

- [Pachete*](#)
 - [Exerciții](#)
- [Colecții și Tipuri generice - Exerciții](#)
- [Interfețe și extinderi - Exerciții](#)
- [Probleme - Tema](#)

Pachete*

Un **pachet** este o colecție de clase și interfețe care poate avea asociat un nume.

Pachetele se utilizează pentru a grupa clasele pe funcționalități, pentru a putea accesa clase din directoare diferite, pentru a evita conflicte de nume (dacă există în aplicație clase cu același nume), pentru a controla accesul la anumite clase și la membrii acestora.

În primele laboratoare unitățile de compilare au fost plasate toate în același director, fără a fi specificat explicit faptul că clasele aparțin unui pachet (fără a folosi instrucțiunea `package`). Atunci ia naștere un **pachet fără nume**, constituit din aceste unități.

Pentru aplicații mai complexe, în care plasarea claselor în același director nu este o soluție multumitoare, apare necesitatea grupării claselor după anumite criterii, în directoare diferite. Sunt necesare astfel pachete cu nume, pentru a putea accesa clase din alte directoare.

Vom prezenta pe scurt în cele ce urmează cum putem grupa clase în pachete cu nume.

Pachetele sunt organizate în **directoare** în sistemul de fișiere al mașinii gazdă, **numele pachetului reflectând această structură de directoare**.

Un pachet poate conține:


- subpachete ale pachetului;
- tipuri (clase sau interfețe) declarate în unitățile de compilare ale pachetului (având toate aceeași declarație `package`).

Pentru a crea în directorul de lucru un pachet, al cărui nume dorim să fie `p1`, trebuie creat un subdirector cu numele `p1` și apoi introduse în subdirector unitățile de compilare ale pachetului.

Fiecare unitate de compilare din pachetul `p1` trebuie să înceapă cu declarația:

`package p1;`

care dă informații compilatorului asupra numelui pachetului din care face parte unitatea de compilare.

 **Exemplul 1** Clasa "principală" `Main` crează un obiect `ob` de tipul unei clase `C` și invocă prin el o metodă `suma` a acestei clase.

Vom plasa cele două clase **în subdirectorul pachet al directorului curent**, prevăzând în ambele fișiere directiva:

```
package pachet;
```

Main.java

```
package pachet;
class Main {
    public static void main(String[] sss) {
        int a=1, b=2;
        C ob = new C();
        System.out.println( ob.suma(a,b) );
    }
}
```

C.java

```
package pachet;
class C {
    int suma(int x, int y) {
        return x+y;
    }
}
```

Compilarea se face:

- din directorul curent prin: `javac pachet\Main.java`
- sau din directorul pachet prin: `javac -classpath .. Main.java`

iar executarea se face:

- din directorul curent prin: `java pachet.Main`
- sau din directorul pachet prin: `java -classpath ../. pachet.Main`

Pentru a înțelege rolul opțiunii `-classpath` (lista de căi unde sunt căutate clase) și cum se poate folosi o clasă dintr-un alt pachet prezentăm câteva noțiuni generale și considerăm un exemplu mai complex, cu pachete și subpachete.

Dacă o unitate de compilare face referire la o clasă `c` din pachetul `p1`, atunci în această unitate trebuie plasată declararea `import`:

```
import p1.C;
```

Dacă dorim acces la toate clasele pachetului, putem folosi următoarea formă pentru declarare:

```
import p1.*; //! Prin import nu se importă și subpachete
```


Structura de pachete este ierarhică (formează un arbore); un pachet `p1` poate avea un subpachet `p2`. Pentru crearea subpachetului `p2` trebuie creat în directorul `p1` subdirectorul `p2`. O unitate de compilare din subpachetul `p2` trebuie **să înceapă** cu declararea:

```
package p1.p2;
```

Observăm că numele subpachetului reflectă structura de directoare, începând de la rădăcină. Această structură trebuie urmată și în declarațiile import

```
import p1.p2.*;
```

O clasă sau interfață dintr-un pachet poate fi accesată din afara pachetului doar dacă are modificatorul `public`. Aceeași regulă este valabilă pentru accesarea directă a membrilor unei clase a pachetului. Pentru constructori trebuie procedat la fel, afară de cazul în care se folosește constructorul implicit, care este considerat ca având modificatorul `public`.

 **Exemplu** Să considerăm o aplicație care reunește într-un meniu câteva dintre exemplele date în laboratoarele trecute, grupate pe teme de care se leagă exemplele. Aplicația are următoarea structură de directoare

```
example
  clase
    citire
  meniu
```

Directorul `example` conține pachetul `clase` (care va conține exemple legate de clase în Java în general), care are ca subpachet `citire` (care va conține clase care ilustrează citirea de la tastatură și din fișier în Java). De asemenea, directorul `example` conține directorul `meniu` (cu clasa principală, care perimte utilizatorului să selecteze ce exemple legate de limbajul Java dorește să ruleze).

Pentru simplitate, fiecare pachet va conține cel mult două clase.

Astfel, în `clase` se află unitatea de compilare `Dreptunghi.java`:

 **Dreptunghi.java**

```
package clase;
public class Dreptunghi{
    double lung,lat;
    public Dreptunghi(){
        lung=1;
        lat=1;
    }
    public Dreptunghi(double lung1,double lat1){
        lung=lung1;
        lat=lat1;
    }
    public double arie(){
        return lung*lat;
    }
    public boolean maiMare(Dreptunghi d){
        return arie()<d.arie();
    }
}
```

```

    public String toString(){
        return "lungime "+lung+",latime "+lat;
    }
}

```

În subdirectorul citire al directorului clase se află unitățile de compilare

Muchie.java și TestScanner.java

Muchie.java

```

package clase.citire;
class Muchie{
    int vi,vf;
    double cost;
    Muchie(int vi,int vf, double cost){
        this.vi=vi;
        this.vf=vf;
        this.cost=cost;
    }
    public String toString(){
        return "("+vi+", "+vf+")"+" "+cost;
    }
}

```

TestScanner.java

```

package clase.citire;
import java.util.*;
import java.io.*;
public class TestScanner{
    static final int NR_MAX=3;
    public void fisier(){
        int i,n;
        Muchie muchii[];
        try{
            Scanner scFisier=new Scanner(new File("muchie.txt"));
            n=scFisier.nextInt();
            muchii=new Muchie[n];
            for(i=0;i<n;i++)
                muchii[i]=new Muchie(scFisier.nextInt(), scFisier.nextInt(),
scFisier.nextDouble());
            scFisier.close();
            for(i=0;i<n;i++)
                System.out.println(muchii[i]);
        }
        catch(FileNotFoundException fnf){
            System.out.println("Fisier inexistent.");
        }
    }
}

```

```

    }

}

```

În directorul `menu` se află unitatea de compilare `Menu.java` și fișierul `muchie.txt`:

`Menu.java`

```

package menu; //poate lipsi

import clase.*; //nu importa si subpachetul citire
import clase.citire.TestScanner; //nu trebuie si Muchie
import java.util.*;

public class Menu{
    public static void main(String arg[]){
        int optiune=0;
        Scanner sc=new Scanner(System.in);
        do{
            System.out.println("Alegeti o optiune");
            System.out.println("1-Clase");
            System.out.println("2-Clasa Scanner");
            System.out.println("0-Iesire");
            optiune=sc.nextInt();
            switch(optiune){
                case 1:
                    Dreptunghi d=new Dreptunghi(3,4);
                    System.out.println(d+" arie "+d.aria());
                    //System.out.println(d.lung); //- NU
                    break;
                case 2:
                    //Muchie m; //-NU
                    TestScanner sts=new TestScanner();
                    sts.fisier();
            }

        }while(optiune!=0)        ;

    }

}

```

`muchie.txt`

```

5
2 3 10
1 3 4

```

```
1 2 17
1 4 20
3 4 5
```

Pentru a compila și rula această aplicație, trebuie să înțelegem unde sunt căutate clasele la compilarea și rularea unei aplicații.

La executarea unor comenzi ale mașinii virtuale Java, ca de exemplu `javac` și `java`, are loc o **căutare de clase**, adică de fișiere cu extensiile `.java` și `.class`. Pentru precizarea locului unde vor fi căutate aceste clase, Java prevede **variabila de mediu CLASSPATH**. Valoarea variabilei este o mulțime de căi separate între ele prin caracterul `'.'`.

Variabilei `CLASSPATH` i se poate atribui o valoare prin comanda:

```
SET CLASSPATH=path
```

sau prin opțiunea `classpath` din comenzile `java` și `javac`.

Ordinea în care apar căile determină ordinea de căutare a claselor.

Executarea comenzii `javac` are ca efect compilarea uneia sau a mai multor unități de compilare, ce conțin definiții de clase sau interfețe. Sunt compilate toate tipurile (clase sau interfețe) din unitățile respective, *dar și toate tipurile referite din ele și care sunt regăsite în calea pentru cod*.

O formă restrânsă a comenzii este:

```
javac opt unit
```

unde atât opțiunile din lista `opt`, cât și unitățile de compilare din lista `unit` sunt separate între ele prin blankuri.

Dintre opțiuni vom menționa numai următoarele două:

```
-d dir
```

```
-classpath path sau, prescurtat: -cp path
```

Prima opțiune permite plasarea fișierelor byte-code, rezultate în urma compilării, în directorul `dir`; în lipsa acestei opțiuni, drept `dir` este considerat directorul curent.

A doua opțiune permite precizarea unei noi căi de cod, în care se va încerca regăsirea (căutarea) claselor. Căutarea include atât fișierele cu extensia `.java`, cât și cele cu extensia `.class`.

La căutarea unui tip (clasă sau interfață) `tip` deosebim trei cazuri:

- 1) este găsit doar `tip.class`: este folosit acest fișier byte-code;
- 2) este găsit doar `tip.java`: este compilat acest tip și este folosit rezultatul compilării;
- 3) sunt găsite atât `tip.class` cât și `tip.java`: este folosit `tip.class` doar dacă acest fișier este mai recent decât `tip.java`.

Atribuirea unei valori variabilei `CLASSPATH` anulează atribuirea precedentă. Astfel, pentru a include directorul curent în cale trebuie inclus explicit caracterul `'.'` (caracterul `'.'` semnifică directorul curent, iar `'..'` directorul părinte al directorului curent).

Precizările de mai sus sunt valabile și pentru setările prin opțiunea `classpath` din comenzile `javac` și `java`, cu observația că în acest caz noua setare este valabilă numai pentru comanda respectivă.

În acest laborator vom utiliza opțiunea `classpath` a comenzilor (nu variabila de mediu `CLASSPATH`, cele două moduri de lucru fiind similare).

Compilarea aplicației din directorul meniu

Să presupunem că directorul `exemplu` este pe partiția `c` (`c:\exemple`).

Presupunem că **ne aflăm în directorul `c:\exemple\meniu`** (în care se găsește clasa principală, cu metoda `main`) (Pentru a ajunge în acest director din linia de comanda se execută `cd c:\exemple\meniu`)

La compilare, variabila `classpath` **trebuie să conțină calea către rădăcina structurii de pachete** care în acest caz este părintele directorului curent (adică directorul `c:\exemple`) și **directorul curent** (dacă sunt utilizate clase din acest director); este suficient să compilăm clasa principală (care conține metoda `main`), fiind compilate automat unitățile de compilare utilizate în clasa principală:

```
javac -classpath .;.. Meniu.java
```

Rularea aplicației din directorul meniu

La rulare variabila `classpath` trebuie să conțină calea către rădăcina structurii de pachete și directorul curent

```
java -classpath .;.. Meniu
```

❗ Observație Dacă includem și clasa `Meniu` în pachet (decomentăm instrucțiunea `package meniu` de la începutul fișierului `Meniu.java`), atunci interpretorului `i` se specifică numele complet al clasei (prefixat cu numele pachetului). Rularea se va face în acest caz cu comanda

```
java -classpath .;.. meniu.Meniu
```

📖 Exerciții:

1. Adăugați clase noi în pachetele din exemplul anterior (de exemplu clasa `Complex` în pachetul `clase`). Adăugați și un subpachet `siruri` pentru pachetul `clase` care să conțină o clasă care exemplifică lucrul cu clasa `String`. Adăugați opțiuni în meniu corespunzătoare claselor adăugate.
2. Compilați și rulați aplicația din directorul `c:\exemple` (în loc de `c:\exemple\meniu`).
3. Compilați și rulați aplicația din orice alt director.

Colecții și Tipuri generice

❗ Observație **NetBeans, Eclipse oferă facilitatea rescrierii automate a metodelor hashCode și equals** (pentru NetBeans: ALT+Insert în interiorul clasei sau din meniul Source -> Insert Code., selectați equals() and hashCode() și apoi câmpurile care intervin în criteriul de egalitate)

Exerciții:

1. Utilizând clasa `LinkedList` din pachetul `java.util`, creați o listă de șiruri de caractere, o listă de întregi și o listă de liste de întregi. Adăugați elemente noi și ștergeți elemente din aceste liste.
2. Ilustrați utilitatea unei alte colecții decât cele prezentate la curs.
3. Rescrieți în clasa de numere complexe metodele `toString`, `equals` și `hashCode`. **Ilustrați utilitatea metodelor rescrise** în lucrul cu colecții.
4. (opțional) În Java, pe lângă tipuri generice, există și metode generice. Scrieți o clasă care conține metode generice și ilustrați cum se utilizează acestea.

Interfețe și extinderi

Exerciții:

1. Scrieți o interfață și o clasă care implementează această interfață și interfața `Comparable`. Creați un tablou cu elemente de tipul clasei scrise și sortați acest tablou folosind `Arrays.sort`. Sortați acest tablou și după un alt criteriu decât cel implicit, utilizând interfața `Comparator`
2. (opțional) Dați un exemplu legate de extinderi de clase în care să fie evidențiate ideea de moștenire, rescrierea metodelor statice și nestatice, polimorfismul și legarea dinamică, utilizarea cuvintelor cheie `super`, `this` și semnificația modificatorilor de acces.

Probleme (TEMĂ)

1. Distanța între documente (1p)

https://en.wikipedia.org/wiki/Cosine_similarity

Fie două documente text, F_1 și F_2 , și $\{c_1, c_2, \dots, c_n\}$ mulțimea cuvintelor care apar în cel puțin unul din cele două documente. Pentru $1 \leq i \leq n$, fie v_{i1}, v_{i2} numărul de apariții al cuvântului i în primul, respectiv în al doilea document. Distanța cosinus dintre cele două documente, notată $dcos(F_1, F_2)$, dintre F_1 și F_2 se calculează după formula:

$$dcos(F_1, F_2) = \frac{\sum_{i=1}^n v_{i1} v_{i2}}{\sqrt{\sum_{i=1}^n v_{i1}^2} \sqrt{\sum_{i=1}^n v_{i2}^2}}$$

Fiind date două fișiere text, F_1 și F_2 , să se calculeze distanța cosinus dintre cele două fișiere. Fiecare dintre cele două fișiere va fi parcurs o singură dată. Cuvintele dintr-un fișier sunt separate prin spații, virgulă, punct. **Exemplu:** dacă F_1 conține textul: *primul laborator primul exercitiu* și

F_2 conține textul: *primul exercitiu usor*

$\{c_1 = \text{'primul'}, c_2 = \text{'laborator'}, c_3 = \text{'exercitiu'}, c_4 = \text{'usor'}\}$

$\{v_{11} = 2, v_{21} = 1, v_{31} = 1, v_{41} = 0\}$

$\{v_{12} = 1, v_{22} = 0, v_{32} = 1, v_{42} = 1\}$

$$dcos(F_1, F_2) = \frac{2 * 1 + 1 * 0 + 1 * 1 + 0 * 1}{\sqrt{6}\sqrt{3}} = 0,7071$$

2. **Graf (1p).** Modificați problema din tema trecută trecut privind parcurgerea unui graf în care utilizați clasa de coadă proprie astfel încât în loc de clasa proprie să utilizați colecția `ArrayDeque` din Java (1p)
3. **Coadă – generică+iterabilă (4p)**
 - a) Modificați clasa de coadă din tema trecută astfel încât să devină generică și iterabilă. (3p)
 - b) Folosind această clasă creați o coadă de șiruri de caractere, o coadă de întregi și o coadă de cozi de numere întregi. Adăugați elemente noi și ștergeți elemente din aceste cozi. (0,5p)
 - c) Modificați clasă de graf din problema 2 astfel încât să folosiți clasa de coadă proprie în loc de `ArrayDeque`; parcurgerea în lățime trebuie să aibă complexitatea $O(|V|+|E|)$ (! Nu ar trebui să schimbați în clasă decât numele clasei `ArrayDeque` cu clasa proprie) (0,5p)
4. **Arbore binar generic (3p).** Scrieți o clasă generică pentru arbori binari de căutare cu valori (chei) distincte având următoarele:
 - o metodă de adăugare a unei valori în arbore; metoda va returna `false` dacă valoarea există deja în arbore și `true` în caz contrar (1p)
 - rescrie metoda `toString`, care va returna un șir cu valorile din arbore ordonate crescător și separate prin spațiu (parcurgerea în inordine a arborelui) (0,5p)
 Pentru a testa clasa:
 - Dat un fișier ce conține numere întregi separate prin spațiu, creați un arbore de căutare având ca valori numere din fișier și afișați arborele creat. (0,5p)
 - Scrieți o clasă care implementează interfața `Comparable` și o altă clasă care derivă din aceasta (o extinde pe aceasta). Creați (prin adăugări succesive) arbori binari de căutare având ca informație obiecte de tipul superclasei (clasei inițiale), respectiv subclasei (clasei derivate) și afișați arborii creați. (1p)
5. **Completare arbore - suplimentar (1p).** Modificați clasa de la problema 4 astfel încât să aibă:

- rescrie metoda `equals`, care va testa dacă arborele curent este egal (ca și arbore cu rădăcină) cu arborele primit ca parametru
- rescrie metoda `hashCode`

Creați doi arbori de căutare diferiți având elemente numere întregi, dar care conțin aceleași valori și testați egalitatea acestora folosind metoda `equals`

Creați doi arbori de căutare egali având ca elemente șiruri de caractere și testați egalitatea acestora folosind metoda `equals`.