar returna valori nule
        System.out.println(student2);

        student2.setNume("Georgescu");
        student2.setPrenume("Marioara");
        student2.setGrupa("99999");
        student2.setNrMatricol("9876");
        //acum atributele obiectului au valori asa ca pot face afisarea:
        System.out.println(student2);
    }
}
```

Al doilea fișier (cel cu clasa Student):

```
package upg.laborator02.exemplu07;
```

```java
public class Student {
	//O clasa definita de mine
	private String nume, prenume, nrMatricol, grupa;

	public Student() {
		//constructorul implicit (fara parametri)

	}

	public Student(String nume, String prenume, String nrMatricol, String grupa) {
		//un constructor explicit (cu parametri)
		this.nume=nume;
		this.prenume=prenume;
		this.nrMatricol=nrMatricol;
		this.grupa=grupa;
	}

	//seteri si geteri pentru toate atributele:
	public String getNume(){
		return nume;
	}

	public void setNume(String nume) {
		this.nume=nume;
	}

	public String getPrenume(){
		return prenume;
	}

	public void setPrenume(String prenume) {
		this.prenume=prenume;
	}

	public String getNrMatricol(){
		return nrMatricol;
	}

	public void setNrMatricol(String nrMatricol) {
		this.nrMatricol=nrMatricol;
	}

	public String getGrupa(){
		return grupa;
	}

	public void setGrupa(String grupa) {
		this.grupa=grupa;
	}

	//o metoda generica folosita pentru afisarea continutului obiectului:
	//(aceasta functie este denumita traditional toString)
	public String toString(){
```

```
                return "Studentul " + nume + " " + prenume + " este de la grupa " + grupa +
"
                si are numarul matricol " + nrMatricol;
        }
}
```

Exemplu 08 - Lucrul cu vectori. Sortarea unui vector folosind metoda “de școală”

```
package upg.laborator02.exemplu08;

import java.util.Scanner;

public class Principal {
    public static void main(String[] args) {
        //declaram vectorul si ii alocam memorie suficienta pentru 10 elemente
        int[] vector=new int[10];
        //declaram un obiect nou de tip Scanner pentru citirea de la consola
        Scanner intrare=new Scanner(System.in);
        System.out.println("Introduceti elementele vectorului:");
        //facem citirea elementelor vectorului
        for(int i=0;i<vector.length;i++)
        //in Java nu se poate utiliza iteratia pe domeniu (for-each loop) pentru
        //modificarea elementelor vectorilor si matricilor. folosesc o iteratie
        //clasica (sau as putea utiliza o iteratie cu iterator propriu-zis)
        {
                System.out.print("element:");
                vector[i]=intrare.nextInt();
        }
        //nu mai am nimic de citit. pot sa inchid intrarea
        intrare.close();
        System.out.println("Elementele vectorului, in ordinea initiala, sunt:");
        //o parcurgere. aici pot sa folosesc iteratia pe domeniu
        for(int x:vector)
        {
                System.out.print(x+" ");
        }
        //sortarea. folosim metoda "de scoala"
        //de asemenea implica modificare elementelor vectorului, deci de asemenea
        //nu se poate utiliza iteratia pe domeniu
        for(int i=0;i<vector.length-1;i++)
                for(int j=i+1;j<vector.length;j++)
                        if(vector[i]>vector[j])
                        {
                                int temp=vector[i];
                                vector[i]=vector[j];
                                vector[j]=temp;
                        }
        System.out.println();
        System.out.println("Elementele vectorului, dupa sortare, sunt:");
        //o parcurgere. aici pot sa folosesc iteratia pe domeniu
        for(int x:vector)
```

```
        {
                System.out.print(x+" ");
        }
    }
}
```

Exemplu 09 - Lucrul cu vectori. Sortarea unui vector folosind metoda *sort* din clasa / biblioteca *Arrays*

```
package upg.laborator02.exemplu09;

import java.util.Scanner;
import java.util.Arrays;

public class Principal {

    public static void main(String[] args) {
        //declaram vectorul si ii alocam memorie suficienta pentru 10 elemente
        int[] vector=new int[10];
        //declaram un obiect nou de tip Scanner pentru citirea de la consola
        Scanner intrare=new Scanner(System.in);
        System.out.println("Introduceti elementele vectorului:");
        //facem citirea elementelor vectorului
        for(int i=0;i<vector.length;i++)
        //in Java nu se poate utiliza iteratia pe domeniu (for-each loop) pentru
        //modificarea elementelor vectorilor si matricilor. folosesc o iteratie
        //clasica (sau as putea utiliza o iteratie cu iterator propriu-zis)
        {
                System.out.print("element:");
                vector[i]=intrare.nextInt();
        }
        //nu mai am nimic de citit. pot sa inchid intrarea
        intrare.close();
        System.out.println("Elementele vectorului, in ordinea initiala, sunt:");
        //o parcurgere. aici pot sa folosesc iteratia pe domeniu
        for(int x:vector)
        {
                System.out.print(x+" ");
        }
        //sortarea. folosim metoda din biblioteca Arrays. este o metoda supraincarcata
        //asa ca poate fi folosita in diverse moduri
        Arrays.sort(vector);
        System.out.println();
        System.out.println("Elementele vectorului, dupa sortare, sunt:");
        //o parcurgere. afisarea continutului vectorilor
        for(int x:vector)
        {
                System.out.print(x+" ");
        }
    }
}
```

Exemplu 10 - Lucrul cu matrice. Transpusa unei matrice

```
package upg.laborator02.exemplu10;

public class Principal {

    public static void main(String[] args) {
        //declaram matricele
        //primeia ii dam direct continutul
        //cu ocazia asta vedem cum putem sa stabilim continutul unei matrice
        // fara sa citim de la consola
        int[][] matrice1 = {{1,2,3},{4,5,6},{7,8,9}};
        //pentru a doua alocam memorie suficienta
        int[][] matrice2 = new int[3][3];

        //obtinerea matricei transpuse. pur si simplu inversez indecsii
        for(int i=0;i<matrice1.length;i++)
               for(int j=0;j<matrice1[0].length;j++)
                      matrice2[j][i]=matrice1[i][j];

        System.out.println("Elementele matricei initiale sunt:");
        //afisarea unei matrice folosind iteratia pe domeniu
        //prima iteratie percurge matricea linie cu linie
        //a doua iteratie parcurge linia curenta, element cu element
        for(int[] linie:matrice1)
        {
               for(int element:linie)
                      System.out.print(element+" ");
               System.out.println();
        }

        System.out.println("Elementele matricei transpuse sunt:");
        for(int[] linie:matrice2)
        {
               for(int element:linie)
                      System.out.print(element+" ");
               System.out.println();
        }
    }
}
```

Exemplu 11 - Lucrul cu vectori. Cele cinci tipuri de iterații în Java

```
package upg.laborator02.exemplu11;

import java.util.Scanner;
import java.util.Arrays;

public class Principal{
```

```
public static void main(String[] args) {
//declaram vectorul si ii initializam elementele
 int[] vector={7,9,11,17,-5,-2,1,3,22,27,29,33};
//imi propun sa parcurg vectorul in toate modurile in care pot face acest lucru

 //folosind iteratia cu contor
 System.out.println("Parcurgere folosind iteratia cu contor:");
 for(int i=0;i<vector.length;i++)
        System.out.print(vector[i]+" ");
 System.out.println();

 //folosind iteratia cu test initial
 System.out.println("Parcurgere folosind iteratia cu test initial:");
 int i=0;
 while(i<vector.length)
        System.out.print(vector[i++]+" ");
 System.out.println();

 //folosind iteratia cu test final
 System.out.println("Parcurgere folosind iteratia cu test final:");
 i=0;
 do
        System.out.print(vector[i++]+" ");
 while(i<vector.length);
 System.out.println();

 //folosind iteratia pe domeniu
 System.out.println("Parcurgere folosind iteratia pe domeniu:");
 for(int elem:vector)
        System.out.print(elem+" ");
 System.out.println();

 //folosind iteratia cu iterator
 //vectorii avand ca elemente tipuri primitive de date nu sunt considerati
 //direct iterabili - nu se poate utiliza pe ei tipul Iterator
 //asa ca declar un al doilea vector care nu foloseste tipul int ci foloseste
 //un alt tip intreg care este de data aceasta o clasa - Integer

 //o clasa care imbraca un tip primitiv de date se numeste clasa wrapper
 //termenul mai este folosit si in alte contexte in acelasi sens - ceva care
 //imbraca altceva

 //se recomanda ca in aplicatiile de "productie" sa se evite folosirea
 //tipurilor primitive de date
 Integer[] vector2={7,9,11,17,-5,-2,1,3,22,27,29,33};
 //convertesc vectorul intr-o structura direct iterabila
 Iterable<Integer> vector3 = Arrays.asList(vector2);
 //declar iteratorul corespunzator
 Iterator<Integer> it=vector3.iterator();
 System.out.println("Parcurgere folosind iteratia cu iterator propriu-zis:");
 //cat timp mai exista un element urmator
 while(it.hasNext())
 {
```

```
                //trec la elementul urmator
                Integer elem=it.next();
                //afisez ce am gasit acolo
                System.out.print(elem+" ");
         }
         System.out.println();
        }
}
```

Exemplu 12 - Lucrul cu vectori. Căutarea binară folosind metoda *binarySearch* existentă în clasa / biblioteca *Arrays*

```
package upg.laborator02.exemplu12;

import java.util.Scanner;
import java.util.Arrays;

public class Principal{

        public static void main(String[] args) {
         //declaram vectorul si ii initializam elementele
         //elementele trebuie sa fie in ordine crescatoare pentru a putea aplica
         //cautarea binara
         int[] vector={-5,-2,1,3,7,9,11,17,22,27,29,33};
         int x;
         //declaram un obiect nou de tip Scanner pentru citirea de la consola
         Scanner intrare=new Scanner(System.in);
         //citim valoarea cautata
         System.out.println("Introduceti valoarea elementului cautat:");
         x=intrare.nextInt();
         //nu mai am nimic de citit. pot sa inchid intrarea
         intrare.close();
         if(Arrays.binarySearch(vector,x)<0)
                System.out.println("Elementul cautat nu a fost gasit in vector!");
         else
                  System.out.println("Elementul  cautat  a  fost  gasit  la  indexul:  "  +
Arrays.binarySearch(vector,x));
        }
}
```

Exemplu 13 - Lucrul cu vectori. Umplerea unui vector cu valori folosind metoda *fill* existentă în clasa / biblioteca *Arrays*

```
package upg.laborator02.exemplu13;

import java.util.Scanner;
import java.util.Arrays;

public class Principal{
```

```
        public static void main(String[] args) {
        //declaram vectorul si ii alocam memorie suficienta pentru 10 elemente
        int[] vector=new int[10];
        //umplu tot vectorul cu valoarea 55
        Arrays.fill(vector, 55);
        System.out.println("Elementele vectorului, dupa prima umplere sunt:");
        //o parcurgere. aici pot sa folosesc iteratia pe domeniu
        for(int x:vector)
        {
                System.out.print(x+" ");
        }
        //umplu vectorul, de la pozitia 4 pana la pozitia 7, cu valoarea 99
        Arrays.fill(vector, 4, 7, 99);
        System.out.println("Elementele vectorului, dupa prima umplere sunt:");
        //o parcurgere. aici pot sa folosesc iteratia pe domeniu
        for(int x:vector)
        {
                System.out.print(x+" ");
        }
        }
}
```

Exemplu 14 - Lucrul cu vectori. Compararea conținutului a doi vectori folosind metoda *equals* existentă în clasa / biblioteca *Arrays*

```
package upg.laborator02.exemplu14;

import java.util.Scanner;
import java.util.Arrays;

public class Principal{

        public static void main(String[] args) {
         int[] vector1={7,9,11,17,-5,-2,1,3,22,27,29,33};
         int[] vector2={7,9,11,17,-5,-2,1,3,22,27,29,33};
         int[] vector3={7,9,11,17,-5,-2,1,3,22,25,29,33};
         System.out.println("vector1 si vector2 contin aceleasi date? " +
                Arrays.equals(vector1, vector2));
         System.out.println("vector1 si vector3 contin aceleasi date? " +
                Arrays.equals(vector1, vector3));
        }
}
```

Exemplu 15 - Lucrul cu clase proprii. Un exemplu de utilizare a unei clase

Clasa driver (clasa principala):

```
package upg.java.clase01;

public class Main {

	public static void main(String[] args) {
		//pun codul propriu-zis intr-o functie separata ca sa dau ocazia
		//colectorului de gunoaie sa elimine obiectele alocate de mine
		//in acest mod demonstrez cand intra in actiune metoda finalize
		testFinalize();
		//sugerez colectorului de gunoaie ca ar fi timpul sa isi faca treaba
		//(daca nu fac asta, nu se oboseste, pentru ca am consumat prea putina
		//memorie in acest program si nu simte nevoia sa faca curat)
		System.gc();
	}

	public static void testFinalize() {
		//demonstrez utilizarea atributului static
		Student.universitate="Universitatea Petrol-Gaze din Ploiesti";
		//un obiect creat cu ajutorul constructorului implicit
		Student student1=new Student();
		//exemplu de utilizare a setterilor (mutatorilor)
		student1.setNume("Stroescu");
		student1.setPrenume("Maria");
		student1.setGrupa("50998");
		//un obiect creat cu ajutorul celui de-al doilea constructor
		Student student2=new Student("Ionescu", "Emanuel");
		//un obiect creat cu ajutorul celui de-al treilea constructor
		Student student3=new Student("Popescu", "Marcela","50999");
		//exemple de utilizare a diverselor metode
		//semestrul si numarul disciplinei sunt indecsi, deci,
		//primul semestru este semestrul cu numarul 0
		//si prima disciplina este disciplina cu numarul 0
		//spun ca am sase semestre
		student1.setNumarSemestre(6);
		//spun ca pe primul semestru am 5 discipline
		student1.setNumarDiscipline(0, 5);
		//pun notele la aceste cinci discipline
		student1.setNota(0, 0, 10.0);
		student1.setNota(0, 1, 9.0);
		student1.setNota(0, 2, 7.0);
		student1.setNota(0, 3, 7.0);
		student1.setNota(0, 4, 9.0);
		//trisez putin, ca sa nu mai scriu mult text.
		//ma folosesc de faptul ca Java, atunci cand ii ceri sa "tipareasca" un
		//obiect daca are in obiectul respectiv o metoda toString() o apeleaza
```

```
                //pe aceea (daca nu are o metoda toString(0, tipareste pur si simplu
                //identificatorul obiectului, informatie inutila in cazul de fata)
                System.out.println(student1);
                System.out.println("a luat in semestrul 1 notele "+student1.getNote(0));
                System.out.println("si are media "+student1.getMediaString(0));
        }
}
```

Clasa Student (o clasa utilizata in clasa driver):

```
package upg.java.clase01;

//o clasa
public class Student {
        //cateva atribute
        String nume, prenume, grupa;
        //un vector de vectori care sa contina notele
        Double[][] note;
        //un atribut static
        static String universitate;

        //constructorul implicit
        public Student() {
                nume="";
                prenume="";
                grupa="";
        }

        //un constructor cu doi parametri
        public Student(String nume, String prenume) {
                this.nume=nume;
                this.prenume=prenume;
                grupa="";
        }

        //un constructor cu trei parametri
        public Student(String nume, String prenume, String grupa) {
                this.nume=nume;
                this.prenume=prenume;
                this.grupa=grupa;
        }

        //o metoda care sa seteze numarul de semestre
        public void setNumarSemestre(Integer numarSemestre) {
                note=new Double[numarSemestre][];
        }

        //o metoda care sa seteze numarul de discipline dintr-un anumit semestru
        public void setNumarDiscipline(Integer semestru, Integer numarDiscipline) {
                note[semestru]=new Double[numarDiscipline];
                for(Integer disciplina=0;disciplina<note[semestru].length;disciplina++)
                        note[semestru][disciplina]=0.0;
```

```
	}

	//o metoda care sa seteze/modifice nota la o anumita disciplina dintr-un anumit
	//semestru
	public void setNota(Integer semestru, Integer numarDisciplina, Double nota) {
		note[semestru][numarDisciplina]=nota;
	}

	//o metoda care sa returneze nota la o anumita disciplina dintr-un anumit
	//semestru
	public Double getNota(Integer semestru, Integer numarDisciplina) {
		return note[semestru][numarDisciplina];
	}

	//o metoda care sa returneze toate notele dintr-un anumit semestru
	public String getNote(Integer semestru) {
		String aux="";
		//observati un mod de utilizare a metodei format din clasa String
		for(Integer disciplina=0;disciplina<note[semestru].length;disciplina++)
			aux+=String.format("%4.2f",note[semestru][disciplina])+" ";
		return aux;
	}

	//o metoda care sa returneze media notelor dintr-un anumit semestru ca valoare
	//numerica
	public Double getMedia(Integer semestru) {
		Double medie=0.0;
		for(Integer disciplina=0;disciplina<note[semestru].length;disciplina++)
			medie+=note[semestru][disciplina];
		return medie/note[semestru].length;
	}

	//o metoda care sa returneze media notelor dintr-un anumit semestru ca string
	//formatat - folosesc ca sa afisez frumos media cu doua zecimale
	public String getMediaString(Integer semestru) {
		return String.format("%4.2f",getMedia(semestru));
	}

	//o metoda toString care sa returneze concatenat atributele principale ale
	//obiectului atunci cand cineva incearca sa afiseze direct obiectul
	public String toString() {
		return "Studentul(a) "+nume+" "+prenume+" de la universitatea
"+universitate+", grupa "+grupa;
	}

	//un getter (un accesor, in terminologie Java) pentru nume
	public String getNume() {
		return nume;
	}

	//un seetter (un mutator, in terminologie Java) pentru nume
	public void setNume(String nume) {
		this.nume = nume;
	}
```

```
	//un getter (un accesor, in terminologie Java) pentru prenume
	public String getPrenume() {
		return prenume;
	}

	//un seetter (un mutator, in terminologie Java) pentru prenume
	public void setPrenume(String prenume) {
		this.prenume = prenume;
	}

	//un getter (un accesor, in terminologie Java) pentru grupa
	public String getGrupa() {
		return grupa;
	}

	//un seetter (un mutator, in terminologie Java) pentru grupa
	public void setGrupa(String grupa) {
		this.grupa = grupa;
	}

	protected void finalize() {
		System.out.println("in acest moment au fost eliminat din memorie un obiect
din clasa Student");
	}
}
```

Exemplu 16 - Lucrul cu clase proprii. Un exemplu de utilizare a mostenirii

Clasa driver (clasa principala):

```
package upg.java.clase02;

public class Main {

	public static void main(String[] args) {
		//demonstrez utilizarea atributului static
		Student.universitate="Universitatea Petrol-Gaze din Ploiesti";
		//un obiect de tip persoana creat cu ajutorul constructorului implicit
		Persoana persoana1=new Persoana();
		//exemplu de utilizare a setterilor (mutatorilor)
		persoana1.setNume("Stroescu");
		persoana1.setPrenume("Maria");
		//un obiect de tip Student creat cu ajutorul constructorului implicit
		Student student1=new Student();
		student1.setNume("Micsunescu");
		student1.setPrenume("Ionica");
		student1.setGrupa("50998");
		//un obiect de tip Student creat cu ajutorul celui de-al doilea constructor
		Student student2=new Student("Ionescu", "Emanuel");
		//un obiect de tip Student creat cu ajutorul celui de-al treilea constructor
```

```
		Student student3=new Student("Popescu", "Marcela","1801231999999");
		//un obiect de tip Student creat cu ajutorul celui de-al patrulea
constructor
		Student student4=new Student("Popescu", "Luiza","1801231999999","50789");
		//un obiect de tip Student creat cu ajutorul celui de-al cincilea
constructor
		Student student5=new Student("Popescu",
"Aniela","1801231999999","50890","6789");
		//exemple de utilizare a diverselor metode
		//semestrul si numarul disciplinei sunt indecsi, deci,
		//primul semestru este semestrul cu numarul 0
		//si prima disciplina este disciplina cu numarul 0
		//spun ca am sase semestre
		student5.setNumarSemestre(6);
		//spun ca pe primul semestru am 5 discipline
		student5.setNumarDiscipline(0, 5);
		//pun notele la aceste cinci discipline
		student5.setNota(0, 0, 10.0);
		student5.setNota(0, 1, 9.0);
		student5.setNota(0, 2, 7.0);
		student5.setNota(0, 3, 7.0);
		student5.setNota(0, 4, 9.0);
		//ma folosesc de toString() ca sa scriu mai putin cod.
		System.out.println(student5);
		System.out.println("a luat in semestrul 1 notele "+student5.getNote(0));
		System.out.println("si are media "+student5.getMediaString(0));
	}
}
```

Clasa Persoana (superclasa utilizata in clasa driver; va fi mostenita de subclasa Student):

```
package upg.java.clase02;

public class Persoana {
	private String nume, prenume, cnp;

	//constructorul implicit pentru clasa Persoana
	public Persoana() {
		nume="";
		prenume="";
		cnp="";
	}

	//un constructor explicit cu doi parametri pentru clasa Persoana
	public Persoana(String nume, String prenume) {
		this.nume=nume;
		this.prenume=prenume;
		this.cnp="";
	}

	//un constructor explicit cu trei parametri pentru clasa Persoana
```

```
	public Persoana(String nume, String prenume, String cnp) {
		this.nume=nume;
		this.prenume=prenume;
		this.cnp=cnp;
	}

	//getter-ul pentru nume
	public String getNume() {
		return nume;
	}

	//setter-ul pentru nume
	public void setNume(String nume) {
		this.nume = nume;
	}

	//getter-ul pentru prenume
	public String getPrenume() {
		return prenume;
	}

	//setter-ul pentru prenume
	public void setPrenume(String prenume) {
		this.prenume = prenume;
	}

	//getter-ul pentru cnp
	public String getCnp() {
		return cnp;
	}

	//setter-ul pentru cnp
	public void setCnp(String cnp) {
		this.cnp = cnp;
	}

	//supraincarcarea metodei toString(), pentru a se adapta la aceasta clasa
	public String toString() {
		return "Persoana "+nume+" "+prenume;
	}
}
```

Clasa Student (subclasa utilizata in clasa driver; mosteneste superclasa Persoana):

```
package upg.java.clase02;

public class Student extends Persoana {
	private Double[][] note;
	private String grupa, nrMatricol;
	public static String universitate;

	//constructorul implicit pentru clasa Student
```

```
	//preia constructorul implicit din superclasa (Persoana) si ii adauga
	//functionalitate
	public Student() {
		super();
		grupa="";
		nrMatricol="";
	}

	//un constructor explicit cu doi parametri pentru clasa Student
	//preia constructorul cu doi parametri din superclasa (Persoana) si ii adauga
	//functionalitate
	public Student(String nume, String prenume) {
		super(nume, prenume);
		grupa="";
		nrMatricol="";
	}

	//un constructor explicit cu trei parametri pentru clasa Student
	//preia constructorul cu trei parametri din superclasa (Persoana) si ii adauga
	//functionalitate
	public Student(String nume, String prenume, String cnp) {
		super(nume, prenume, cnp);
		grupa="";
		nrMatricol="";
	}

	//un constructor explicit cu patru parametri pentru clasa Student
	//preia constructorul cu trei parametri din superclasa (Persoana) si ii adauga
	//functionalitate
	public Student(String nume, String prenume, String cnp, String grupa) {
		super(nume, prenume, cnp);
		this.grupa=grupa;
		nrMatricol="";
	}

	//un constructor explicit cu cinci parametri pentru clasa Student
	//preia constructorul cu trei parametri din superclasa (Persoana) si ii adauga
	//functionalitate
	public Student(String nume, String prenume, String cnp, String grupa, String
nrMatricol) {
		super(nume, prenume, cnp);
		this.grupa=grupa;
		this.nrMatricol=nrMatricol;
	}

	//o metoda care sa seteze numarul de semestre
	public void setNumarSemestre(Integer numarSemestre) {
		note=new Double[numarSemestre][];
	}

	//getter-ul pentru grupa
	public String getGrupa() {
		return grupa;
	}
```

```java
//setter-ul pentru grupa
public void setGrupa(String grupa) {
	this.grupa = grupa;
}

//getter-ul pentru nrMatricol
public String getNrMatricol() {
	return nrMatricol;
}

//setter-ul pentru nrMatricol
public void setNrMatricol(String nrMatricol) {
	this.nrMatricol = nrMatricol;
}

//o metoda care sa seteze numarul de discipline dintr-un anumit semestru
public void setNumarDiscipline(Integer semestru, Integer numarDiscipline) {
	note[semestru]=new Double[numarDiscipline];
	for(Integer disciplina=0;disciplina<note[semestru].length;disciplina++)
		note[semestru][disciplina]=0.0;
}

//o metoda care sa seteze/modifice nota la o anumita disciplina dintr-un anumit
//semestru
public void setNota(Integer semestru, Integer numarDisciplina, Double nota) {
	note[semestru][numarDisciplina]=nota;
}

//o metoda care sa returneze nota la o anumita disciplina dintr-un anumit
//semestru
public Double getNota(Integer semestru, Integer numarDisciplina) {
	return note[semestru][numarDisciplina];
}

//o metoda care sa returneze toate notele dintr-un anumit semestru
public String getNote(Integer semestru) {
	String aux="";
	//observati un mod de utilizare a metodei format din clasa String
	for(Integer disciplina=0;disciplina<note[semestru].length;disciplina++)
		aux+=String.format("%4.2f",note[semestru][disciplina])+" ";
	return aux;
}

//o metoda care sa returneze media notelor dintr-un anumit semestru ca valoare
//numerica
public Double getMedia(Integer semestru) {
	Double medie=0.0;
	for(Integer disciplina=0;disciplina<note[semestru].length;disciplina++)
		medie+=note[semestru][disciplina];
	return medie/note[semestru].length;
}

//o metoda care sa returneze media notelor dintr-un anumit semestru ca string
```

```
	//formatat - folosesc ca sa afisez frumos media cu doua zecimale
	public String getMediaString(Integer semestru) {
		return String.format("%4.2f",getMedia(semestru));
	}

	//o metoda toString care sa returneze concatenat atributele principale ale
	//obiectului atunci cand cineva incearca sa afiseze direct obiectul
	public String toString() {
		return "Studentul(a) "+getNume()+" "+getPrenume()+" de la universitatea
"+universitate+", grupa "+getGrupa()+", nr. matricol "+getNrMatricol();
	}
}
```

Exemplu 17 - Lucrul cu Swing. Un exemplu de aplicație cu interfață grafică

```
package upg.java.clase03;

//bibloteca AWT, clasa Container - asta folosesc ca sa pot crea un container de
//componente; AWT e prescurtarea de la Abstract Window Toolkit - o biblioteca pentru
//lucrul cu ferestre
import java.awt.Container;
//bibloteca AWT, clasa Dimension - asta folosesc ca sa ma pot juca cu diemnsiunile
//componentelor
import java.awt.Dimension;
//bibloteca AWT, clasa EventQueue - asta folosesc ca sa pot declansa unul sau mai //multe
evenimente in succesiune
import java.awt.EventQueue;
//biblioteca Swing, clasa JFrame - asta folosesc pentru a construi o fereastra
import javax.swing.JFrame;
//biblioteca Swing, clasa JButton - asta folosesc pentru a construi o componenta //oarecare
import javax.swing.JComponent;
//biblioteca Swing, clasa JButton - asta folosesc pentru a construi o infatisare de
//componenta
import javax.swing.GroupLayout;
//biblioteca Swing, clasa JButton - asta folosesc pentru a construi o eticheta
import javax.swing.JLabel;
//biblioteca Swing, clasa JLabel - asta folosesc pentru a construi un buton
import javax.swing.JButton;
//biblioteca Swing, clasa JTextField - asta folosesc pentru a construi un textbox
import javax.swing.JTextField;


//clasa driver va mosteni clasa JFrame - e o modalitate de a face aplicatii cu //interfata
grafica
public class Main extends JFrame {

	//constructorul implicit al clasei driver. ma folosesc de el ca sa configurez
	//fereastra aplicatiei
	public Main() {

	initUI();
```

```
}

//am concentrat tot ce tine de configurarea ferestrei intr-o functie
    private void initUI() {
            //o eticheta noua pe care scrie "a="
            JLabel eticheta1 = new JLabel("a=");
            //o eticheta noua pe care scrie "b="
            JLabel eticheta2 = new JLabel("b=");
            //o eticheta noua pe care scrie "a+b="
            JLabel eticheta3 = new JLabel("a+b=");
            //un textbox nou pe care nu scrie nimic
            JTextField text1 = new JTextField("");
            //un textbox nou pe care nu scrie nimic
            JTextField text2 = new JTextField("");
            //un textbox nou pe care nu scrie nimic
            JTextField text3 = new JTextField("");
            //un buton nou pe care scrie "Adunare"
            JButton buton1 = new JButton("Adunare");
            //cand se apasa pe buton va fi apelata metoda adunare cu parametrii
            //text1, text2, text3 (ca sa stie de unde sa citeasca si unde sa scrie)
     buton1.addActionListener((event) -> adunare(text1, text2, text3));
     //ii pun si un indiciu butonului
     buton1.setToolTipText("Apasa aici pentru adunare.");
     //un buton nou pe care scrie "Iesire"
     JButton buton2 = new JButton("Iesire");
     //cand se apasa pe buton va fi apelata metoda iesire()
     buton2.addActionListener((event) -> iesire());
     //ii pun si un indiciu butonului
     buton2.setToolTipText("Apasa aici pentru iesire.");

     //stabilesc o dimensiune prestabilita pentru etichete
     Dimension marimeEticheta=new Dimension(50,20);
     //stabilesc o dimensiune prestabilita pentru textbox-uri
     Dimension marimeText=new Dimension(150,20);
     //stabilesc o dimensiune prestabilita pentru butoane
     Dimension marimeButon=new Dimension(100,20);
     //setez dimensiunea etichetelor
    eticheta1.setMaximumSize(marimeEticheta);
    eticheta2.setMaximumSize(marimeEticheta);
    eticheta3.setMaximumSize(marimeEticheta);
    //setez dimensiunea textbox-urilor
    text1.setMaximumSize(marimeText);
    text2.setMaximumSize(marimeText);
    text3.setMaximumSize(marimeText);
    //setez dimensiunea butoanelor
    buton1.setMaximumSize(marimeButon);
    buton2.setMaximumSize(marimeButon);
    //apelez functia creazaAspect care va configura modul in care sunt
    //asezate componentele in fereastra
creazaAspect(eticheta1, text1, eticheta2, text2, eticheta3, text3, buton1, buton2);
    //titlul noii ferestre
    setTitle("O fereastra facuta cu Swing ...");
    //dimensiunea noii ferestre
     setSize(400, 200);
```

```
    //locatia (locul in care va aparea pe ecran) noii ferestre
     setLocationRelativeTo(null);
     //ce sa se intample cu toata aplicatia atunci cand inchid fereastra
     setDefaultCloseOperation(EXIT_ON_CLOSE);
}

    private void creazaAspect(JComponent... arg) {
     //e nevoie de un caontainer pentru componente. il creez
    Container panel = getContentPane();
    //creez un aspect pentru acest container
     GroupLayout aspectGrup = new GroupLayout(panel);
     //spun ca noul container are aspectul pe care tocmai l-am produs
     panel.setLayout(aspectGrup);
     //spun ca in noul aspect se lasa automat spatii intre componente
     aspectGrup.setAutoCreateContainerGaps(true);
     //grupez componentele pe orizontala (label1 cu text1 si pauza
     //intre ele, label2 cu text2 si pauza intre ele samd)
     //createParallelGrup creeaza un grup in care componentele
     //sunt paralele (aici pe orizontala)
     aspectGrup.setHorizontalGroup(aspectGrup.createParallelGroup()
                   //createSequentialGrup creeaza un grup in care componentele
                   //urmeaza una dupa alta (aici pe orizontala)
                   .addGroup(aspectGrup.createSequentialGroup()
                           .addComponent(arg[0])
                           .addGap(10)
                           .addComponent(arg[1]))
                   //createSequentialGrup creeaza un grup in care componentele
                   //urmeaza una dupa alta (aici pe orizontala)
                   .addGroup(aspectGrup.createSequentialGroup()
                           .addComponent(arg[2])
                           .addGap(10)
                           .addComponent(arg[3]))
                   //createSequentialGrup creeaza un grup in care componentele
                   //urmeaza una dupa alta (aici pe orizontala)
                   .addGroup(aspectGrup.createSequentialGroup()
                           .addComponent(arg[4])
                   .addGap(10)
                   .addComponent(arg[5]))
                   //createSequentialGrup creeaza un grup in care componentele
                   //urmeaza una dupa alta (aici pe orizontala)
                   .addGroup(aspectGrup.createSequentialGroup()
                  .addGap(35)
                  .addComponent(arg[6]))
                   //createSequentialGrup creeaza un grup in care componentele
                   //urmeaza una dupa alta (aici pe orizontala)
                   .addGroup(aspectGrup.createSequentialGroup()
                           .addGap(35)
                  .addComponent(arg[7]))
                   );
    //grupez componentele pe orizontala (grupul cu label1 si text1, dupa grupul cu
    //label2 si text2 samd)
    //createSequentialGrup creeaza un grup in care componentele
    //urmeaza una dupa alta (aici pe verticala)
    //la verticala gruparea este exact inversa decat la orizontala
```

```
    //(unde era grupare secventiala aici e paralela si invers)
     aspectGrup.setVerticalGroup(aspectGrup.createSequentialGroup()
                    //createParallelGrup creeaza un grup in care componentele
              //sunt paralele (aici pe verticala)
                    .addGroup(aspectGrup.createParallelGroup()
                           .addComponent(arg[0])
                           .addComponent(arg[1]))
                    //createParallelGrup creeaza un grup in care componentele
              //sunt paralele (aici pe verticala)
                    .addGroup(aspectGrup.createParallelGroup()
                           .addComponent(arg[2])
                           .addComponent(arg[3]))
                    //createParallelGrup creeaza un grup in care componentele
              //sunt paralele (aici pe verticala)
                    .addGroup(aspectGrup.createParallelGroup()
                           .addComponent(arg[4])
                    .addComponent(arg[5])
                    .addGap(40))
                    //createParallelGrup creeaza un grup in care componentele
              //sunt paralele (aici pe verticala)
                    .addGroup(aspectGrup.createParallelGroup()
                           .addComponent(arg[6]))
                    //createParallelGrup creeaza un grup in care componentele
              //sunt paralele (aici pe verticala)
                    .addGroup(aspectGrup.createParallelGroup()
                        .addComponent(arg[7]))
                    );
}

    private void adunare(JTextField... arg){
           //declar variabilele
           double a, b, s;
           //incerc sa fac citirea
           try {
                  //il preiau pe a din primul textbox
                  //e un pic complicat in sensul ca trebuie sa iau textul din
                  //textbox, sa il convertesc in string, dupa care sa convertesc
                  //string-ul in double
                  a=Double.parseDouble((arg[0]).getText().toString());
                  //il preiau pe b din al doilea textbox
                  b=Double.parseDouble((arg[1]).getText().toString());
                  //fac adunarea
                  s=a+b;
                  //scriu rezultatul in al treilea textbox
                  arg[2].setText(String.format("%5.7f",s));
           }
           //daca nu a mers citirea pentru ca in primul si al doilea textbox nu
           //erau scrise niste numere
           //lucrez grosolan in sensul ca nu fac diferenta intre tipurile de
           //exceptii, dar aici nu conteaza
           catch(Exception e) {
                  ((JTextField)arg[2]).setText("a sau b nu este numar");
           }
    }
```

```
        private void iesire(){
                //opresc programul
                System.exit(0);
        }

    //metoda main
        public static void main(String[] args) {
                //folosesc programatorul de evenimente pentru a creea un fir de
                //executie nou in care sa ruleze fereastra grafica.
                //in aplicatia asta s-ar putea si fara el, dar in general aceasta este
                //metoda corecta.
                //daca lucrez in acest mod (cu un fir de executie pentru aplicatie
                //si un altul pentru interfata grafica) ma asigur ca cele doua nu se
                //"jeneaza" intre ele
                 EventQueue.invokeLater(() -> {
                //un obiect nou din clasa Main
            Main fereastra = new Main();
            //fac noua fereastra vizibila
            fereastra.setVisible(true);
         });
    }
}
```

Exemplu 18 - Lucrul cu clase generice

Clasa driver (clasa principala):

```
package upg.java.clase03;

import java.util.Scanner;

public class Main{

    //metoda main
        public static void main(String[] args) {
                EcuatieGrad2<Double> e1=new EcuatieGrad2<Double>();
                Scanner intrare=new Scanner(System.in);
                System.out.println("Aplicatie pentru rezolvarea ecuatie de gradul 2. Va rog
sa introduceti coeficientii ecuatiei ax^2+bx+c=0:");
                System.out.print("a=");
                e1.setA(intrare.nextDouble());
                System.out.print("b=");
                e1.setB(intrare.nextDouble());
                System.out.print("c=");
                e1.setC(intrare.nextDouble());
                e1.rezolvare();
                //fac afisare folosind metoda toString()
                System.out.print(e1);
        }
}
```

Clasa EcuatieGrad2 (clasa utilizată):

```
package upg.java.clase04;

public class EcuatieGrad2<TipDeDate> {
	//coeficientii ecuatiei
	TipDeDate a, b, c;
	//trei stringuri folosite pentru afisare
	String mesaj, solutie1, solutie2;
	//o variabila in care retin ce s-a intamplat la rezolvare
	Integer cod;

	//un constructor implicit
	public EcuatieGrad2() {
		cod=-1;
		mesaj="";
		solutie1="";
		solutie2="";
	}

	public Integer rezolvare()
	{
		//folosesc conversii explicite de la tipul generic la tipul Double
		if((Double)a == 0 && (Double)b == 0 && (Double)c == 0)
		{
			mesaj = "Ecuatia este o identitate (0=0). Nu necesita rezolvare.";
			cod = 4;
			//ma folosesc de return ca sa ies din rezolvare
			return 0;
		}
		if((Double)a==0 && (Double)b==0 && (Double)c!=0)
		{
			mesaj = "Ecuatia este imposibila ("+((Double)c).toString()+"=0). Nu
necesita rezolvare.";
			cod = 3;
			return 0;
		}
		if((Double)a==0 && (Double)b!=0)
		{
			mesaj = "Ecuatia este o ecuatie de gradul 1.";
			Double temp = -((Double)c)/((Double)b);
			solutie1 = temp.toString();
			cod = 2;
			return 0;
		}
		Double delta=(Double)b*(Double)b-4*(Double)a*(Double)c;
		if(delta<0)
		{
			mesaj="Ecuatia este o ecuatie de gradul 2 cu solutii complexe
conjugate.";
			Double temp1 = -(Double)b / (2*(Double)a);
			Double temp2 = Math.sqrt(-delta) / (2*(Double)a);
			solutie1 = temp1.toString() + "-i*" + temp2.toString();
```

```
                solutie2 = temp1.toString() + "+i*" + temp2.toString();
                cod = 1;
                return 0;
            }
            else
            {
                mesaj="Ecuatia este o ecuatie de gradul 2 cu solutii reale.";
                Double temp = (-(Double)b - Math.sqrt(delta)) / (2*(Double)a);
                solutie1 = temp.toString();
                temp = (-(Double)b + Math.sqrt(delta)) / (2*(Double)a);
                solutie2 = temp.toString();
                cod = 0;
                return 0;
            }
        }

        //folosesc metoda clasica toString pentru a concatena toate elementele
        //rezolvarii intr-un mesaj
        public String toString() {
            //identitate
            if(cod==4)
                return mesaj;
            //ecuatie imposibila
            if(cod==3)
                return mesaj;
            //ecuatie de gradul 1
            if(cod==2)
                return mesaj + " Solutia unica este: " + solutie1;
            //ecuatie de gradul 2
            return mesaj + " Solutiile ecuatiei sunt: " + solutie1 + " si " + solutie2;
        }

        //mutatorul pentru a
        public void setA(TipDeDate a) {
            this.a = a;
        }

        //mutatorul pentru b
        public void setB(TipDeDate b) {
            this.b = b;
        }

        //mutatorul pentru c
        public void setC(TipDeDate c) {
            this.c = c;
        }

}
```