```
import org.json.JSONArray;
import org.json.JSONTokener;


import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class Main {
   static final String CALE_PACIENTI = "date\\pacienti.txt";
   static final String CALE_SECTII = "date\\sectii.json";
   static final int SERVER_PORT = 8276;
   static List<Pacient> pacienti;
   static List<Sectie> sectii = new ArrayList<>();

   public static void main(String[] args) throws Exception {
       try(var fisier = new FileReader(CALE_SECTII)){
           var array = new JSONArray(new JSONTokener(fisier));
           for(int i=0;i<array.length();i++){
               var jsonSectie = array.getJSONObject(i);

               var cod = jsonSectie.getInt("cod_sectie");
               var den =  jsonSectie.getString("denumire");
               var nr = jsonSectie.getInt("numar_locuri");
               sectii.add(new Sectie(cod,den,nr));
           }
       }
       sectii.stream().filter(p -> p.getNrLocuri()>10).forEach(p
->System.out.println(p));

       try(var fisier = new BufferedReader(new
FileReader(CALE_PACIENTI))){
           pacienti = fisier.lines()
                   .map(linie -> new Pacient(
                           Long.parseLong(linie.split(",")[0]),
                           linie.split(",")[1],
                           Integer.parseInt(linie.split(",")[2]),
                           Integer.parseInt(linie.split(",")[3])
                   ))
                   .collect(Collectors.toList());
       }
       for (Pacient p2 :pacienti) {
           sectii.stream()
                   .filter(p -> p.getCod() == p2.getCodSectie())
```

```
.findFirst().orElse(null).getListaPacienti().add(p2);
        }
        System.out.println();
        sectii.stream().sorted((p1,p2) ->
Float.compare(p2.medieVarsta(), p1.medieVarsta()))
                .forEach(p -> System.out.println(p));

        try(var fisier = new PrintWriter(new
FileWriter("date\\jurnal.txt"))){
            sectii.stream().forEach(p -> fisier.printf("%d %s %d%n",
p.getCod(), p.getDenumire(), p.getListaPacienti().size()));
        }

        new Thread(Main::serverTCP).start();

        try(var socket = new Socket("localhost", SERVER_PORT);
        var out = new ObjectOutputStream(socket.getOutputStream());
        var in = new ObjectInputStream(socket.getInputStream());
        ){
            int codS = 2;
            out.writeObject(codS);
            int locuriLibere = (Integer)in.readObject();
            System.out.println(locuriLibere);
        }
    }

    public static void serverTCP() {
        try(
                var serverSocket = new ServerSocket(SERVER_PORT);
                var socket = serverSocket.accept();
                var in = new
ObjectInputStream(socket.getInputStream());
                var out = new
ObjectOutputStream(socket.getOutputStream());
                ){
            int codSectie = (Integer) in.readObject();
            Sectie sectie = sectii.stream().filter(p -> p.getCod()
== codSectie).findFirst().orElse(null);
            out.writeObject(sectie.getNrLocuri() -
sectie.getListaPacienti().size());
        }
        catch(Exception ex){
            ex.printStackTrace();
        }
    }

}
```

//pacient

```
final class Pacient {
   private final long codPacient;
   private final String nume;
   private final int varsta;
   private final int codSectie;

   public Pacient(long codPacient, String nume, int varsta, int
codSectie) {
        this.codPacient = codPacient;
        this.nume = nume;
        this.varsta = varsta;
        this.codSectie = codSectie;
   }

   public long getCodPacient() {
        return codPacient;
   }

   public String getNume() {
        return nume;
   }

   public int getVarsta() {
        return varsta;
   }

   public int getCodSectie() {
        return codSectie;
   }
}
```

//sectie

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

final class Sectie {
   final int cod;
   final String denumire;
   final int nrLocuri;
   final List<Pacient> listaPacienti;

   public Sectie(int cod, String denumire, int nrLocuri) {
        this.cod = cod;
        this.denumire = denumire;
        this.nrLocuri = nrLocuri;
```

```
        listaPacienti = new ArrayList<>();
    }

    public int getCod() {
        return cod;
    }

    public String getDenumire() {
        return denumire;
    }

    public int getNrLocuri() {
        return nrLocuri;
    }

    public List<Pacient> getListaPacienti() {
        return listaPacienti;
    }

    @Override
    public String toString() {
        return "Sectie{" +
                "cod=" + cod +
                ", denumire='" + denumire + '\'' +
                ", nrLocuri=" + nrLocuri +
                ", Varsta Medie="+medieVarsta()+
                '}';
    }

    public float medieVarsta()
    {
        float m = 0;
        for (Pacient p: listaPacienti
             ) {
            m+=p.getVarsta();
        }
        return  m/listaPacienti.size();
    }
}
```

//SUBIECT EXAMEN de la Vințe

```
import org.json.JSONArray;
import org.json.JSONTokener;

import java.io.*;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
```

```java
import java.util.List;
import java.util.Scanner;
import java.util.stream.Collectors;


final class Tranzactie {
    final int cantitate;

    public Tranzactie(int cantitate) {
        this.cantitate = cantitate;
    }

    public int getCantitate() {
        return cantitate;
    }

    @Override
    public String toString() {
        return "Tranzactie{" +
                "cantitate=" + cantitate +
                '}';
    }
}


final class Produs {

    private final int cod;
    private final String denumire;
    private final double pret;
    private final List<Tranzactie> tranzactii;

    public Produs(int cod, String denumire, double pret) {
        this.cod = cod;
        this.denumire = denumire;
        this.pret = pret;
        tranzactii = new ArrayList<>();
    }

    public int getCod() {
        return cod;
    }

    public String getDenumire() {
        return denumire;
    }

    public double getPret() {
```

```
        return pret;
    }

    public List<Tranzactie> getTranzactii() {
        return tranzactii;
    }

    public double getValoareStoc() {
        int cantitateCurenta = 0;
        for (var tranzactie : tranzactii) {
            cantitateCurenta += tranzactie.getCantitate();
        }
        return cantitateCurenta * getPret();
    }

    @Override
    public String toString() {
        return "Produs{" +
                "cod=" + cod +
                ", denumire='" + denumire + '\'' +
                ", pret=" + pret +
                ", tranzactii=" + tranzactii +
                ", valStoc=" + getValoareStoc() +
                '}';
    }
}

class ExempluSubiect {

    static final String CALE_PRODUSE = "date\\produse.txt";
    static final String CALE_TRANZACTII = "date\\tranzactii.json";

    static final int SERVER_PORT =8276;

    static List<Produs> produse;

    public static void main(String[] args) throws Exception {

//        int numarProduse = 0;
//        try (var fisier = new Scanner(new File(CALE_PRODUSE))) {
//            while (fisier.hasNextLine()) {
//                var linie = fisier.nextLine();
//                var produs = new Produs(
//                        Integer.parseInt(linie.split(",")[0]),
//                        linie.split(",")[1],
//                        Double.parseDouble(linie.split(",")[2])
//                );
//                numarProduse++;
```

```
//             }
//         }
//         System.out.printf("CERINTA 1: Numar produse %d%n",
numarProduse);
//         System.out.println();

        try (var fisier = new BufferedReader(new
FileReader(CALE_PRODUSE))) {
            produse = fisier.lines()
                    .map(linie -> new Produs(
                            Integer.parseInt(linie.split(",")[0]),
                            linie.split(",")[1],
                            Double.parseDouble(linie.split(",")[2])
                    ))
                    .collect(Collectors.toList());
        }

        System.out.printf("CERINTA 1: Numar produse %d%n",
produse.size());
        System.out.println();

        System.out.println("CERINTA 2:");
        produse.stream()
                .sorted((p1, p2) ->
p1.getDenumire().compareTo(p2.getDenumire()))
                .forEach(p -> System.out.println(p.getDenumire()));
        System.out.println();

        try (var fisier = new FileReader(CALE_TRANZACTII)) {
            var array = new JSONArray(new JSONTokener(fisier));
            for (int i = 0; i < array.length(); i++) {
                var jsonTranzactie = array.getJSONObject(i);

                var cod = jsonTranzactie.getInt("codProdus");
                var cantitate = jsonTranzactie.getInt("cantitate");
                var tip = jsonTranzactie.getString("tip");

                if (tip.equals("iesire")) {
                    cantitate = -cantitate;
                }

                produse.stream()
                        .filter(p -> p.getCod() == cod)
                        .findFirst().orElse(null)
                        .getTranzactii().add(new
Tranzactie(cantitate));
            }
        }
```

```
        System.out.println("CERINTA 3: -> in fisier");
        try (var fisier = new PrintWriter(new FileWriter(
                "date\\lista.txt"))) {
            produse.stream()
                    .sorted((p1, p2) ->
Integer.compare(p2.getTranzactii().size(),
p1.getTranzactii().size()))
                    .forEach(p -> fisier.printf("%s, %d%n",
                            p.getDenumire(),
                            p.getTranzactii().size()));
        }

        double valoareTotala = 0;
        for (var produs : produse) {
            valoareTotala += produs.getValoareStoc();
        }
        System.out.printf(
                "CERINTA 4: Valoarea totala a stocului curent este
%.2f lei%n",
                valoareTotala);

        // -> pornire server pe thread secundar
        new Thread(ExempluSubiect::serverTcp).start();

        // -> trimitere cerere către server
        try (var socket = new Socket("localhost", SERVER_PORT);
             var out = new
ObjectOutputStream(socket.getOutputStream());
             var in = new
ObjectInputStream(socket.getInputStream());
        ) {
            int codProdus = 2;

            out.writeObject(2);

            // -> citeste de pe socket raspunsul de la server
            double valoareStoc = (Double) in.readObject();

            System.out.printf(
                    "CERINTA 5: Valoarea stocului pentru produsul
#%d este %.2f lei%n",
                    codProdus,
                    valoareStoc);
        }
    }

    public static void serverTcp() {
```

```
        try (
                var serverSocket = new ServerSocket(SERVER_PORT);
                var socket = serverSocket.accept();
                var in = new
ObjectInputStream(socket.getInputStream());
                var out = new
ObjectOutputStream(socket.getOutputStream());
        ) {
            int codProdus = (Integer) in.readObject();
            Produs produs = produse.stream()
                    .filter(p -> p.getCod() == codProdus)
                    .findFirst().orElse(null);
            out.writeObject(produs.getValoareStoc());
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}
```

//fisierul cerinte.txt

```
Fie datele de intrare (in directorul date\subiect1):

a) Lista de produse (cod produs - intreg, denumire - string, pret
- double) - fișier text de forma:
1,Stafide 200g,6.03
2,Seminte de pin 300g, 21.18
...

Denumirea produsului nu poate conține caracterul virgulă.

b) Lista de tranzacții (cod produs, cantitate - intreg, tip -
intrare / iesire) - fisier JSON de forma:
[
 {
   "codProdus": 3,
   "cantitate": 7,
   "tip": "intrare"
 },
 {
   "codProdus": 1,
   "cantitate": 7,
   "tip": "intrare"
 },
 {
   "codProdus": 1,
   "cantitate": 2,
   "tip": "iesire"
 }
]
```

Cerințe:

1) Să se afișeze la consolă numărul de produse

2) Să se afișeze la consolă lista de produse ordonate alfabetic.

3) Să se scrie în fișierul text date\subiect1\lista.txt un raport de forma:
Denumire Produs, Numar tranzactii

Produsele trebuie să fie ordonate în ordinea descrescătoare a numărului de tranzacții.

4) Să se afișeze la consolă valoarea totală a stocurilor.

5) Să se implementeze funcționalitățile de server și client TCP/IP și să se
execute următorul scenariu: componenta client trimite serverului un cod de produs
iar componenta server va întoarce clientului valoarea stocului corespunzător.

- server pe thread secundar
- server-ul se oprește după ce a servit o conexiune

```
////////////
        String[] numeSub30 = persoane.stream()          // construire stream
                .filter(item -> item.getVarsta() < 30)  // operație intermediară (expresie lambda)
                .map(Persoana::getNume)                 // operație intermediară (referință metodă)
                .toArray(String[]::new);                // operație terminală

        System.out.println("Persoane cu varsta sub 30 de ani:");
        Arrays.stream(numeSub30)
                .forEach(item -> System.out.println(item));

        Consumer<Persoana> afisare = persoana -> System.out.println(persoana);

        System.out.println("Afisare persoane cu in ordine alfabetica (lexicografica dupa atributul String nume):");
        persoane.stream()
                .sorted((a, b) -> a.getNume().compareTo(b.getNume()))
```

```
                .forEach(afisare);

        System.out.println("Lista de persoane majore:");
        var numePersoaneMajore = persoane.stream()
                .filter(p -> p.getVarsta() >= 18)
                .map(Persoana::getNume)
                .collect(Collectors.toList());
        numePersoaneMajore.stream().forEach(System.out::println);

        int suma = persoane.stream()
                .limit(4)
                .map(Persoana::getVarsta)
                .reduce(0, (s, varsta) -> s + varsta);
        System.out.printf("Suma vârstelor pentru primii 3 este %d
ani.%n", suma);

        class Medie {
            final int numar;
            final int suma;

            public Medie() {
                numar = 0;
                suma = 0;
            }

            public Medie(int numar, int suma) {
                this.numar = numar;
                this.suma = suma;
            }

            public double getMedie() {
                return numar > 0 ? (double)suma / numar : 0;
            }

            public Medie accepta(Persoana persoana) {
                return new Medie(numar + 1, suma +
persoana.getVarsta());
            }

            public Medie combina(Medie medie){
                return new Medie(numar + medie.numar, suma +
medie.suma);
            }
        };

        double medie = persoane.stream()
                .parallel()
```

```
                .reduce(new Medie(), Medie::accepta,
Medie::combina)
                .getMedie();
        System.out.printf("Vârsta medie este %.2f ani.%n", medie);
//thread pool
class TaskCautareInFisier implements Runnable {
    String caleFisier;
    String textDeCautat;

    public TaskCautareInFisier(String caleFisier, String
textDeCautat) {
        this.caleFisier = caleFisier;
        this.textDeCautat = textDeCautat;
    }

    @Override
    public void run() {
        if (!caleFisier.endsWith(".java")) {
            return;
        }
        try (var fileIn = new BufferedReader(
                new FileReader(caleFisier)
        )) {
            int lineNumber = 1;
            String line;
            Thread.sleep(3);

            while ((line = fileIn.readLine()) != null) {
                if (line.contains(textDeCautat)) {
                    System.out.printf("%s:%d %s%n", caleFisier,
lineNumber, line);
                }
                lineNumber++;
            }
        }
        catch (Exception exception) {
            exception.printStackTrace();
        }

    }
}

public class Program07_ThreadPool {

    static final int NUMAR_FIRE = 3;
    static ExecutorService threadPool =
Executors.newFixedThreadPool(NUMAR_FIRE);
```

```
    public static void showFolders(File cale, String textDecautat)
throws IOException {
        if (cale.isDirectory()) {
            for (var child : cale.listFiles()) {
                showFolders(child, textDecautat);
            }
        } else {
            threadPool.submit(
                new TaskCautareInFisier(cale.getCanonicalPath(),
textDecautat)
            );
        }
    }

    public static void main(String[] args) throws Exception {
        var start = System.nanoTime();
        showFolders(new File("..\\"), "particula");
        threadPool.shutdown();
        threadPool.awaitTermination(30, TimeUnit.SECONDS);
        System.out.printf("Gata - %d ms", (System.nanoTime() -
start) / 1000000);
    }
}

public class Program01_ReflectionSimple {
    public static void main(String[] args) throws Exception {

        // 1. Obținere instanțe Class<T>
        Persoana ion = new Persoana(1, "Ion");
        Class<?> clasaPersoana = ion.getClass();    // dintr-un
obiect existent
        Class<?> clasaString = Class.forName("java.lang.String");
        Class<?> clasaInt = int.class;  // pe baza tipului

        // 2. Obținere informații despre clasă
        System.out.println("Modificatori pentru " +
clasaString.getName());
        int modificatori = clasaString.getModifiers();
        System.out.println(Modifier.toString(modificatori));

        // 3. Manipulare câmpuri
        for (Field camp : clasaPersoana.getDeclaredFields()) {
            // obligatoriu pentru campurile private
            camp.setAccessible(true);

            System.out.printf("%s %s %s = %s%n",
                    Modifier.toString(camp.getModifiers()),
                    camp.getType().getName(),
```

```
                    camp.getName(),
                    camp.get(ion));
        }

        var campCod = clasaPersoana.getDeclaredField("cod");
        campCod.setAccessible(true);
        campCod.set(ion, 3);
        System.out.println(ion);

        // 4. Utilizare metode

        // Obținere informații
        for (Method metoda : clasaPersoana.getDeclaredMethods()) {
            String modif = Modifier.toString(metoda.getModifiers());
            Class<?> tipReturnat = metoda.getReturnType();
            String denumire = metoda.getName();
            System.out.printf("%s %s %s, parametri: ",
                    modif, tipReturnat.getName(), denumire);
            for (Parameter param : metoda.getParameters()) {
                System.out.printf("%s ", param.getType().getName());
            }
            System.out.println();
        }

        // Invocare
        Method metodaSetCod = clasaPersoana
                .getDeclaredMethod("setCod", int.class);
        // pentru metode private: metodaSetCod.setAccessible(true);
        metodaSetCod.invoke(ion, 4);
        System.out.println(ion);

        // 5. Utilizare constructori

        // afișare
        for (Constructor<?> constructor :
clasaPersoana.getConstructors()) {
            System.out.println("Constructor cu parametrii:");
            for (Parameter parametru : constructor.getParameters())
{
                System.out.println(parametru.getType().getName() +
" " + parametru.getName());
            }
        }

        // construire obiect
        var constructor = clasaPersoana.getConstructor(int.class,
String.class);
```

```
        Persoana maria = (Persoana) constructor.newInstance(2,
"Maria");
        System.out.println(maria);
    }
}

class Persoana {
    public transient String nume; // acest atribut nu va fi
serializat
    private int cod;

    public Persoana(int cod, String nume) {
        this.cod = cod;
        this.nume = nume;
    }

    public int getCod() {
        return cod;
    }

    public void setCod(int cod) {
        this.cod = cod;
    }

    public String getNume() {
        return nume;
    }

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

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Persoana{");
        sb.append("cod=").append(cod);
        sb.append(", nume='").append(nume).append('\'');
        sb.append('}');
        return sb.toString();
    }
}

public class Program01_ServerTCP {

    public static void main(String[] args) throws IOException {
        final int PORT_NUMBER = 8293;
        try (ServerSocket serverSocket = new
ServerSocket(PORT_NUMBER)) {
```

```
            while (true) {
                System.out.println("Așteptăm un client...");
                try (Socket socket = serverSocket.accept();
                     BufferedReader in = new BufferedReader(
                             new
InputStreamReader(socket.getInputStream()));
                     PrintWriter out = new
PrintWriter(socket.getOutputStream(), true);
                ) {
                    System.out.println("Am stabilit conexiunea.
Așteptăm mesajul...");

                    String mesajPrimit = in.readLine();
                    System.out.println("Mesaj primit - " +
mesajPrimit);
                    out.printf("Mesajul de raspuns server - am
primit: %s%n", mesajPrimit);

                    System.out.println("Am terminat de procesat
cererea - închidem conexiunea.");
                }
            }
        }
    }
public class Program02_ClientTCP {
    public static void main(String[] args) throws IOException {
        final int PORT_NUMBER = 8293;
        try (Socket socket = new Socket("127.0.0.1", PORT_NUMBER);
             BufferedReader in = new BufferedReader(
                     new
InputStreamReader(socket.getInputStream()));
             PrintWriter out = new
PrintWriter(socket.getOutputStream(), true);) {

            System.out.println("Am deschis conexiunea cu
server-ul.");

            String mesajCerere = "Test";
            out.println(mesajCerere);
            System.out.println("Am trimis mesajul: " + mesajCerere +
". Așteptăm răspunsul ...");

            System.out.println(in.readLine());

            System.out.println("Am primit răspunsul - închidem
conexiunea.");
        }
    }
```

```
}

public class Program03_ServerTCPMultithreaded {

   public static void main(String[] args) throws IOException {
       final int PORT_NUMBER = 8293;
       try (ServerSocket serverSocket = new
ServerSocket(PORT_NUMBER)) {
           while (true) {
               System.out.println("Așteptăm un client...");
               Socket socket = serverSocket.accept();
               new Thread(() -> procesareCerere(socket)).start();
               // Runnable este o interfata functionala, a se urmari
link-ul de mai jos:
               //
https://docs.oracle.com/javase/8/docs/api/java/lang/Runnable.html#:~
:text=Interface%20Runnable&text=This%20is%20a%20functional%20interfa
ce,lambda%20expression%20or%20method%20reference.&text=The%20Runnabl
e%20interface%20should%20be,be%20executed%20by%20a%20thread.&text=Fo
r%20example%2C%20Runnable%20is%20implemented%20by%20class%20Thread%2
0.
           }
       }
   }

   private static void procesareCerere(Socket socket) {
       try {
           try (BufferedReader in = new BufferedReader(
                   new
InputStreamReader(socket.getInputStream()));
                PrintWriter out = new
PrintWriter(socket.getOutputStream(), true);
           ) {
               System.out.println("Am stabilit conexiunea. Așteptăm
mesajul...");

               String mesajPrimit = in.readLine();
               System.out.println("Mesaj primit - " + mesajPrimit);
               out.printf("Mesajul de raspuns server - am primit:
%s%n", mesajPrimit);

               System.out.println("Am terminat de procesat cererea
- închidem conexiunea.");
           }
       } catch (IOException exception) {
           exception.printStackTrace();
       } finally {
           try {
```

```
                socket.close();
            } catch (IOException exception) {
            }
        }
    }
}
//operatii jdbc
class Persoana {
    private int cod;
    private String nume;

    public Persoana() {
        this(0, "-");
    }

    public Persoana(int cod, String nume) {
        this.cod = cod;
        this.nume = nume;
    }

    public int getCod() {
        return cod;
    }

    public void setCod(int cod) {
        this.cod = cod;
    }

    public String getNume() {
        return nume;
    }

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

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Student{");
        sb.append("cod=").append(cod);
        sb.append(", nume='").append(nume).append('\'');
        sb.append('}');
        return sb.toString();
    }
}

public class Program01_OperatiiJDBC {
```

```
    public static void main(String[] args) throws Exception {
        String url = "jdbc:sqlite:date\\persoane.db";

        // 1. Cerere simplă (ștergerea înregistrărilor existente)
        try (Connection connection =
DriverManager.getConnection(url);
             Statement statement = connection.createStatement()) {

            int numarInregistrari = statement.executeUpdate("DELETE
FROM Persoane");
            System.out.printf("Au fost șterse %d înregistrări.%n",
numarInregistrari);
        }

        // 2. Cerere cu parametru (inserare înregistrări din listă)
        var persoane = List.of(
                new Persoana(1, "Popescu Maria"),
                new Persoana(2, "Marinescu Ion"));

        try (Connection connection =
DriverManager.getConnection(url);
             PreparedStatement statement = connection
                     .prepareStatement("INSERT INTO Persoane(Cod,
Nume) VALUES (?, ?)");) {

            for (var persoana : persoane) {
                statement.setInt(1, persoana.getCod());
                statement.setString(2, persoana.getNume());
                statement.executeUpdate();
            }
        }

        // 3. Citire înregistrări din tabelă (utilizare ResultSet)

        List<Persoana> persoaneBD = new ArrayList<>();

        try (Connection connection =
DriverManager.getConnection(url);
             Statement statement = connection.createStatement();) {

            try (ResultSet date = statement.executeQuery("SELECT
Cod, Nume FROM Persoane")) {

                while (date.next()) {
                    persoaneBD.add(new Persoana(date.getInt(1),
date.getString(2)));
                    // sau: date.getInt("Cod"),
date.getString("Nume")
```

```
                }
            }
        }

        persoaneBD.stream().forEach(student ->
System.out.println(student));

        // 4. Utilizare tranzacții explicite
        persoaneBD.add(new Persoana(3, "Ionescu Țițel"));
        try (Connection connection =
DriverManager.getConnection(url)) {
            connection.setAutoCommit(false);

            try {
                // Operația 1: Ștergem datele existente
                try (var deleteStatement =
connection.createStatement()) {
                    deleteStatement.executeUpdate("DELETE FROM
Persoane");
                }

                // Operația 2: Adăugâm înregistrările din listă
                try (var insertStatement =
                             connection.prepareStatement("INSERT
INTO Persoane(Cod, Nume) VALUES (?, ?)")) {
                    for (var persoana : persoaneBD) {
                        insertStatement.setInt(1,
persoana.getCod());
                        insertStatement.setString(2,
persoana.getNume());
                        insertStatement.executeUpdate();
                    }
                }

                connection.commit();
            } catch (Exception e) {
                connection.rollback();
                throw e;
            }
        }

        // verificare date după tranzacție
        try (Connection connection =
DriverManager.getConnection(url);
             Statement statement = connection.createStatement();) {

            try (ResultSet date = statement.executeQuery("SELECT
Cod, Nume FROM Persoane")) {
```

```
                while (date.next()) {
                    System.out.printf("%d - %s%n",
                            date.getInt("Cod"),
                            date.getString("Nume"));
                }
public class Program02_UtilizareMetadate {

   public static <T> List<T> citireLista(String url, String
numeTabela, Class<T> clasa)
           throws SQLException, NoSuchMethodException,
IllegalAccessException, InvocationTargetException,
InstantiationException {

       var rezultat = new ArrayList<T>();

       var sql = String.format("SELECT * FROM %s", numeTabela);

       try (Connection connection =
DriverManager.getConnection(url);
            Statement statement = connection.createStatement();) {

           try (ResultSet resultSet = statement.executeQuery(sql))
{
               var metadate = resultSet.getMetaData();
               while (resultSet.next()) {
                   var obiect =
clasa.getConstructor().newInstance();
                   for (int colIndex = 1; colIndex <=
metadate.getColumnCount(); colIndex++) {
                       // pentru fiecare celulă din rândul curent
căutam  metoda
                       // set corespunzătoare și o apelăm (dacă
există)
                       for (var metoda : clasa.getMethods()) {
                           if (metoda.getName().equals("set" +
metadate.getColumnName(colIndex))) {
                               metoda.invoke(obiect,
resultSet.getObject(colIndex));
                               break;
                           }
                       }
                   }
                   rezultat.add(obiect);
               }
           }
       }
       return rezultat;
```

```
    }

    public static void main(String[] args) throws Exception {
        String url = "jdbc:sqlite:date\\persoane.db";

        // 1. Afișare tabele existente
        try (Connection connection =
DriverManager.getConnection(url);) {
            DatabaseMetaData metadate = connection.getMetaData();
            try (var tables = metadate.getTables(null, null, null,
null);) {
                while (tables.next()) {
                    var denumireTabela =
tables.getString("TABLE_NAME");
                    System.out.println(denumireTabela);

                    // Afișare coloane
                    try(var coloane =
metadate.getColumns(null,null,denumireTabela,null))
                    {
                        while (coloane.next()) {
                            System.out.printf("%-12s %s%n",

coloane.getString("COLUMN_NAME"),
                                    coloane.getString("TYPE_NAME"));
                        }
                    }
                    System.out.println();
                }
            }
        }

        // 2. Metodă generică pentru citire date
        citireLista(url, "Persoane", Persoana.class).stream()
                .forEach(elem -> System.out.println(elem));

        citireLista(url, "Produse", Produs.class).stream()
                .forEach(elem -> System.out.println(elem));
    }
}


class Produs {
    private int cod;
    private String denumire;
    private double pret;
    private int cantitate;
```

```
    public Produs() {
        denumire = "";
    }

    public int getCod() {
        return cod;
    }

    public void setCod(int cod) {
        this.cod = cod;
    }

    public String getDenumire() {
        return denumire;
    }

    public void setDenumire(String denumire) {
        this.denumire = denumire;
    }

    public double getPret() {
        return pret;
    }

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

    public int getCantitate() {
        return cantitate;
    }

    public void setCantitate(int cantitate) {
        this.cantitate = cantitate;
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Produs{");
        sb.append("cod=").append(cod);
        sb.append(", denumire='").append(denumire).append('\'');
        sb.append(", pret=").append(pret);
        sb.append(", cantitate=").append(cantitate);
        sb.append('}');
        return sb.toString();
    }
class Tranzactie {
    private int codProdus;
```

```
    private String denumireProdus;
    private double pret;
    private int cantitate;

    public Tranzactie(int codProdus, String denumireProdus, double
pret, int cantitate) {
        this.codProdus = codProdus;
        this.denumireProdus = denumireProdus;
        this.pret = pret;
        this.cantitate = cantitate;
    }

    public int getCodProdus() {
        return codProdus;
    }

    public void setCodProdus(int codProdus) {
        this.codProdus = codProdus;
    }

    public String getDenumireProdus() {
        return denumireProdus;
    }

    public void setDenumireProdus(String denumireProdus) {
        this.denumireProdus = denumireProdus;
    }

    public double getPret() {
        return pret;
    }

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

    public int getCantitate() {
        return cantitate;
    }

    public void setCantitate(int cantitate) {
        this.cantitate = cantitate;
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Tranzactie{");
        sb.append("codProdus=").append(codProdus);
```

```
        sb.append(",
denumireProdus='").append(denumireProdus).append('\'');
        sb.append(", pret=").append(pret);
        sb.append(", cantitate=").append(cantitate);
        sb.append('}');
        return sb.toString();
    }
}

public class Program03_UtilizareTranzactii_JDBC {

    static final String url = "jdbc:sqlite:date\\tranzactii.db";

    static void adaugaTranzactii(List<Tranzactie> tranzactii) throws
SQLException {
        try (
                var connection = DriverManager.getConnection(url);
                var cmdInsert = connection.prepareStatement(
                        "INSERT INTO Tranzactii VALUES (?, ?, ?,
?)");
        ) {
            for (var tranzactie : tranzactii) {
                cmdInsert.setInt(1, tranzactie.getCodProdus());
                cmdInsert.setString(2,
tranzactie.getDenumireProdus());
                cmdInsert.setDouble(3, tranzactie.getPret());
                cmdInsert.setInt(4, tranzactie.getCantitate());

                cmdInsert.execute();
            }
        }
    }

    static void genereazaBazaDeDate() throws SQLException {
        try (
                var connection = DriverManager.getConnection(url);
                var command = connection.createStatement();
        ) {
            command.executeUpdate(
                    "CREATE TABLE IF NOT EXISTS Tranzactii (" +
                            "CodProdus INTEGER, " +
                            "DenumireProdus STRING(100), " +
                            "Pret REAL, " +
                            "Cantitate INTEGER)");


            command.executeUpdate("DELETE FROM Tranzactii");
        }
```

```
        final String[] produse = {
                "Stafide 200g",
                "Seminte de pin 300g",
                "Bulion Topoloveana 190g",
                "Paine neagra Frontera",
                "Ceai verde Lipton"
        };

        var random = new Random();

        List<Tranzactie> tranzactii = new ArrayList<>();
        for (int index = 0; index < 30; index++) {
            int cod = random.nextInt(produse.length);
            String denumire = produse[cod];
            double pret = 1 + random.nextDouble() * 10;
            int cantitate = random.nextInt(30) - 10;
            tranzactii.add(new Tranzactie(cod, denumire, pret,
cantitate));
        }

        adaugaTranzactii(tranzactii);
    }

    static void afisareStocuri() throws SQLException {
        List<Tranzactie> tranzactii = new ArrayList<>();
        try (
                var connection = DriverManager.getConnection(url);
                var select = connection.createStatement();
                var rs = select.executeQuery("SELECT * FROM
Tranzactii")) {
            while (rs.next()) {
                tranzactii.add(new Tranzactie(
                        rs.getInt("CodProdus"),
                        rs.getString("DenumireProdus"),
                        rs.getDouble("Pret"),
                        rs.getInt("Cantitate")
                ));
            }
        }

        var produse = tranzactii.stream()
                .collect(Collectors.toMap(
                        Tranzactie::getCodProdus,
                        Tranzactie::getDenumireProdus,
                        (p1, p2) -> p1));

        var tranzactiiGrupate = tranzactii.stream()
```

```
.collect(Collectors.groupingBy(Tranzactie::getCodProdus));

        for (var codProdus : produse.keySet()) {

            var pretMediuIntrare =
tranzactiiGrupate.get(codProdus).stream()
                    .filter(t -> t.getCantitate() > 0)
                    .mapToDouble(Tranzactie::getPret)
                    .average().orElse(0);

            var stoc = tranzactiiGrupate.get(codProdus).stream()
                    .mapToInt(Tranzactie::getCantitate)
                    .sum();

            System.out.printf("%-30s %3d %6.2f RON%n",
                    produse.get(codProdus), stoc,
pretMediuIntrare);
        }
    }

    public static void main(String[] args) throws SQLException {
        genereazaBazaDeDate();
        afisareStocuri();
    }

public class ProblemaSerializareDeserializare {

    static Object Deserializare(Scanner sc) throws
ClassNotFoundException, NoSuchMethodException,
IllegalAccessException, InvocationTargetException,
InstantiationException, NoSuchFieldException {
        String linie = sc.nextLine();
        Class<?> cls = Class.forName(linie);
        Object obj = cls.getDeclaredConstructor().newInstance();
        int nrAtribute = Integer.parseInt(sc.nextLine()); //
sc.nextInt()
        for (int i = 0; i < nrAtribute; i++) {
            String numeAtr = sc.nextLine();
            Field fld = cls.getDeclaredField(numeAtr);
            String numeMetoda =
"set"+fld.getName().substring(0,1).toUpperCase()+fld.getName().sub
string(1);
            Method metodaSet = cls.getMethod(numeMetoda,
fld.getType());
            if (fld.getType() == String.class){
                String valoare = sc.nextLine();
                metodaSet.invoke(obj,valoare);
```

```
            }
            else if (fld.getType() == int.class) {
                String valoare = sc.nextLine();
                metodaSet.invoke(obj, Integer.parseInt(valoare));
            }
            else
                //throw new UnsupportedOperationException("Tipul
"+fld.getType().getName()+" nu este acceptat.");
            {
                metodaSet.invoke(obj, Deserializare(sc));
            }
        }
        return obj;
    }

    static void Serializare(PrintWriter fisier, Object obiect)
throws IllegalAccessException {
        Class<?> cls = obiect.getClass();
        fisier.println(cls.getName());
        fisier.println(cls.getDeclaredFields().length);
        for (Field fld : cls.getDeclaredFields()){
            fld.setAccessible(true);
            fisier.printf("%s\n", fld.getName());
            if (fld.getType() == int.class || fld.getType() ==
String.class)
                fisier.printf("%s\n", fld.get(obiect));
            else
                Serializare(fisier, fld.get(obiect));
        }
    }

    public static void main(String[] args) {
        List<Student> lstStud = new ArrayList<>();
        lstStud.add(new Student("Ion", 17, "Str. Zambilelor", new
Persoana("4770493281234l2")));
        lstStud.add(new Student("Marian", 21, "Str. Florilor", new
Persoana("3454232242434434")));
        String caleFisier = "fisier.txt";
        try (PrintWriter pw = new PrintWriter(caleFisier)) {
            for (Student s : lstStud) {
                Serializare(pw, s);
            }
        } catch (FileNotFoundException fne) {
            fne.printStackTrace();
        } catch (IllegalAccessException iae) {
            iae.printStackTrace();
        }
        try (Scanner sc = new Scanner(new File(caleFisier))) {
```

```
            while(sc.hasNext()) {
                Object obj = Deserializare(sc);
                System.out.println(obj.toString());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

//seminar 6
class Profesor {
    private final int idProfesor;
    private final String prenume;
    private final String nume;
    private final String departament;

    public Profesor(int idProfesor, String prenume, String nume,
String departament) {
        this.idProfesor = idProfesor;
        this.prenume = prenume;
        this.nume = nume;
        this.departament = departament;
    }

    public int getIdProfesor() {
        return idProfesor;
    }

    public String getPrenume() {
        return prenume;
    }

    public String getNume() {
        return nume;
    }

    public String getNumeComplet(){
        return getNume() + " " + getPrenume();
    }

    public String getDepartament() {
        return departament;
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Profesor{");
```

```
        sb.append("idProfesor=").append(idProfesor);
        sb.append(", prenume='").append(prenume).append('\'');
        sb.append(", nume='").append(nume).append('\'');
        sb.append(",
departament='").append(departament).append('\'');
        sb.append('}');
        return sb.toString();
    }
}

class Programare {
    private final String ziua;
    private final String interval;
    private final Profesor profesor;
    private final String disciplina;
    private final String sala;
    private final boolean esteCurs;
    private final String formatie;

    public Programare(String ziua, String interval, Profesor
profesor, String disciplina, String sala, boolean esteCurs, String
formatie) {
        this.ziua = ziua;
        this.interval = interval;
        this.profesor = profesor;
        this.disciplina = disciplina;
        this.sala = sala;
        this.esteCurs = esteCurs;
        this.formatie = formatie;
    }

    public String getZiua() {
        return ziua;
    }

    public String getInterval() {
        return interval;
    }

    public Profesor getProfesor() {
        return profesor;
    }

    public String getDisciplina() {
        return disciplina;
    }

    public String getSala() {
```

```
        return sala;
    }

    public boolean esteCurs() {
        return esteCurs;
    }

    public String getFormatie() {
        return formatie;
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Programare{");
        sb.append("ziua='").append(ziua).append('\'');
        sb.append(", interval='").append(interval).append('\'');
        sb.append(", profesor=").append(profesor);
        sb.append(", disciplina='").append(disciplina).append('\'');
        sb.append(", sala='").append(sala).append('\'');
        sb.append(", esteCurs=").append(esteCurs);
        sb.append(", formatie='").append(formatie).append('\'');
        sb.append('}');
        return sb.toString();
    }
}
class ComponentaSerie {
    private final String serie;
    private final int grupa;

    public ComponentaSerie(String serie, int grupa) {
        this.serie = serie;
        this.grupa = grupa;
    }

    public String getSerie() {
        return serie;
    }

    public int getGrupa() {
        return grupa;
    }

    @Override
    public String toString() {
        final StringBuilder sb = new StringBuilder("Serie{");
        sb.append("litera serie='").append(serie).append('\'');
        sb.append(", grupe='").append(grupa).append('\'');
        sb.append('}');
```

```
        return sb.toString();
    }
}

public class Orar {

    public static void main(String[] args) throws Exception {

        // 1. Citire date
        String grupa;
        Map<Integer, Profesor> profesori;
        List<Programare> programari;

        //initializare grupe ca liste de vectori
        List<String> grupa1 = Arrays.asList("1045"
,"1046","1047","1048","1049");
        List<String> grupa2 =
Arrays.asList("1050","1051","1052","1053","1054");
        List<String> grupa3 =
Arrays.asList("1055","1056","1057","1058");

        // initializare map
        Map<String, List<String>> componentaSerii = Map.ofEntries(
                new AbstractMap.SimpleEntry<String,
List<String>>("C", grupa1),
                new AbstractMap.SimpleEntry<String,
List<String>>("D", grupa2),
                new AbstractMap.SimpleEntry<String,
List<String>>("E", grupa3)
        );


        try (var fisier = new BufferedReader(new
FileReader("dataIN/profesori.txt"))) {
            profesori = fisier.lines()
                    .map(linie -> new Profesor(
                            Integer.parseInt(linie.split("\t")[0]),
                            linie.split("\t")[1],
                            linie.split("\t")[2],
                            linie.split("\t")[3])
                    )

.collect(Collectors.toMap(Profesor::getIdProfesor,
Function.identity()));
        }

        try (var fisier = new BufferedReader(new
FileReader("dataIN/programari.txt"))) {
```

```
            programari = fisier.lines()
                    .map(linie -> new Programare(
                            linie.split("\t")[0],
                            linie.split("\t")[1],

profesori.get(Integer.parseInt(linie.split("\t")[2])),
                            linie.split("\t")[3],
                            linie.split("\t")[4],

Boolean.parseBoolean(linie.split("\t")[5]),
                            linie.split("\t")[6])
                    )
                    .collect(Collectors.toList());
        }

        // 2. Prelucrari

        // Afișare lista cursuri în ordine alfabetică
        programari.stream()
                .filter(Programare::esteCurs)
                .map(Programare::getDisciplina)
                .distinct()
                .sorted()
                .forEach(disciplina ->
System.out.println(disciplina));

        // Afișare număr de activități pentru fiecare profesor
        System.out.printf("%30s %2s %2s%n", "Profesor", "C", "S");
        programari.stream()

.collect(Collectors.groupingBy(Programare::getProfesor))
                .forEach((profesor, programariProfesor) -> {
                    System.out.printf("%30s %2d %2d%n",
                            profesor.getNumeComplet(),

programariProfesor.stream().filter(Programare::esteCurs).count(),
                            programariProfesor.stream().filter(p ->
!p.esteCurs()).count());
                });

        // Lista departamentelor ordonate descrescator dupa numărul
de activități
        class Departament {
            String denumire;
            long numarActivitati;

            public Departament(String denumire, long
numarActivitati) {
```

```
                this.denumire = denumire;
                this.numarActivitati = numarActivitati;
            }

            @Override
            public String toString() {
                return String.format("%-75s - %d activități",
denumire, numarActivitati);
            }
        }

        programari.stream()
                .map(programare ->
programare.getProfesor().getDepartament())
                .distinct()
                .map(denumire -> {
                    var numarActivitati = programari.stream()
                            .filter(programare ->
programare.getProfesor().getDepartament().equals(denumire))
                            .count();
                    return new Departament(denumire,
numarActivitati);
                })
                .sorted((a, b) -> Long.compare(b.numarActivitati,
a.numarActivitati))
                .forEach(departament ->
System.out.println(departament));
        //    System.out.println("Orar grupe");

//        componentaSerii.stream()
//
.collect(Collectors.groupingBy(ComponentaSerie::getGrupa))
//                .forEach((profesor, programariProfesor) -> {
//                    System.out.printf("%30s %2d %2d%n",
//                            profesor.getNumeComplet(),
//                            profesor.get
//
programariProfesor.stream().filter(Programare::esteCurs).count(),
//                            programariProfesor.stream().filter(p
-> !p.esteCurs()).count());
//                });

        // afisare serii
        Set set = componentaSerii.entrySet();
        Iterator iterator = set.iterator();
        while(iterator.hasNext()) {
            Map.Entry mentry = (Map.Entry)iterator.next();
```

```
            System.out.println("Seria " + mentry.getKey() + " are
grupele:");
            System.out.println(mentry.getValue());
        }

//         static void afisareOrarGrupa(
//                 String grupa,
//                 List<Programare> programari,
//                 Map<String, List<String>> componentaSerii) {
//
//         }
    }
public class Biblioteca {
//    private static String dbURL =
"jdbc:derby:biblioteca;create=true"; // folosim derby.jar
//    private static String dbURL =
"jdbc:derby:biblioteca;create=true;server=MeDerby;password=MeDerby";
    // Pentru situatia in care serverul Derby ruleaza pe o alta
masina, respectiv
    // intr-o alta Java VM
    private static String dbURL =
"jdbc:derby://127.0.0.1:1527/biblioteca;create=true"; // folosim
derbyclient.jar
    private static String dbUser = "APP";
    private static String dbPassword = "APP";
    private static String dbName = "biblioteca";
    private static String tableName = "carte";

    // pentru interactiunea cu JDBC
    private static Connection conn;
    private static Statement sqlStmt;

    public static void main(String[] args) {
        Carte c1 = new Carte("Cota-0001", "Cel mai iubit dintre
pamanteni",
                "Marin Preda", 1980);
        Carte c2 = new Carte("Cota-0003", "ActiveMQ in Action",
                "Bruce Snyder", 2016);
        Carte c3 = new Carte("Cota-0007", "Un veac de singuratate",
                "Gabriel Garcia Marquez", 1985);

        createConnection();
        dropTable();
        createTable();
        insertCarte(c1);
        insertCarte(c2);
        insertCarte(c3);
//         updateCarte(c1);
```

```
//        deleteCarte(c2);
        selectCarti();
        shutDown();
    }

    private static void createConnection() {
        try {
//            conn = DriverManager.getConnection(dbURL, dbUser,
dbPassword);
            conn = DriverManager.getConnection(dbURL);
            System.out.println("Conexiune realizata cu succes!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private static void createTable() {
        try {
            sqlStmt = conn.createStatement();
            sqlStmt.execute("create table " + tableName +
                    " (cota varchar(16) primary key, titlu
varchar(64), autori varchar(64), an int)");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private static void dropTable() {
        try {
            sqlStmt = conn.createStatement();
            sqlStmt.execute("drop table " + tableName);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private static void insertCarte(Carte carte) {
        try {
            sqlStmt = conn.createStatement();
            sqlStmt.execute("insert into " + dbUser + "." +
                    tableName + " values ('" + carte.getCota() + "',
'" +
                    carte.getTitlu() + "', '" +
                    carte.getAutori() + "', " +
                    carte.getAn() + ")");
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
    }

    private static void updateCarte(Carte carte) {
        try {
            sqlStmt = conn.createStatement();
//              sqlStmt.executeUpdate("update " + tableName +
//                      " set cota = '" + carte.getCota() + "' where
an = " + carte.getAn());
            sqlStmt.executeUpdate("update " + tableName +
                    " set cota = 'Cota-0005' " + "where an = " +
carte.getAn());
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private static void deleteCarte(Carte carte) {
        try {
            sqlStmt = conn.createStatement();
            sqlStmt.executeUpdate("delete from " + tableName +
                    " where an = " + carte.getAn());
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private static void selectCarti() {
        try {
            sqlStmt =
conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
            ResultSet resultSet = sqlStmt.executeQuery("select *
from " + tableName +
                    " where cota like '%Cota%'");

            // Pozitionare la ultima linie din ResultSet
            resultSet.last();
            int nrArticole = resultSet.getRow();

            System.out.println("Au fost selectate " + nrArticole +
                    " articole din tabela " + tableName);

            ResultSetMetaData metaData = resultSet.getMetaData();
            int nrColoane = metaData.getColumnCount();

            // Tiparire la consola a header-ului tabelei carte
            for (int i=1; i<=nrColoane; i++) {
```

```
                System.out.print(metaData.getColumnName(i) +
"\t\t");
            }
            System.out.println();

            List<Carte> carteList = new ArrayList<Carte>();
            resultSet.beforeFirst();
            while (resultSet.next()) {
                Carte localCarte = new Carte();

                localCarte.setCota(resultSet.getString("cota"));
                localCarte.setTitlu(resultSet.getString("titlu"));
                localCarte.setAutori(resultSet.getString(3));
                localCarte.setAn(resultSet.getInt(4));

                carteList.add(localCarte);
                System.out.println(localCarte.toString());
            }

            // Extragerea si tiparirea listei de cote a cartilor
selectate
            List<String> cotaList = carteList.stream()
                    .map(x->x.getCota())
                    .collect(Collectors.toList());
            System.out.println(cotaList);

            // Extragerea si tiparirea listei anilor de aparitie a
cartilor selectate
            List<Integer> anList = carteList.stream()
                .map(x->x.getAn())
                .collect(Collectors.toList());
            System.out.println(anList);

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private static void shutDown() {
        try {
              if (sqlStmt != null)
                sqlStmt.close();

              if (conn != null)
                  conn.close();

        } catch (SQLException e) {
            e.printStackTrace();
```

```
        }
    }
```