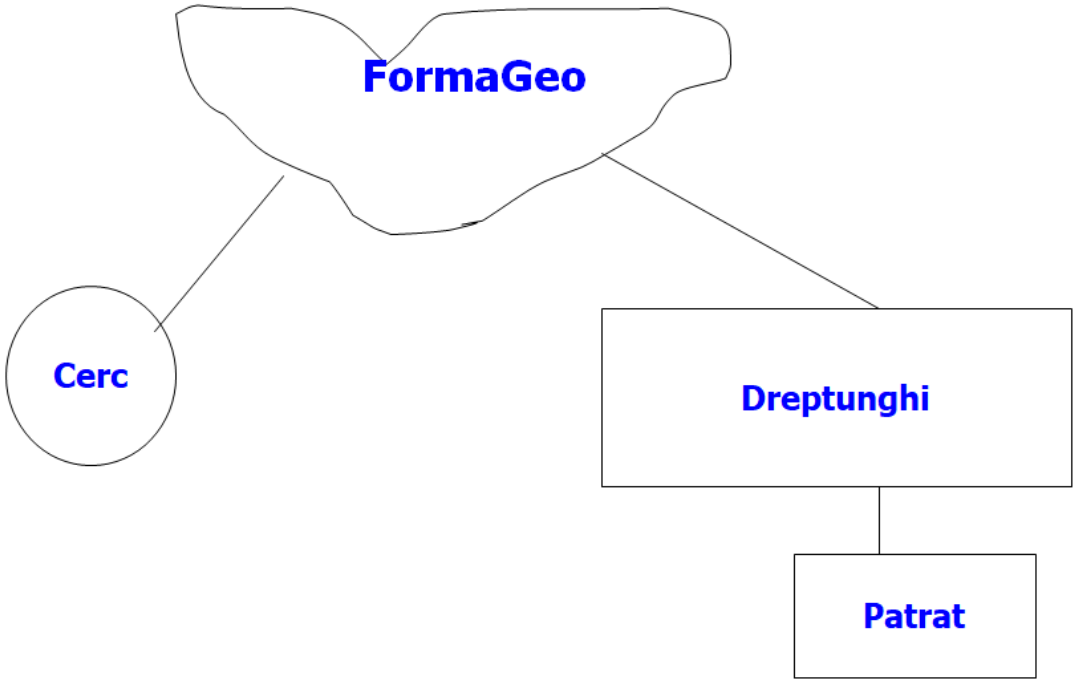
Laborator 12 - Java
Polimorfism

Probleme rezolvate:

Scrieti, compilati si rulati urmatoarele 2 exemple din acest laborator:

Problema rezolvata nr.1. Un exemplu simplu de clasa abstracta este **clasa formelor geometrice**. Din clasa formelor geometrice pot deriva forme specifice cum ar fi: **cerc**, **dreptunghi**. Se poate, apoi, deriva **clasa patratului** ca un caz particular de dreptunghi.



Fisierul-sursa urmator (**FormaGeo.java**) prezinta clasa abstracta FormaGeo.

```
/* Superclasa abstracta pentru forme FormeGeo
 *
 * CONSTRUIREA: nu este permisa, FormaGeo fiind abstracta.
 * Constructorul cu un parametru este necesar ptr. Clasele derivate.
 * ------------------- metode publice ---------------------
 * double arie()        --> Intoarce aria (abstracta)
 * boolean maiMic       --> Compara doua forme dupa arie
 * String toString      --> Metoda uzuala pentru afisare
 */
 abstract class FormaGeo  {
```

```
        private String nume;
        abstract public double arie();
        public FormaGeo(String numeForma)  {
                nume = numeForma;
        }
        final public boolean maiMic(FormaGeo rhs)  {
                return arie() < rhs.arie();
        }
        final public String toString()  {
                return nume + ", avand aria " + arie();
        }
}
```

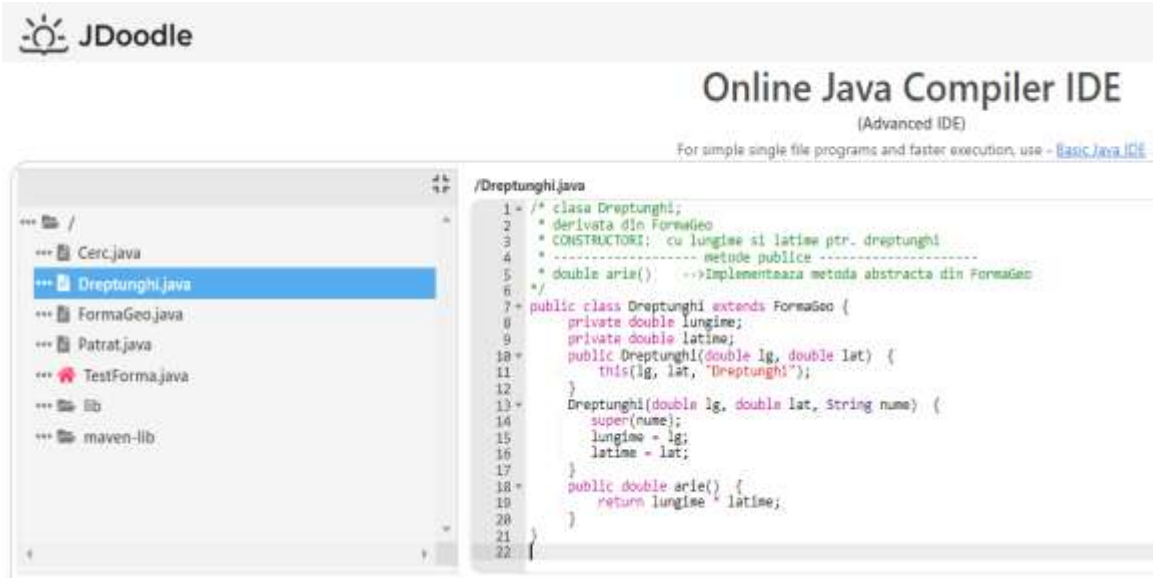
Mai intai, trebuie sa definim subclasele Cerc, Dreptunghi si Patrat. Definirea lor se face distinct in fisiere-sursa separate (**Cerc.java**, **Dreptunghi.java** si **Patrat.java**).

Fisierul-sursa urmator (**Cerc.java**) prezinta clasa **Cerc**.

```
/* clasa Cerc
 * derivata din FormaGeo
 * CONSTRUCTORI: cu raza ptr. cerc
 * ------------------- metode publice ---------------------
 * double arie()          -->Implementeaza metoda abstracta din FormaGeo
 */
public class Cerc extends FormaGeo  {
        private double raza;
        public Cerc(double rad)  {
                super("Cerc");
                raza = rad;
        }
        public double arie()  {
                return Math.PI * raza * raza;}
 }
```

Fisierul-sursa urmator (**Dreptunghi.java**) prezinta clasa **Dreptunghi**.

```
/* clasa Dreptunghi;
 * derivata din FormaGeo
 * CONSTRUCTORI:  cu lungime si latime ptr. dreptunghi
 * ------------------- metode publice ---------------------
 * double arie()          -->     Implementeaza metoda abstracta din FormaGeo
*/
public class Dreptunghi extends FormaGeo {
            private double lungime;
            private double latime;
            public Dreptunghi(double lg, double lat)  {
                        this(lg, lat, "Dreptunghi");
            }
            Dreptunghi(double lg, double lat, String nume)  {
                        super(nume);
                        lungime = lg;
                        latime = lat;
            }
            public double arie()  {
                        return lungime * latime;
            }
}
```



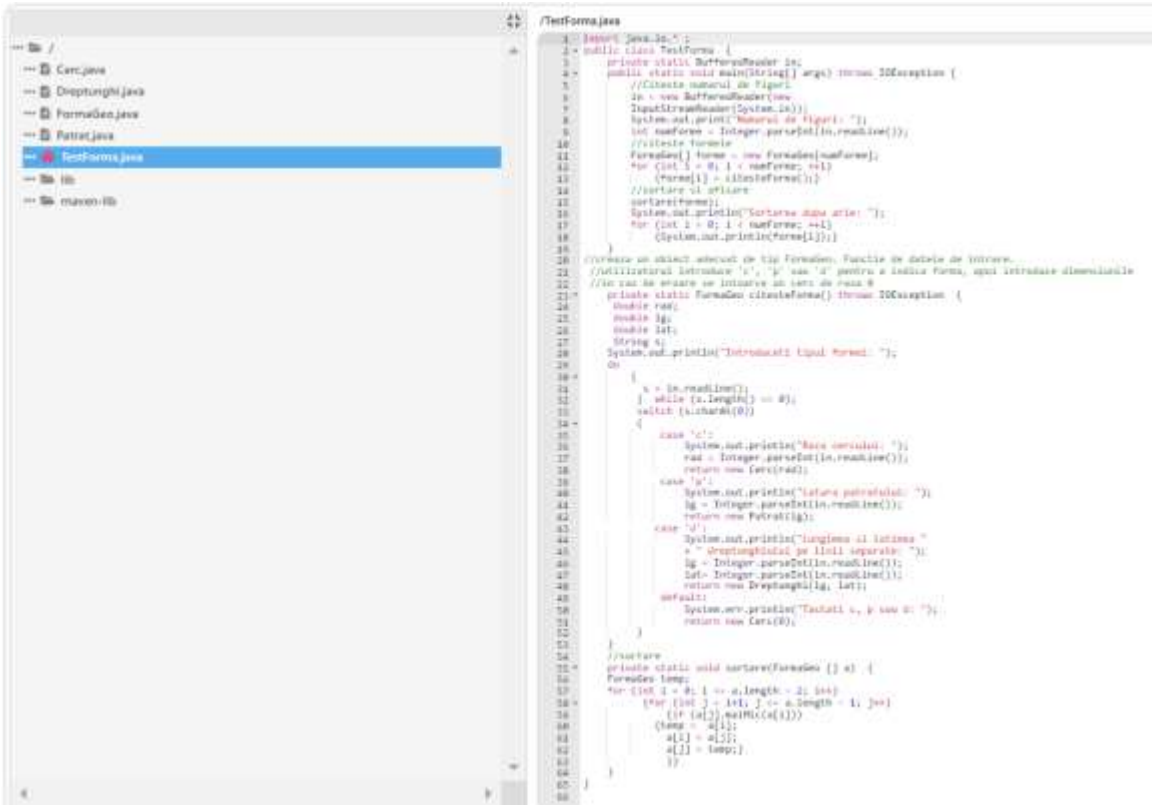
Fisierul-sursa urmator (**Patrat.java**) prezinta clasa **Patrat**.

```
/* clasa Patrat
 * derivata din Dreptunghi
 * CONSTRUCTORI: cu latura ptr. patrat
 * ---------- metode publice ----------
 * double arie()          -->Implementeaza metoda abstracta din FormaGeo
 */
public class Patrat extends Dreptunghi
{
        public Patrat(double latura)
        {
                super(latura, latura, "Patrat");
        }
}
```

Fisierul-sursa (**TestForma.java**) care realizeaza ordonarea si punerea in executie a aplicatiei se prezinta in continuare.

```
import java.io.* ;
class TestForma  {
        private static BufferedReader in;
        public static void main(String[] args) throws IOException {
                //Citeste numarul de figuri
                in = new BufferedReader(new
                InputStreamReader(System.in));
                System.out.print("Numarul de figuri: ");
                int numForme = Integer.parseInt(
                in.readLine());
                //citeste formele
                FormaGeo[] forme = new FormaGeo[numForme];
                for (int i = 0; i < numForme; ++i)
                        {forme[i] = citesteForma();}
                //sortare si afisare
                sortare(forme);
                System.out.println("Sortarea dupa arie: ");
                for (int i = 0; i < numForme; ++i)
                        {System.out.println(forme[i]);}
        }
//creaza un obiect adecvat de tip FormaGeo. Functie de datele de intrare.
 //utilizatorul introduce 'c', 'p' sau 'd' pentru a indica forma, apoi introduce dimensiunile
 //in caz de eroare se intoarce un cerc de raza 0
        private static FormaGeo citesteForma() throws IOException  {
         double rad;
         double lg;
         double lat;
         String s;
        System.out.println("Introduceti tipul formei: ");
        do
                {
                 s = in.readLine();
                 }  while (s.length() == 0);
                 switch (s.charAt(0))
                 {
                        case 'c':
                                System.out.println("Raza cercului: ");
                                rad = Integer.parseInt(in.readLine());
                                return new Cerc(rad);
                        case 'p':
                                System.out.println("Latura patratului: ");
                                lg = Integer.parseInt(in.readLine());
                                return new Patrat(lg);
                       case 'd':
                                System.out.println("Lungimea si latimea "
```

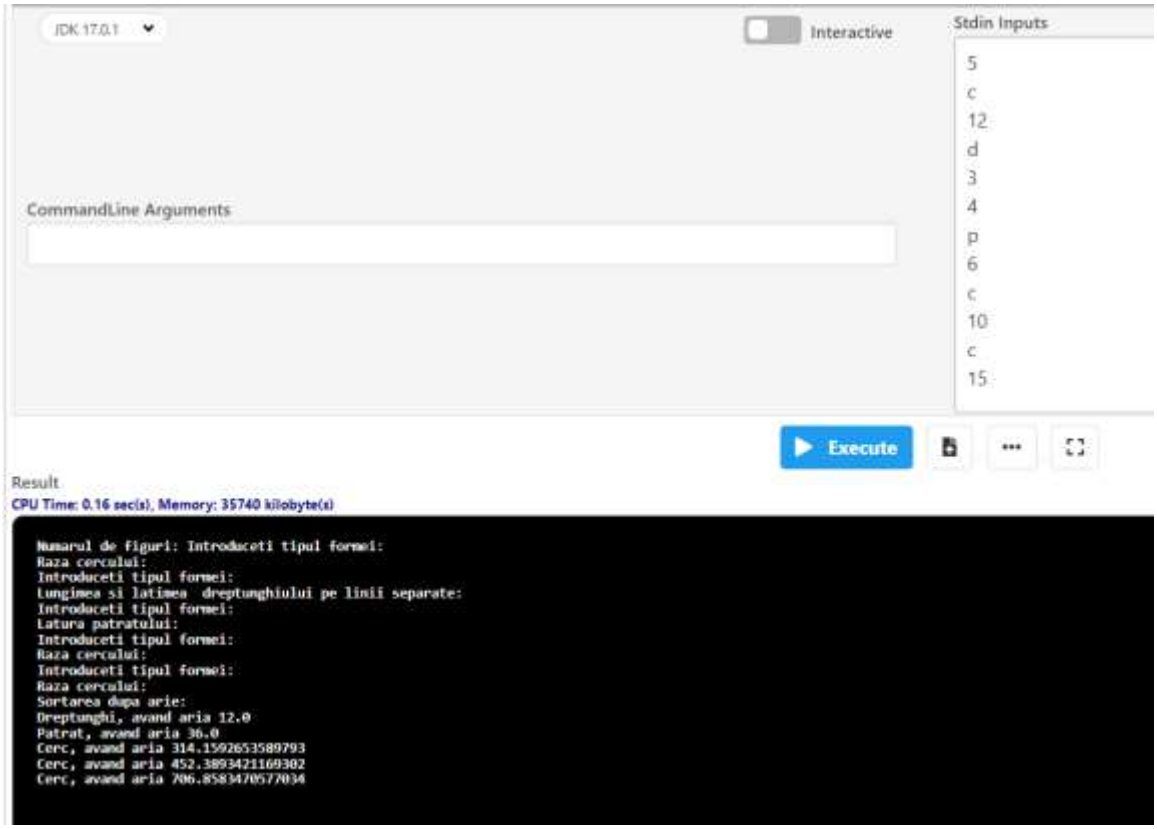
```
                                + " dreptunghiului pe linii separate: ");
                                lg = Integer.parseInt(in.readLine());
                                lat= Integer.parseInt(in.readLine());
                                return new Dreptunghi(lg, lat);
                        default:
                                System.err.println("Tastati c, p sau d: ");
                                return new Cerc(0);
                }
        }
        // ---------- sortare
        private static void sortare(FormaGeo [] a)  {
        FormaGeo temp;
        for (int i = 0; i <= a.length - 2; i++)
            {for (int j = i+1; j <= a.length - 1; j++)
                {if (a[j].maiMic(a[i]))
                    {temp =  a[i];
                     a[i] = a[j];
                     a[j] = temp;}
                    }
                }
            }
}
```



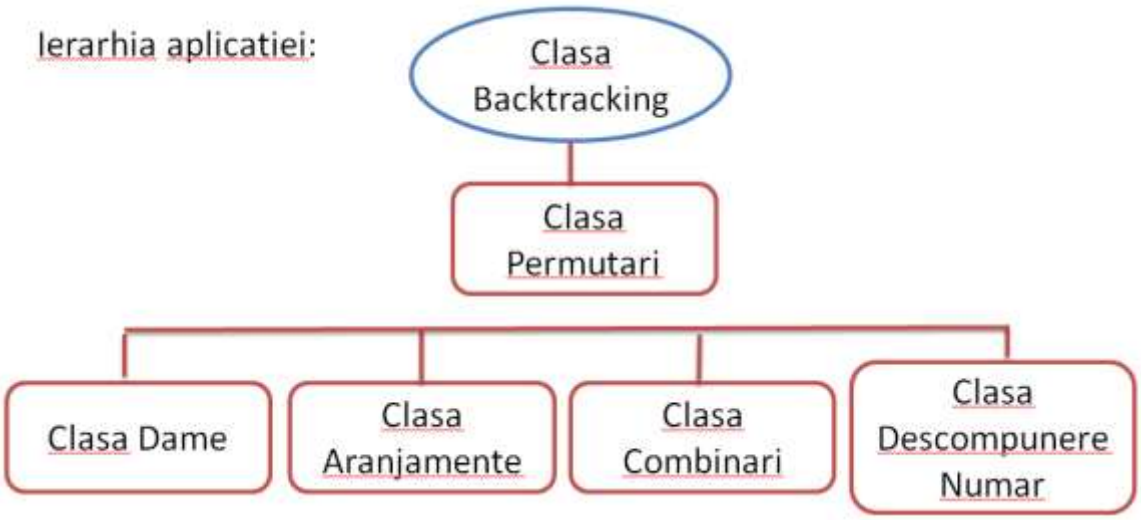
Recomandam utilizarea compilatorului online **https://www.jdoodle.com/online-java-compiler-ide/** (varianta de compilator Java Advanced, in care se poate salva codul programului in clasa dorita).

Rezultatul executiei programului este:
Numarul de figuri: 5
Introduceti tipul formei: c
Raza cercului: 12
Introduceti tipul formei: d
Lungimea si latimea dreptunghiului pe linii separate: 3
4
Introduceti tipul formei: p
Latura patratului: 6
Introduceti tipul formei: c
Raza cercului: 10
Introduceti tipul formei: c
Raza cercului: 15

Sortarea dupa arie:
Dreptunghi, avand aria 12.0
Patrat, avand aria 36.0
Cerc, avand aria 314.1592653589793
Cerc, avand aria 452.3893421169302
Cerc, avand aria 706.8583470577034



Problema rezolvata nr.2. Programul urmator (**testBacktracking.java**) prezinta o clasa abstracta (Backtracking), din care deriva clasa Permutari, din care deriva clasele **Dame**, **Aranjamente**, **Combinari** si **DescompunereNumar**, care se pot folosi (declarand obiecte din clasele respective) pentru rezolvarea problemelor cu ajutorul metodei de programare – Backtracking.



testBacktracking.java

```
1  package testBacktracking;
2  import java.util.*;
3
4  abstract class Backtracking
5  {
6      int[] st; int n,p;
7      abstract void init(int k);
8      abstract boolean Succesor(int k);
9      abstract boolean Valid(int k);
10     abstract boolean solutie(int k);
11     abstract void tipar();
12     void back()
13     {
14             boolean AS;
15             int k=1;
16             st = new int[n+1];
17             init(k);
18             while (k>0)
19             {
20                 do { AS=Succesor(k); } while ( AS && !Valid(k) );
21                 if( AS )
22                     if( solutie(k) )
23                         tipar();
24                     else{
25                         k++; init(k);
26                     }
27                 else k--;
28             }//while
29     }//back
30 }//class Backtracking
31
```

testBacktracking.java

```
31
32  class Permutari extends Backtracking
33  {
34      Permutari(int n)
35      {
36          this.n=n;
37      }//Permutari
38      void init(int k)
39      {
40          st[k]=0;
41      }//init;
42      boolean Succesor(int k)
43      {
44      if ( st[k]<n )
45      { st[k]++; return true; }
46      return false;
47      }//Succesor
48      boolean Valid(int k)
49      {
50          for(int i=1; i<k; i++)
51          if(st[i]==st[k])
52              return false;
53      return true;
54      }//Valid
55      boolean solutie(int k)
56      {
57          return k==n;
58      }//solutie
59      void tipar()
60      {
61          for(int i=1; i<=n; i++)
62          System.out.print(" "+st[i]);
63          System.out.println();
64      }//tipar
65  }//class Permutari
```

testBacktracking.java

```
66
67  class Dame extends Permutari
68  {
69      Dame(int n)
70      {
71          super(n);
72      }//Dame
73      boolean Valid(int k)
74      {
75          for(int i=1; i<k; i++)
76          if( (st[i]==st[k]) || (Math.abs(st[k]-st[i])==Math.abs(k-i)) )
77              return false;
78          return true;
79      }//Valid
80  }//class Dame
```

testBacktracking.java

```
81
82  class Aranjamente extends Permutari
83  {
84      Aranjamente(int n, int p)
85      {
86          super(n);
87          this.p=p;
88      }//Aranjamente
89      boolean solutie(int k)
90      {
91          return k==p;
92      }//solutie
93      void tipar()
94      {
95          for(int i=1; i<=p; i++) System.out.print(" "+st[i]);
96          System.out.println();
97      }//tipar
98  }//class Aranjamente
99
```

testBacktracking.java

```
 99
100 class Combinari extends Permutari
101 {
102     Combinari(int n, int p)
103     {
104         super(n);
105         this.p=p;
106     }//Combinari
107
108     boolean Valid(int k)
109     {
110         for(int i=1; i<k; i++)
111           if(st[i]==st[k]) return false;
112         if(k>1)
113             if(st[k]<st[k-1]) return false;
114         return true;
115     }//Valid
116     boolean solutie(int k)
117     {
118         return k==p;
119     }//solutie
120     void tipar()
121     {
122         for(int i=1; i<=p; i++) System.out.print(" "+st[i]);
123         System.out.println();
124     }//tipar
125 }//class Combinari
126
```

testBacktracking.java

```
126
127 class DescompunereNumar extends Permutari
128 {
129     DescompunereNumar(int n)
130     {
131         super(n);
132     }//DescompunereNumar
133     void init(int k)
134     {
135         st[k]=0;
136     }//init;
137     boolean Succesor(int k)
138     {
139     if ( st[k]<n &&k<=n)
140     { st[k]++; return true; }
141     return false;
142     }//Succesor
143     boolean Valid(int k)
144     {
145         int s=0;
146         for(int i=1; i<=k; i++) s=s+st[i];
147         if(s>n) return false;
148         return true;
149     }//Valid
150     boolean solutie(int k)
151     {
152         int s=0;
153         for(int i=1; i<=k; i++) s=s+st[i];
154         if(s==n) return true;
155         else return false;
156     }//solutie
157     void tipar()
158     {
159         for(int i=1; i<=n; i++)
160             if(i!=n) System.out.print(st[i]+"+");
161             else System.out.print(st[i]);
162         System.out.println();
163     }//tipar
164 }//class DescompunereNumar
165
```

testBacktracking.java

```
165
166 public class testBacktracking {
167
168     private static Scanner input;
169
170     public static void main(String[] args) {
171         System.out.print ("Introduceti numarul de elemente ale multimii A (n) =  ");
172         input = new Scanner(System.in);
173         int n = input.nextInt();
174         int p;
175         System.out.print ("Introduceti numarul p ale multimii A (p) =  ");
176         p = input.nextInt();
177
178         System.out.println(" Permutarile unei multimi cu "+n+" elemente");
179         Backtracking o1 = new Permutari(n);
180         o1.back();
181         System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
182
183         System.out.println(" Asezarea a "+n+" dame pe o tabla de sah");
184         Dame o2 = new Dame(n);
185         o2.back();
186         System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
187
188         System.out.println(" Aranjamentele unei multimi cu "+n+" elemente"+" luate cate "+p);
189         Backtracking o3 = new Aranjamente(n,p);
190         o3.back();
191         System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
192
193         System.out.println(" Combinarile unei multimi cu "+n+" elemente"+" luate cate "+p);
194         Backtracking o4 = new Combinari(n,p);
195         o4.back();
196         System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
197
198         System.out.println(" Descompunerile unui numar "+n+" in suma de numere");
199         Backtracking o5 = new DescompunereNumar(n);
200         o5.back();
201         System.out.println("- - - - - - - - - - - - - - - - - - - - - - - - - - - - -");
202     }
203 }
204
```

Probleme propuse spre rezolvare

Lab12_1: Clasa **Tren**. Se cere sa se modeleze o garnitura de tren. Se va defini in acest scop o clasa **Tren**. Un obiect de tip **Tren** contine mai multe referinte spre obiecte de tip Vagon care sunt pastrate intr-un tablou. Un vagon poate fi de 3 tipuri: **CalatoriA**, **CalatoriB** si **Marfa**. Despre garnitura de tren si vagoane mai cunoastem urmatoarele:

- un tren poate contine maxim 15 vagoane, indiferent de tipul vagoanelor.
- un vagon de tip **CalatoriA**
  - are capacitatea de 40 pasageri si 300 colete postale.
  - furnizeaza doua servicii pentru deschiderea, respectiv inchiderea automata a usilor.
- un vagon de tip **CalatoriB**
  - are capacitatea de 50 pasageri si 400 colete postale.
  - furnizeaza doua servicii pentru deschiderea, respectiv inchiderea automata a usilor.
  - fiind un vagon mai nou, furnizeaza un serviciu automat pentru blocarea geamurilor.
- un vagon de tip **Marfa**
  - are capacitatea de 400 colete postale.
  - nu furnizeaza servicii pentru deschiderea, respectiv inchiderea automata a usilor, aceste operatii executandu-se manual

Clasa **Tren** dispune de un constructor prin intermediul caruia se vor atasa vagoanele constituente. Se cere:

- ✓ sa se implementeze clasa care modeleaza conceptul de vagon impreuna cu eventualele sale clase derivate. Se mentioneaza ca apelurile serviciilor pentru deschiderea, respectiv inchiderea usilor, blocarea geamurilor vor afisa pe ecran un mesaj corespunzator, spre exemplu, apelul serviciului pentru blocarea geamurilor ar putea tipari “Geamurile s-au blocat”.
- ✓ sa se implementeze o metoda pentru determinarea egalitatii dintre doua trenuri, presupunandu-se ca doua trenuri sunt egale daca pot transporta acelasi numar de colete, precum si o metoda pentru afisarea tipurilor de vagoane existente intr-un tren (ATENTIE: Tipul unui vagon trebuie determinat prin apeluri polimorfice).
- ✓ sa se scrie o metoda main pentru exemplificarea apelurilor celor 2 metode definite la punctul precedent.

Lab12_2: Clasa **ColectieGreutati**. Consideram o colectie de greutati in cadrul careia elementele sunt retinute sub forma unui tablou. Fiecare greutate are o anumita capacitate care poate fi obtinuta apeland metoda *public int capacitate()* pe care o are fiecare greutate. Greutatile pot fi de urmatoarele tipuri:

- **simple**, a caror capacitate este setata prin constructor.
- **duble**, adica formate din 2 greutati ale caror capacitati sunt setate prin constructor iar capacitatea acestui tip de greutate e egalacu suma capacitatilor celor doua greutati continute.
- **multiple**, care reprezinta o insiruire de greutati simple, duble, si/sau eventual alte greutati multiple. Cu alte cuvinte, o greutate multipla reprezinta o insiruire de greutati. Capacitatea unei greutati de acest tip este egala cu suma capacitatilor greutatilor componente. Componentele acestui tip de greutate se seteaza prin constructorul clasei, dar se poate alege si o alta modalitate de inserare a componentelor.

Sistemul mai cuprinde si clasa **ColectieGreutati** care contine un tablou de greutati (acestea reprezinta continutul efectiv al colectiei). Clasa **ColectieGreutati** va contine urmatoarele metode:

- *public void adauga(Greutate g):* are rolul de a adauga elemente in tabloul de greutati. Presupunem ca o colectie de greutati are o capacitate maxima de greutati care se seteaza prin intermediul constructorului.
- *public double medie():* returneaza greutatea medie a colectiei (capacitate/numar de greutati).

Se cere:

- ✓ implementarea claselor prezentate in diagrama.
- ✓ o metoda main in care se va crea un obiect **ColectieGreutati**, cateva greutati simple, duble si multiple care vor fi adaugate colectiei de greutati. Se va afisa greutatea medie a colectiei.

Bibliografie:

[1] http://www.pbinfo.ro *Descrierea site-ului:"www.pbinfo.ro îți propune să rezolvi probleme de informatică, cu evaluator automat. Știi pe loc dacă soluția ta este corectă sau dacă trebuie să mai lucrezi la ea. Problemele sunt grupate după programa de informatică pentru liceu. Dar nu trebuie să fii la liceu ca să rezolvi aceste probleme. Poți fi elev de gimnaziu, student, profesor sau pur și simplu pasionat de informatică. De fapt, trebuie doar să vrei!!"*

[2] https://www.runceanu.ro/adrian

[3] Adrian Runceanu *„Programarea și utilizarea calculatoarelor"*, Editura Academica Brâncuşi din Târgu-Jiu, 2003, ISBN 973-8436-44-3

[4] Adrian Runceanu, Mihaela Runceanu, *„Noțiuni de programare – limbajul C++"*, Editura Academica Brâncuși din Târgu-Jiu, 2012, ISBN 978-973-144-550-2

[5] Adrian Runceanu, Mihaela Runceanu, *„Algoritmi implementati in limbajul C++. Volumul I – Algoritmi elementari"*, Editura Academica Brâncuși din Târgu Jiu, 2021, ISBN 978-606-9614-06-8