JAVA SCHITE

Citirea dintr-un fisier text:

FileInputStream -> InputStreamReader -> BufferedReader;

String line = null;

While((line = bufferedReader.readLine()) != null)

{

String[ ] values = line.split(“ ”);

int codSectie = Integer.Parse(values[0]);

...  
}

bufferedReader.close();

Scriere intr-un fisier text:

FileOutputStream -> OutputStreamWriter -> BufferedWriter

For(Obiect o : lista)

{

Pt elemente de tip int/ float : bufferedWriter.write(String.valueOf(o.getCeva()));

Pt elemente de tip String: bufferedWriter.write(o.getString());

}

bufferedWriter.close();

Baza de date:

Main

{

Class.forName(“org.sqlite.JDBC”);

Connection connection = DriverManager.getConnection(jdbc:sqlite:database.db);

connection.setAutoCommit(false);

createTable(connection);

insertValues(connection);

selectValuesIntoList(connection);

}

Metodele :

public static void createTable(Connection connection)

{

String sqlDrop = “DROP TABLE IF EXISTS Pacient”;

String sqlCreate = “CREATE TABLE PACIENT (id int primary key, nume text, cod\_sectie int)”;

Statement statement = connection.createStatement();

statement.executeUpdate(sqlDrop);

statement.executeUpdate(sqlCreate);

statement.close();

connection.commit();

}

public static void insertValues(Connection connection)

{

String sqlInsert1 = “INSERT INTO PACIENT VALUES(1,’Terbea Ovidiu’,1)”;

Statement statement = connection.createStatement();

statement.executeUpdate(sqlInsert1);

statement.close();

connection.commit();

}

public static void selectValuesIntoList(Connection connection)

{

String sqlSelect = “SELECT \* FROM PACIENT”;

Statement statement = connection.createStatement ();

ResultSet rs = statement.executeQuery(sqlSelect);

while(rs.next())

{

int id = rs.getInt(“id”);

String nume = rs.getString(“nume”);

int cod\_sectie = rs.getInt(“cod\_sectie”);

Pacient p = new Pacient(id,nume,cod\_sectie);

pacienti.Add(p);

}

}

Stream-uri :

Exemple de stream-uri :

List<Integer> list = Arrays.*asList*(3,1,2,4,1,5,6,8,9);

long count =list.stream().filter(x -> x%2 ==0).count();

System.***out***.println(count);

List<Integer>sublist=list.stream().filter(x>x<7).sorted().distinct().collect(Collectors.*toLi*;

for(Integer x : sublist)

System.***out***.println(x);

List<String> strings = Arrays.*asList*("a","ab","bc","abc","bca");

strings.stream().filter(s -> s.startsWith("a")).forEach( s -> System.***out***.println(s));

String result = strings.stream().filter(s -> s.length() > 2).sorted().collect(Collectors.*joining*(", "))

System.***out***.println(result);

list.stream().distinct().map(x -> x\*x).sorted().forEach(System.***out***::println);

list.stream().distinct().map(Main::*cube*).sorted().forEach(System.***out***::println);

TCP

Server:

**try**(ServerSocket server = **new** ServerSocket(7777))

{

System.***out***.println("Server started!");

Socket socket = server.accept();

InputStream inputStream = socket.getInputStream();

ObjectInputStream objectInputStream = **new** ObjectInputStream(inputStream);

Tren t = (Tren)objectInputStream.readObject();

System.***out***.println("Tren" + t);

}

Client:

**try**(Socket socket = **new** Socket("localhost", 7777))

{

OutputStream outputStream = socket.getOutputStream();

ObjectOutputStream objectOutput = **new** ObjectOutputStream(outputStream);

objectOutput.writeObject(*t*);

objectOutput.close();

}

UDP

Client:

**try**(DatagramSocket socket = **new** DatagramSocket())

{

ByteArrayOutputStream byteArrayOutputStream = **new** ByteArrayOutputStream();

ObjectOutputStream objectOutput = **new** ObjectOutputStream(byteArrayOutputStream);

objectOutput.writeObject(*t*);

objectOutput.close();

**byte**[] buffer = byteArrayOutputStream.toByteArray();

DatagramPacket packetToBeSend = **new** DatagramPacket(buffer, buffer.length, InetAddress.*getByName*("localhost"), 7777);

socket.send(packetToBeSend);

}

Server:

**try**(DatagramSocket socket = **new** DatagramSocket(7777))

{

System.***out***.println("Server started!");

**byte**[] buffer = **new** **byte**[2560];

DatagramPacket packetToBeReceived = **new** DatagramPacket(buffer, buffer.length);

socket.receive(packetToBeReceived);

ObjectInputStream objectInputStream = **new** ObjectInputStream(**new** ByteArrayInputStream(buffer));

Tren tren = (Tren)objectInputStream.readObject();

System.***out***.println("Client requested: " + tren);

}

Cloneable

Se face implement implement Clonable

*@Override*

public Object clone() throws CloneNotSupportedException

{

Car copy = (Car)super.clone();

copy.producer= producer;

copy.model = model;

copy.speed = speed;

copy.capacity = capacity;

return copy;

}

Comparable

Se face implement Comparable<obiect>

*@Override*

public int compareTo(Car o)

{

return Comparator.comparingInt(Car::getCapacity).thenComparing(Car::getName).compare(this,o);

}

JSON

Scriere in fisier JSON:

List<JSONObject> pacientiJson = **new** ArrayList<JSONObject>();

**for**(Pacient p : *pacienti*)

{

JSONObject o = **new** JSONObject();

o.put("cnp", p.getCnpPacient());

o.put("nume",p.getNumePacient());

o.put("cod\_sectie", p.getCodSectie());

pacientiJson.add(o);

}

FileWriter file = **new** FileWriter("pacienti.json");

**for**(JSONObject obiect : pacientiJson)

{

file.write(obiect.toString());

}

file.close();

}

}