

2011 - 2012

Programare Orientata spre Obiecte (*Object-Oriented Programming*)

a.k.a. Programare Obiect-Orientata

Titular curs: Eduard-Cristian Popovici

Suport curs: <http://discipline.elcom.pub.ro/POO-Java/>

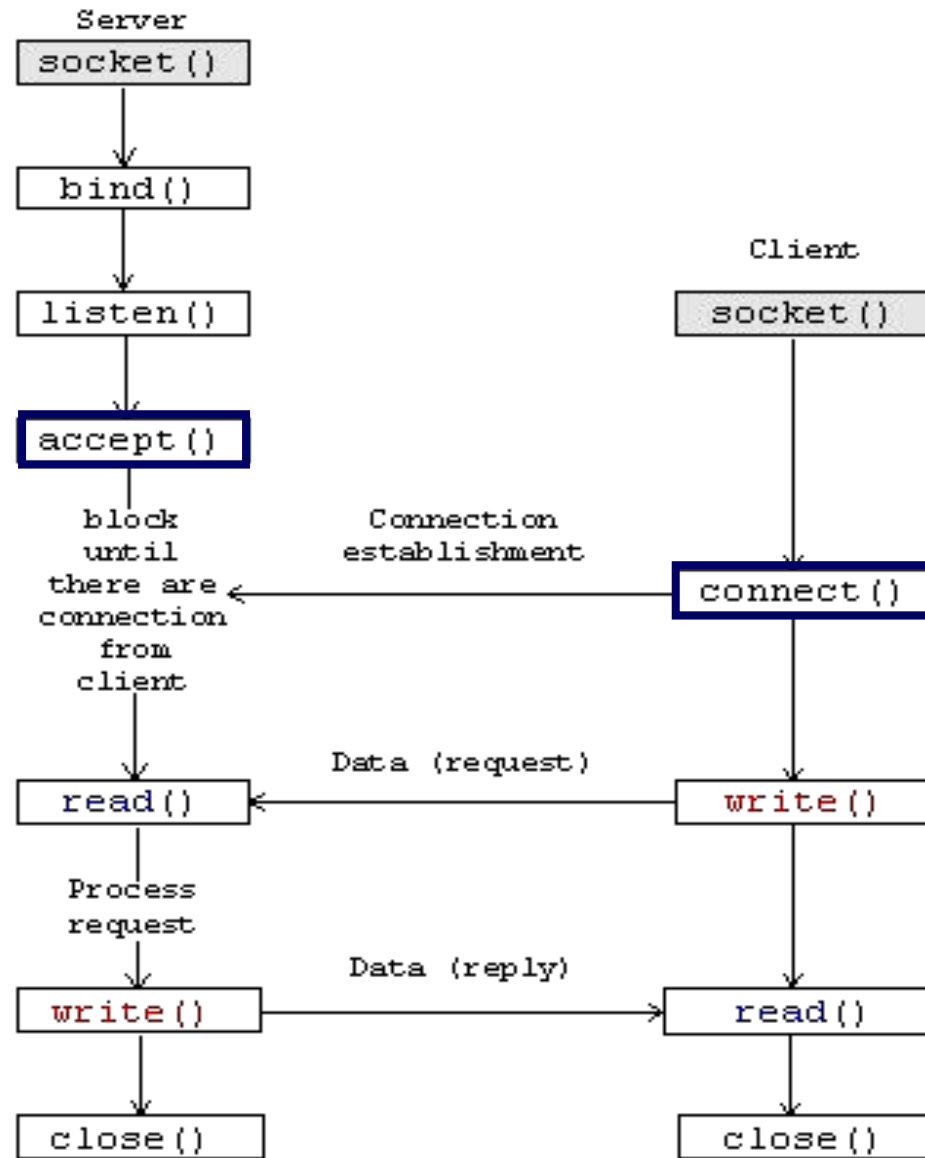
3. Programarea la nivel socket cu Java

3.3. Socketuri flux (TCP) Java

Socketuri flux (TCP) Java



Crearea si utilizarea socket-urilor TCP in C/C++

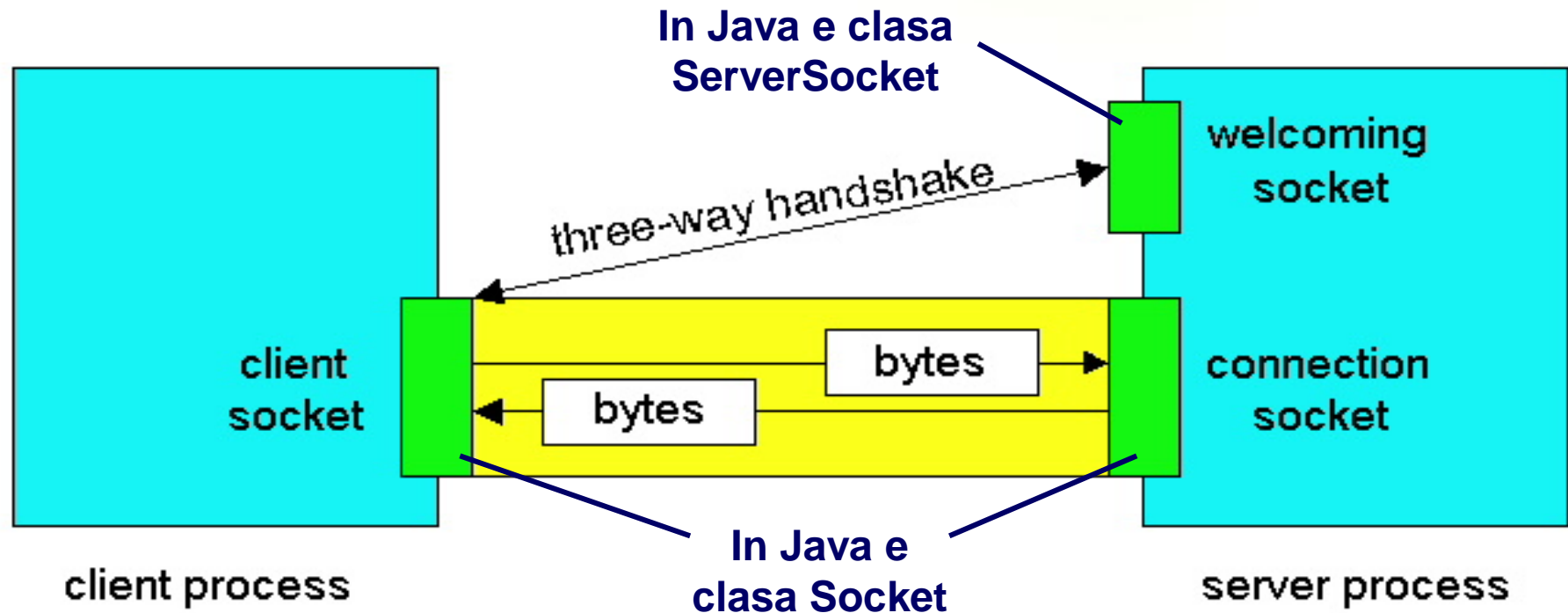




Crearea si utilizarea socket-urilor TCP

Clasa Socket reprezinta punctul terminal al unei conexiuni TCP intre doua masini (eventual un client si un server)

Clasa ServerSocket reprezinta socket-ul (aflat eventual pe un server bazat pe TCP) care asteapta si accepta cereri de conexiune (eventual de la un client bazat pe TCP)

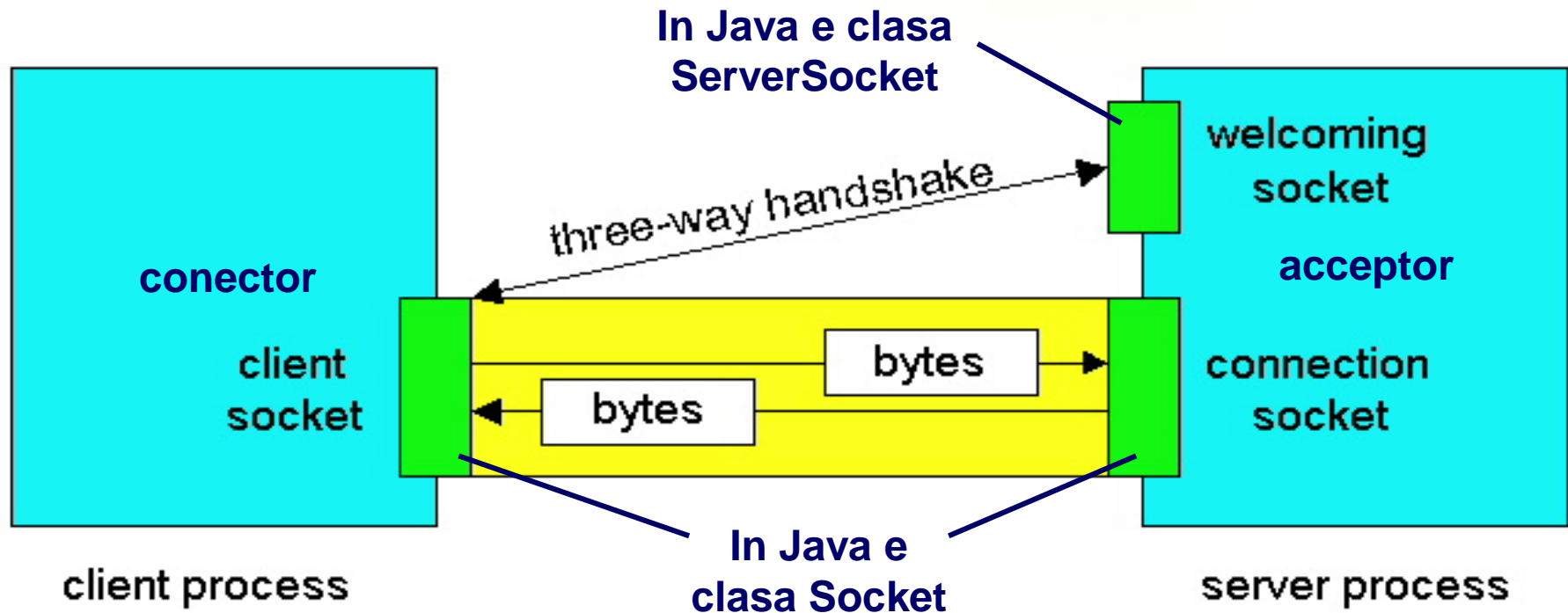




Crearea si utilizarea socket-urilor TCP

Masina **conector** (de obicei **clientul**) creaza un punct terminal Socket in momentul in care cererea sa de conexiune este lansata si acceptata

Masina **acceptor** (de obicei **serverul**) asteapta cereri pe ServerSocket, creeaza un Socket in momentul in care primeste si accepta o cerere de conexiune, si continua sa asculte si sa astepte alte cereri pe ServerSocket.





Crearea si utilizarea socket-urilor TCP

Principalii constructori ai clasei **Socket**:

Socket()

Creeaza un socket flux neconectat, cu implementarea implicita platformei.

Socket (String host, int port)

Creeaza un socket flux si il conecteaza la portul de numar specificat, la masina a carui nume este specificat.

Socket (InetAddress address, int port)

Creeaza un socket flux si il conecteaza la portul de numar specificat, la adresa IP specificata.

Principalii constructori ai clasei **ServerSocket**:

ServerSocket()

Creaza un socket pentru server (de tip acceptor) nelegat la vreun port.

ServerSocket(int port)

Creaza un *socket* pentru server (de tip acceptor) legat la portul local de numar specificat. Valoarea 0 va conduce la crearea unui *socket* legat la un port liber nespecificat explicit. Numarul de indicatii privind cererile de conexiune care pot sta in asteptare la un moment dat este implicit 50.



Crearea si utilizarea socket-urilor TCP

Declaratiile si descrierea catorva metode ale clasei **Socket**:

void	close() Inchide <i>socket-ul</i> curent.
InetAddress	getInetAddress() Returneaza un obiect care incapsuleaza adresa IP la care este conectat <i>socket-ul</i> curent.
InputStream	getInputStream() Returneaza un flux de intrare a octetilor dinspre <i>socket-ul</i> curent.
InetAddress	getLocalAddress() Returneaza un obiect care incapsuleaza adresa IP locala la care <i>socket-ul</i> curent este legat.
int	getLocalPort() Returneaza numarul portului local la care <i>socket-ul</i> curent este legat.



Crearea si utilizarea socket-urilor TCP

Declaratiile si descrierea catorva metode ale clasei Socket:

OutputStream	getOutputStream() Returneaza un flux de iesire a octetilor catre socket-ul curent.
int	getPort() Returneaza numarul portului la care <i>socket-ul</i> curent este conectat.
int	getReceiveBufferSize() Returneaza valoarea optiunii SO_RCVBUF pentru <i>socket-ul</i> curent, adica dimensiunea <i>buffer-ului</i> utilizat de platforma pentru fluxul de intrare dinspre <i>socket</i> .
int	getSendBufferSize() Returneaza valoarea optiunii SO_SNDBUF pentru <i>socket-ul</i> curent, adica dimensiunea <i>buffer-ului</i> utilizat de platforma pentru fluxul de iesire catre <i>socket</i> .



Crearea si utilizarea socket-urilor TCP

Declaratiile si descrierea catorva metode ale clasei Socket:

int	<p>getSoTimeout()</p> <p>Returneaza valoarea optiunii SO_TIMEOUT pentru <i>socket-ul</i> curent, adica durata in milisec. cat apelul read() asupra fluxului de intrare asociat <i>socket-ului</i> curent blocheaza aplicatia, asteptand sosirea unor octeti prin flux. Valoarea 0 semnifica asteptare (blocare) la infinit. Valoarea nu poate fi negativa. La expirarea unei valori nenule a <i>timeout-ului</i>, e generata exceptia <code>java.net.SocketTimeoutException</code>, dar <i>socket-ul</i> continua functionarea</p>
int	<p>getTrafficClass()</p> <p>Obtine clasa de trafic (<i>traffic class</i>) sau tipul de serviciu (ToS) din antetul IP al pachetelor trimise prin <i>socket-ul</i> curent.</p>
boolean	<p>isClosed()</p> <p>Returneaza true daca <i>socket-ul</i> curent e inchis, altfel returneaza false.</p>
boolean	<p>isConnected()</p> <p>Returneaza true daca <i>socket-ul</i> curent este conectat cu succes, altfel returneaza false.</p>



Crearea si utilizarea socket-urilor TCP

Declaratiile si descrierea catorva metode ale clasei Socket:

boolean	<p>isInputShutdown() Returneaza true daca fluxul de intrare al <i>socket-ului</i> curent este "la sfarsit de flux" (EOF), altfel returneaza false.</p>
boolean	<p>isOutputShutdown() Returneaza true daca fluxul de iesire al <i>socket-ului</i> curent este valid, altfel returneaza false.</p>
void	<p>setReceiveBufferSize(int size) Stabileste valoarea optiunii SO_RCVBUF pentru <i>socket-ul</i> curent, adica dimensiunea <i>buffer-ului</i> utilizat de platforma pentru fluxul de intrare dinspre <i>socket</i>.</p>
void	<p>setSendBufferSize(int size) Stabileste valoarea optiunii SO_SNDBUF pentru <i>socket-ul</i> curent, adica dimensiunea <i>buffer-ului</i> utilizat de platforma pentru fluxul de iesire catre <i>socket</i>.</p>



Crearea si utilizarea socket-urilor TCP

Declaratiile si descrierea catorva metode ale clasei Socket:

void	setSoTimeout(int timeout) Stabileste valoarea optiunii SO_TIMEOUT pentru <i>socket-ul</i> curent, adica durata in ms cat apelul read() blocheaza aplicatia, asteptand sosirea unor octeti prin fluxul de intrare. Valoarea 0 semnifica asteptare la infinit. Valoarea nu poate fi negativa. La expirarea unei valori nenule e generata <code>java.net.SocketTimeoutException</code> , dar <i>socket-ul</i> continua functionarea
void	setTrafficClass(int tc) Stabileste clasa de trafic (<i>traffic class</i>) sau tipul de serviciu (ToS) din antetul IP al pachetelor trimise prin <i>socket-ul</i> curent.
void	shutdownInput() Plaseaza fluxul de intrare al <i>socket-ului</i> curent "la sfarsit de flux" (EOF).
void	shutdownOutput() Invalideaza fluxul de iesire al <i>socket-ului</i> curent (implicit este valid).
String	toString() Returneaza un String continand informatii privind <i>socket-ul</i> curent.



Crearea si utilizarea socket-urilor TCP

Declaratiile si descrierea catorva metode ale clasei **ServerSocket**:

Socket	accept() Asteapta cereri de conexiune facute catre <i>socket-ul</i> curent si le accepta. Metoda blocheaza executia pana cand e primita o cerere de conexiune. Metoda returneaza un obiect Socket prin care se poate desfasura comunicatia utilizand fluxuri de octeti.
void	close() Inchide <i>socket-ul</i> curent.
InetAddress	getInetAddress() Returneaza adresa IP locala a <i>socket-ului</i> curent.
int	getLocalPort() Returneaza numarul de port local pe care asculta <i>socket-ul</i> curent.



Crearea si utilizarea socket-urilor TCP

Declaratiile si descrierea catorva metode ale clasei `ServerSocket`:

int	<p><code>getReceiveBufferSize()</code></p> <p>Returneaza valoarea optiunii <code>SO_RCVBUF</code> pentru <i>ServerSocket-ul</i> curent, adica dimensiunea <i>buffer-ului</i> propus a fi utilizat de pentru fluxurile de intrare dinspre <i>socket-urile</i> obtinute prin acceptarea conexiunilor prin <i>socket-ul</i> curent.</p>
int	<p><code>getSoTimeout()</code></p> <p>Returneaza valoarea optiunii <code>SO_TIMEOUT</code> pentru <i>socket-ul</i> curent, adica durata in milisecunde cat apelul <code>accept()</code> asupra <i>socket-ului</i> curent blocheaza aplicatia, asteptand sosirea unor cereri de conexiune. Valoarea 0 semnifica asteptare (blocare) la infinit. Valoarea nu poate fi negativa. La expirarea unei valori nenule a <i>time-out-ului</i>, este generata o exceptie <code>java.net.SocketTimeoutException</code>, dar <i>socket-ul</i> curent continua sa functioneze.</p>
boolean	<p><code>isClosed()</code></p> <p>Returneaza true daca <i>socket-ul</i> curent e inchis, altfel returneaza false.</p>



Crearea si utilizarea socket-urilor TCP

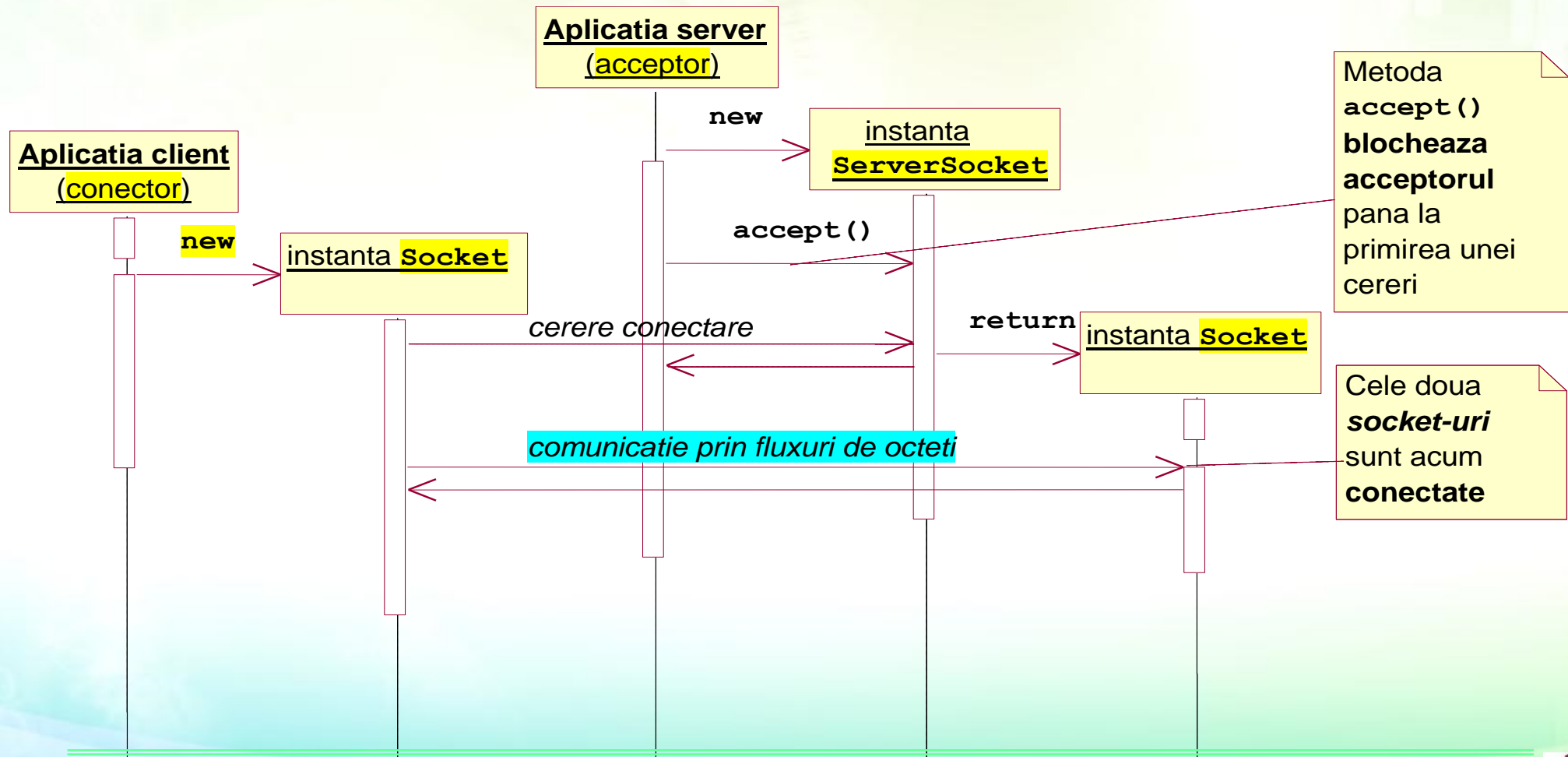
Declaratiile si descrierea catorva metode ale clasei `ServerSocket`:

void	<p><code>setReceiveBufferSize(int size)</code> Stabileste valoarea optiunii <code>SO_RCVBUF</code> pentru fluxurile de intrare dinspre <i>socket-urile</i> obtinute prin acceptarea conexiunilor prin <i>socket-ul</i> curent.</p>
void	<p><code>setSoTimeout(int timeout)</code> Stabileste valoarea optiunii <code>SO_TIMEOUT</code> pentru <i>socket-ul</i> curent, adica durata in milisecunde cat apelul <code>accept()</code> blocheaza aplicatia, asteptand sosirea unor cereri de conexiune. Valoarea 0 semnifica asteptare (blocare) la infinit. Valoarea nu poate fi negativa. La expirarea unei valori nenule a <i>time-out-ului</i>, este generata o exceptie <code>java.net.SocketTimeoutException</code>, dar <i>socket-ul</i> curent continua sa functioneze.</p>
String	<p><code>toString()</code> Returneaza un String continand informatii (adresa IP si numarul de port) privind <i>socket-ul</i> curent.</p>



Crearea si utilizarea socket-urilor TCP

Secventa tipica a mesajelor schimbate intre cele doua masini:





Crearea si utilizarea socket-urilor TCP

Secventa tipica pentru crearea socket-ului unei aplicatii conector (client)

```
String adresaServer = "localhost"; // Adresa serverului
int portServer = 2000; // Portul serverului

// Crearea socketului (implicit realizata conexiunea cu serverul)
Socket socketTCPClient = new Socket(adresaServer, portServer);
```

Secventa tipica pentru crearea socket-ului server al unei aplicatii acceptor (server):

```
int portServer = 2000; // Portul serverului

// Crearea socketului server (care accepta conexiunile)
ServerSocket serverTCP = new ServerSocket(portServer);
```

Secventa tipica pentru crearea socket-ului aflat la server pentru tratarea conexiunii TCP cu un client

```
Socket conexiuneTCP = serverTCP.accept();
```




Crearea si utilizarea socket-urilor TCP

Dupa stabilirea conexiunii, trebuie utilizate metodele **getInputStream()** si **getOutputStream()** ale clasei **Socket** pentru a obtine fluxuri de octeti, de intrare respectiv iesire, pentru comunicatia intre aplicatii

```
// Obtinerea fluxului de intrare octeti TCP
InputStream inTCP = socketTCPClient.getInputStream() ;

// Obtinerea fluxului de intrare caractere dinspre retea
InputStreamReader inTCPCaractere =
    new InputStreamReader(inTCP) ;

// Adaugarea facilitatilor de stocare temporara
BufferedReader inRetea = new BufferedReader(inTCPCaractere) ;

// Obtinerea fluxului de iesire octeti TCP
OutputStream outTCP = socketTCPClient.getOutputStream() ;

// Obtinerea fluxului de iesire spre retea,
// cu facilitate de afisare (similare consolei de iesire)
PrintStream outRetea = new PrintStream(outTCP) ;
```



Crearea si utilizarea socket-urilor TCP

Secventa tipica pentru trimiterea de date:

```
// Crearea unui mesaj
String mesajDeTrimis = "Continut mesaj";

// Scrierea catre retea (trimiterea mesajului)
outRetea.println(mesajDeTrimis);

// Fortarea trimiterii
outRetea.flush();
```

Secventa tipica pentru primirea de date:

```
// Citirea dinspre retea (receptia unui mesaj)
String mesajPrimit = inRetea.readLine();

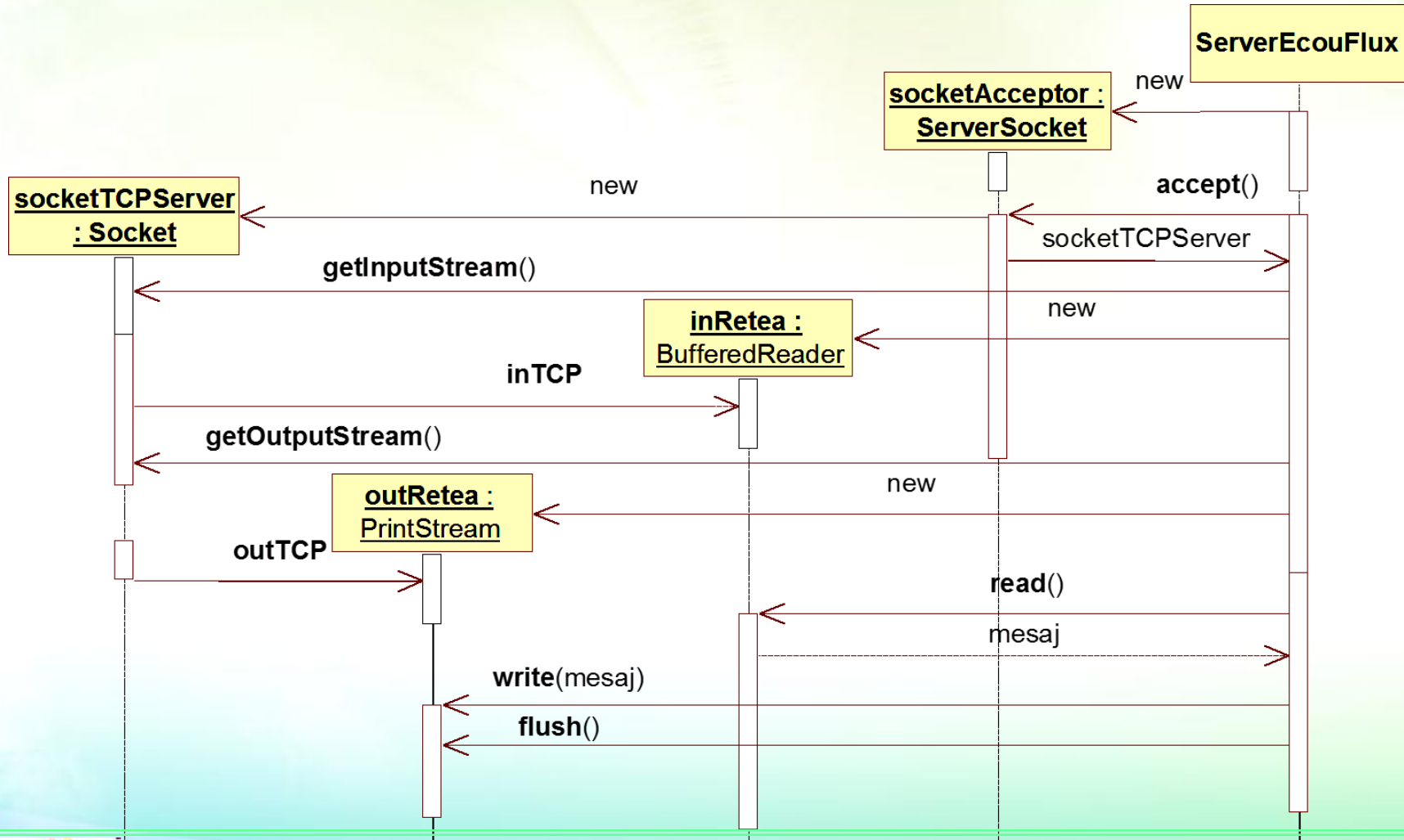
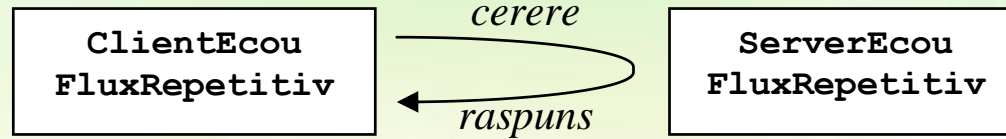
// Afisarea mesajului primit
System.out.println(mesajPrimit);
```

Cienti si servere bazate pe socketuri TCP

Socketuri flux (TCP) Java

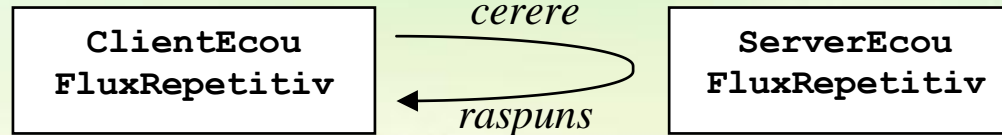


Server TCP ecou





Server TCP ecou

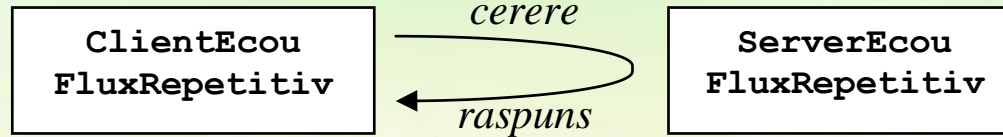


```

1  import java.net.*;
2  import java.io.*;
3  import javax.swing.JOptionPane;
4
5  public class ServerEcouFluxRepetiv {           // Server ecou flux
6      public static void main (String args[]) throws IOException {
7
8          int portServer = Integer.parseInt(JOptionPane.showInputDialog(
9              "Introduceti numarul de port dorit: "));
10
11         // Crearea socketului server (care accepta conexiunile)
12         ServerSocket socketAcceptor = new ServerSocket(portServer);
13         System.out.println("Server in asteptare pe portul "+portServer+"...");
14
15         // Blocare in asteptarea cererii de conexiune - in momentul acceptarii
16         // cererii se creaza socketul care serveste conexiunea
17         Socket socketTCPServer = socketAcceptor.accept();
18         System.out.println("Conexiune TCP pe portul " + portServer + "...");
19
20         // Crearea fluxurilor de caractere conectate la fluxurile de octeti
21         // obtinute de la socketul TCP
22         PrintStream outRetea = new PrintStream(socketTCPServer.getOutputStream());
23         BufferedReader inRetea = new BufferedReader(
24             new InputStreamReader(socketTCPServer.getInputStream()));
  
```



Server TCP ecou



```

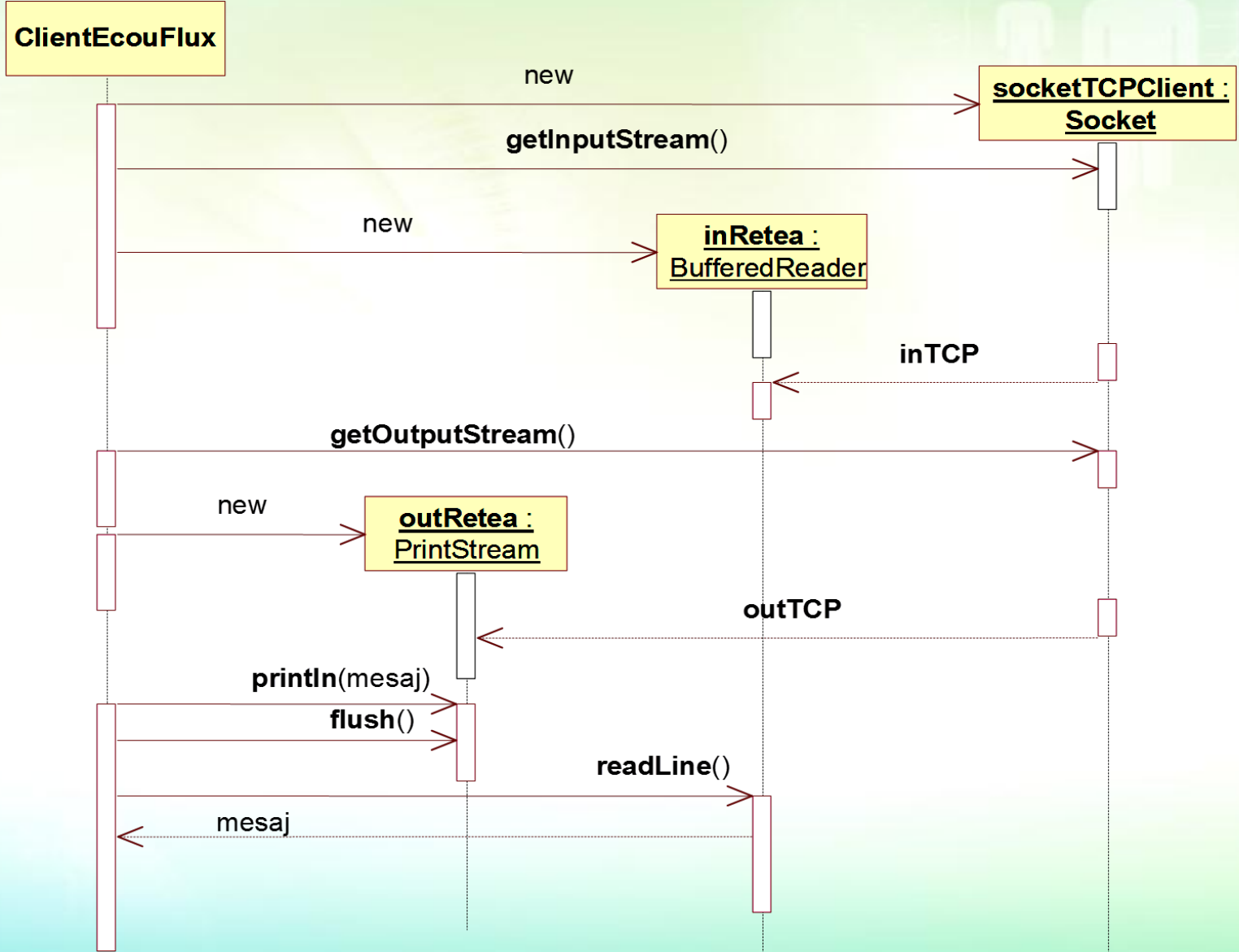
25     while (true) {
26
27         // Citirea unei linii din fluxul de intrare TCP
28         String mesajPrimit = inRetea.readLine();
29
30         // Afisarea liniei citite la consola de iesire
31         System.out.println("Mesaj primit: " + mesajPrimit);
32
33         // Scrierea liniei in fluxul de iesire TCP, cu fortarea trimiterii
34         outRetea.println(mesajPrimit);
35         outRetea.flush();
36
37         // Testarea conditiei de oprire a servirii
38         if (mesajPrimit.equals(".")) break;
39     }
40
41     // Inchiderea socketului (si implicit a fluxurilor)
42     socketTCPServer.close();
43
44     System.out.println("Bye!");
45 }
46
47 }
  
```



Socketuri flux (TCP) Java

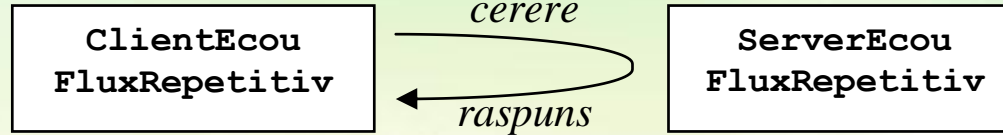


Client TCP ecou





Client TCP ecou



```

1  import java.net.*;
2  import java.io.*;
3  import javax.swing.JOptionPane;
4
5  public class ClientEcouFluxRepetitiv {    // Client pentru server ecou flux
6  public static void main (String args[]) throws IOException {
7
8      String adresaServer = JOptionPane.showInputDialog(
9          "Introduceti adresa IP a serverului: ");
10     int portServer = Integer.parseInt(JOptionPane.showInputDialog(
11         "Introduceti numarul de port al serverului: "));
12     // Creare socket
13     Socket socketTCPClient = new Socket(adresaServer, portServer);
14
15     System.out.println("Conexiune TCP cu serverul " + adresaServer +
16         ":" + portServer + "...");
17     System.out.println("Pentru oprire introduceti '.' si <Enter>");
18
19     // Creare fluxuri
20     PrintStream outRetea = new PrintStream(socketTCPClient.getOutputStream());
21
22     BufferedReader inRetea = new BufferedReader(
23         new InputStreamReader(socketTCPClient.getInputStream()));
24
  
```




Client TCP ecou

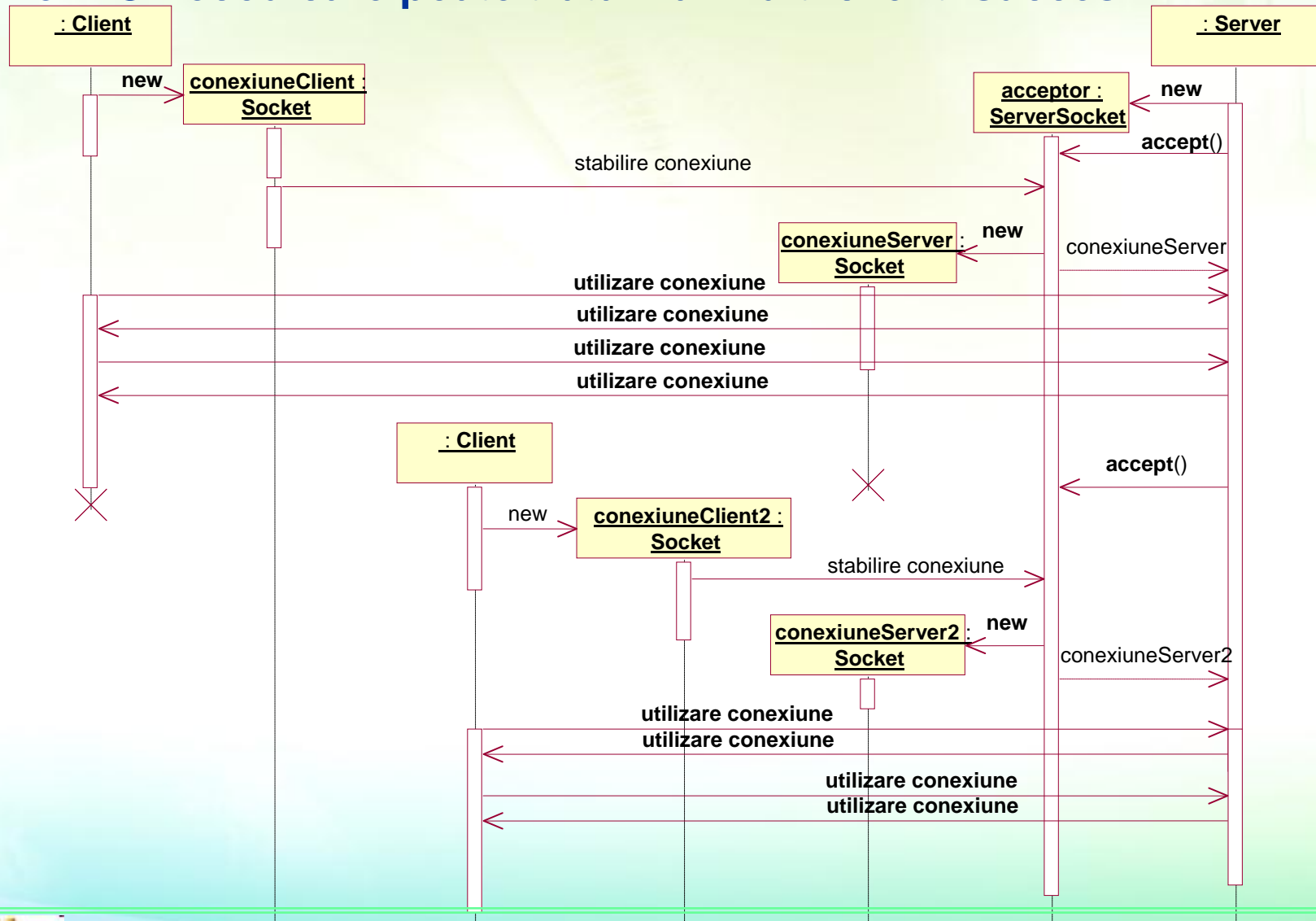
```

25     while (true) {           // Cat timp conditia de oprire nu este indeplinita
26
27         String mesajTrimis = JOptionPane.showInputDialog("Se trimite: "); // Mesaj
28
29         outRetea.println(mesajTrimis);           // Scrierea in fluxul de iesire TCP
30
31         outRetea.flush();
32
33         String mesajPrimit = inRetea.readLine(); // Citirea din fluxul de intrare
34
35         System.out.println("S-a primit: " + mesajPrimit); // Afisarea la consola
36
37         if (mesajPrimit.equals(".")) break;     // Testarea conditiei de oprire
38
39     }
40
41     socketTCPClient.close(); // Inchiderea socketului (si implicit a fluxurilor)
42
43     System.out.println("Bye!");
44
45 }
46
47 }

```



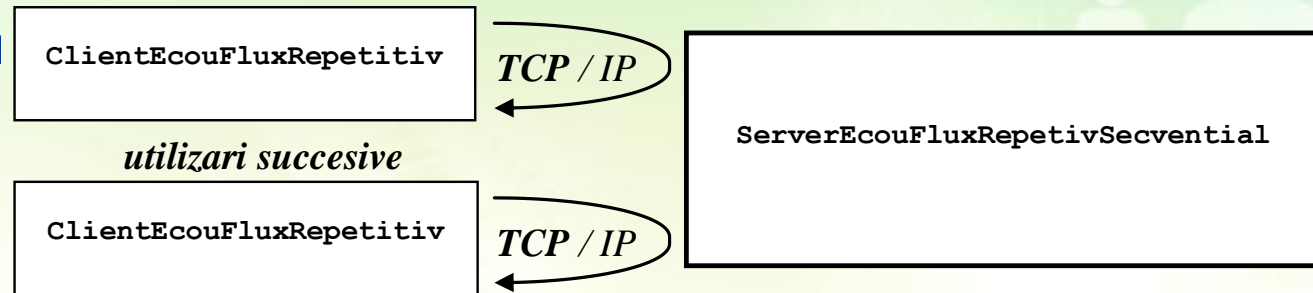
Server TCP ecou care poate trata mai multi clienti succesiv



Socketuri flux (TCP) Java



Server TCP ecou care poate trata mai multi clienti succesiv



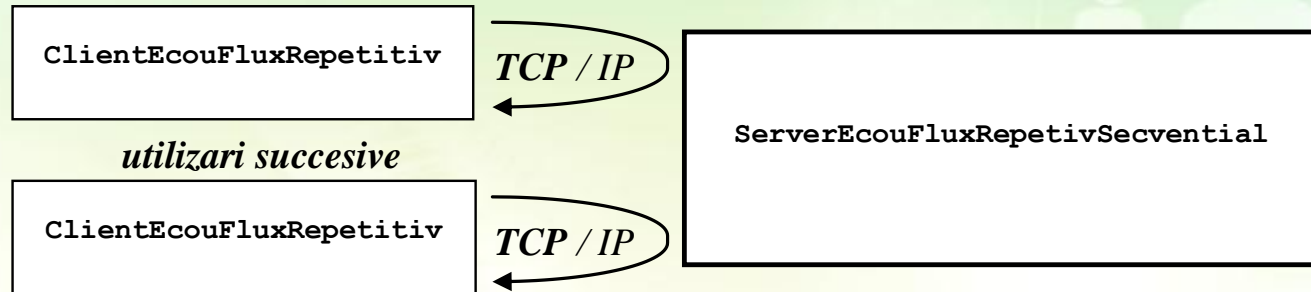
```

1 // Servirea mai multor clienti succesiv (in mod secvential)
2 while (true) {
3
4     System.out.println("Server in asteptare pe port "+portServer+"...");
5
6     // Blocare in asteptarea cererii de conexiune - in momentul
7     // acceptarii cererii se creaza socketul care serveste conexiunea
8     Socket conexiuneTCP = serverTCP.accept();
9
10    System.out.println("Conexiune TCP pe portul " + portServer + "...");
11
12    // Crearea fluxurilor de caractere conectate la fluxurile de octeti
13    // obtinute de la socketul TCP
14    PrintStream outRetea = new
15        PrintStream(conexiuneTCP.getOutputStream());
16
17    BufferedReader inRetea = new BufferedReader(new InputStreamReader(
18        conexiuneTCP.getInputStream()));
  
```

Socketuri flux (TCP) Java

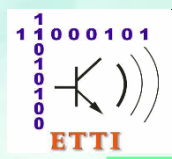


Server TCP ecou care poate trata mai multi clienti succesiv



```

19 // Servirea clientului curent
20 while (true) {
21
22 // Citirea unei linii din fluxul de intrare TCP
23 String mesajPrimit = inRetea.readLine();
24
25 // Afisarea liniei citite la consola de iesire
26 System.out.println("Mesaj primit: " + mesajPrimit);
27
28 // Scrierea liniei in fluxul de iesire TCP, cu fortarea trimiterii
29 outRetea.println(mesajPrimit);
30 outRetea.flush();
31
32 if (mesajPrimit.equals(".")) break; // Testarea conditiei de oprire
33 }
34
35 // Inchiderea socketului (si implicit a fluxurilor)
36 conexiuneTCP.close();
37 System.out.println("Bye!");
38 }
  
```



Socketuri flux (TCP) Java



Java Threads



Fire de executie Java (*Java Threads*)

Programele de calcul simple sunt **secventiale**

- fiecare avand **un inceput**, o **secventa de executii** si **un sfarsit**

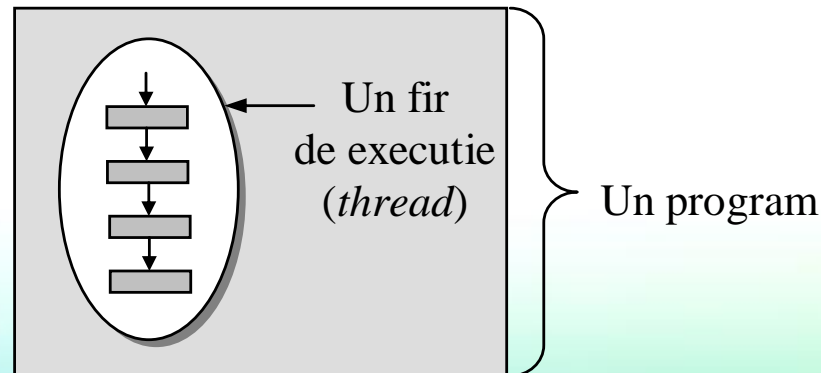
In orice moment pe durata executiei unui astfel de program

- exista **un singur punct de executie**

Un **fir de executie** (*thread*), sau mai simplu, un fir

- este **similar** acestor programe secventiale, in sensul ca **are un inceput**, o **secventa de executii** si **un sfarsit**

- si in orice moment pe durata executiei firului exista **un singur punct de executie**





Fire de executie Java (*Java Threads*)

Totusi, **un fir nu este** el insusi un **program**

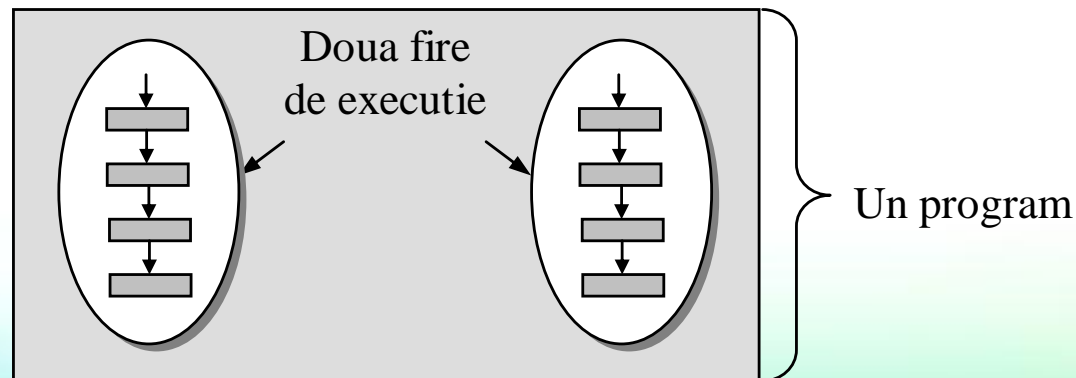
- deoarece **nu poate fi executat de sine statator**

In schimb

- **firul** este executat (**ruleaza**) **intr-un program**

Posibilitatea utilizarii mai multor fire de executie **intr-un singur program**, ruland (fiind executate) **in acelasi timp** si realizand **diferite sarcini** (nu in mod necesar diferite)

- este numita **multifilaritate** (*multithreading*)





Fire de executie Java (*Java Threads*)

Pentru a crea un nou fir de executie exista doua modalitati

1. Se poate declara o clasa ca subclasa a clasei Thread

- subclasa care trebuie sa **rescrie codul** (*override*) metodei **run()** a clasei Thread (care nu contine nici un cod)
- **noul fir** de executie fiind **creat prin alocarea si lansarea unei instante** a subclasei

```
class FirT extends Thread {
    public void run() {
        // codul firului de executie
    }
}
```

Formatul pentru **crearea unei instante a subclasei**

```
FirT fir = new FirT(); // extinde Thread
```




Fire de executie Java (*Java Threads*)

2. Ca alternativa, **se poate declara o clasa care implementeaza interfata Runnable**

- interfata care **contine doar declaratia** unei metode **run()** (si clasa **Thread** implementeaza interfata **Runnable**)
- **se creeaza o instanta** a noii clase care
 - **e pasata constructorului** la crearea unei noi instante a clasei **Thread** si
 - **se lanseaza** acea instanta a clasei **Thread**

```
class FirR implements Runnable {
    public void run() {
        // codul firului de executie
    }
}
```

Formatul pentru **crearea unei instante a noii clase si a instantei clasei Thread**:

```
FirR r = new FirR(); // implementeaza Runnable  
Thread fir = new Thread(r); // este un Thread
```



Fire de executie Java (*Java Threads*)

In ambele cazuri formatul pentru **lansarea noului fir** de executie, este urmatorul:

```
fir.start(); // apeleaza fir.run()
```

Variante compacte pentru **crearea si lansarea noilor fire** de executie:

```
new FirT().start(); // nu exista variabila de tip FirT
// care sa refere explicit firul
```

sau

```
FirR r = new FirR();
new Thread(r).start(); // nu exista variabila tip Thread
// care sa refere explicit firul
```

sau

```
Thread fir = new Thread(new FirR());
fir.start(); // nu exista variabila de tip FirR
```

sau

```
new Thread(new FirR()).start(); // nu exista variabila tip Thread
// care sa refere explicit firul
// si nici variabila de tip FirR
```



Lucrul cu fire de executie Java (*Java Threads*)

```

1  public class FirSimplu extends Thread { // obiectele din clasa curenta sunt fire
2
3      public FirSimplu(String str) { // constructor
4          super(str); // invocarea constructorului Thread(String)
5      } // al superclasei Thread
6
7      public void run() { // "metoda principala" a thread-ului curent
8
9          for (int i = 0; i < 5; i++) {
10             System.out.println(i + " " + getName()); // obtinerea numelui threadului
11
12             try {
13                 sleep((long) (Math.random() * 1000)); // thread-ul "doarme" 0..1 sec
14             } catch (InterruptedException e) {}
15
16             }
17             System.out.println("Gata! " + getName()); // obtinerea numelui threadului
18         }
19
20     public static void main (String[] args) {
21         new FirSimplu("Unu").start(); // crearea si "lansarea" threadului
22     }
23 }

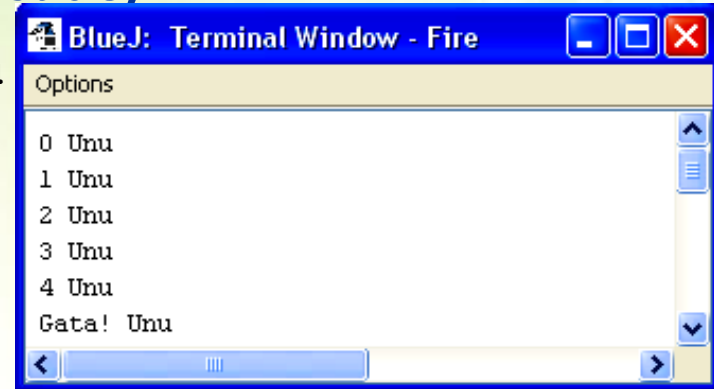
```



Lucrul cu fire de executie Java (*Java Threads*)

Rezultatul executiei programului `FirSimplu`

Clasa `DemoDouaFire` lanseaza doua fire de executie de tip `FirSimplu` executate **concurrent**



```

1 public class DemoDouaFire {
2     public static void main (String[] args) {
3         new FirSimplu("Unu").start(); // "lansarea" primului thread
4         new FirSimplu("Doi").start(); // "lansarea" celui de-al doilea thread
5     }
6 }

```

Rezultatele a doua executii succesive `DemoDouaFire`

Firele au evolutii diferite, in functie de durata intarzierii introdusa in linia de cod a clasei `FirSimplu`

```

sleep((long) (Math.random() * 1000));

```

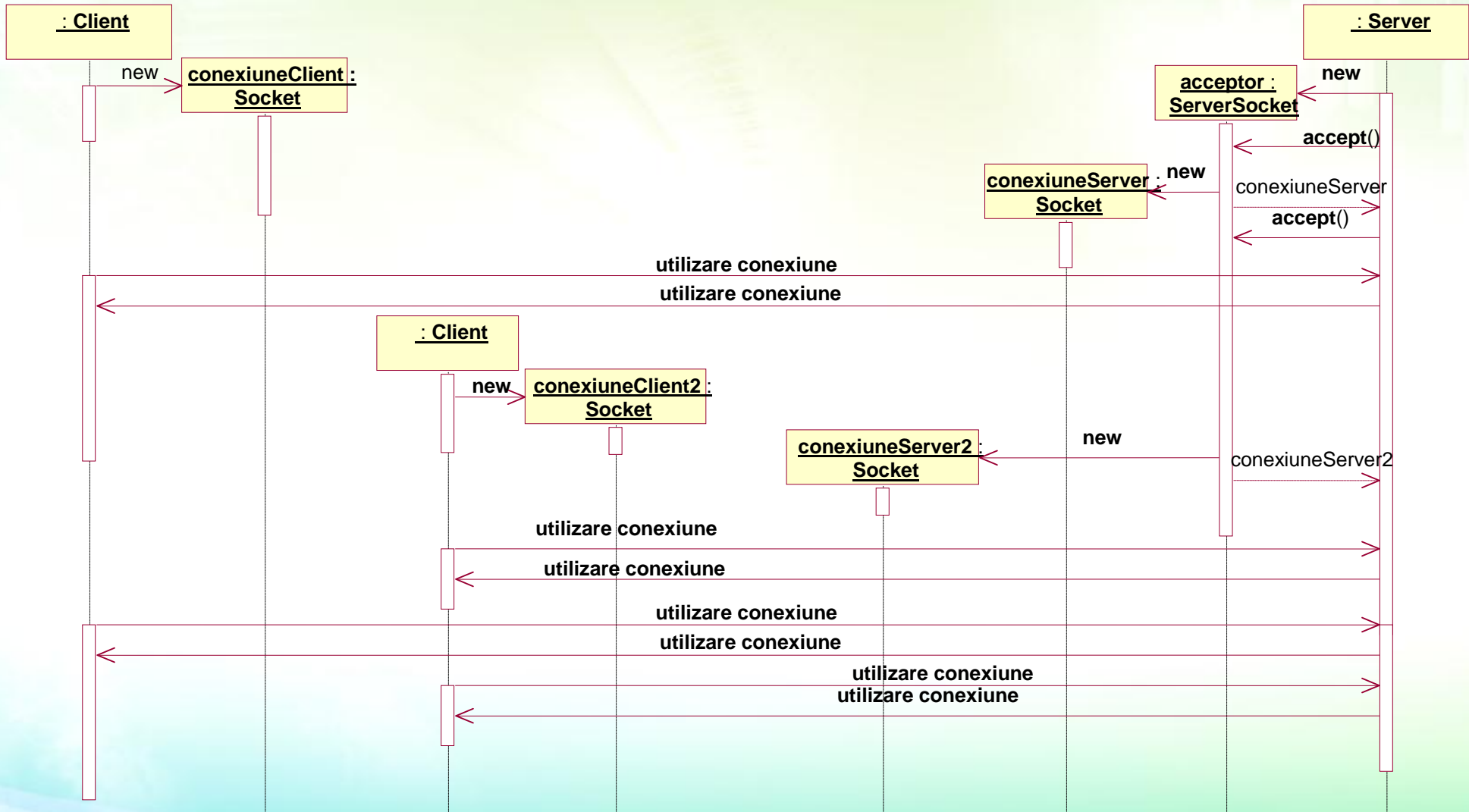
0 Unu		0 Unu
0 Doi		0 Doi
1 Doi	↔	1 Unu
1 Unu		1 Doi
2 Doi		2 Unu
2 Unu		3 Unu
3 Doi		2 Doi
4 Doi		4 Unu
3 Unu		3 Doi
4 Unu		Gata! Unu
Gata! Unu		4 Doi
Gata! Doi		Gata! Doi



Servere multifilare bazate pe socketuri TCP



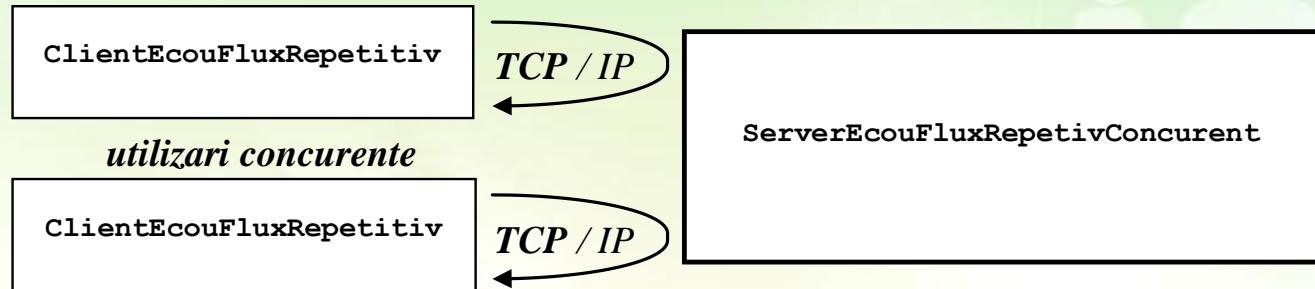
Server TCP ecou care poate trata mai multi clienti concurrent



Socketuri flux (TCP) Java



**Server TCP ecou
 care poate trata
 mai multi clienti
 concurrent**



```

1 // Servirea mai multor clienti in acelasi timp (in mod concurent)
2 while (true) {
3
4 // Blocare in asteptarea cererii de conexiune - in momentul
5 // acceptarii cererii se creaza socketul care serveste conexiunea
6 Socket socketTCP = serverTCP.accept();
7
8 new ServerEcouFluxRepetitivConcurent(socketTCP).start(); // run()
9 }
10 }
11
12 // Fir de servire client
13 public void run() {
14 try {
15
16 // Crearea fluxurilor de caractere conectate la fluxuri de la socket
17 PrintStream outRetea = new PrintStream(conexiuneTCP.getOutputStream());
18 BufferedReader inRetea = new BufferedReader(
19 new InputStreamReader(conexiuneTCP.getInputStream()));
20

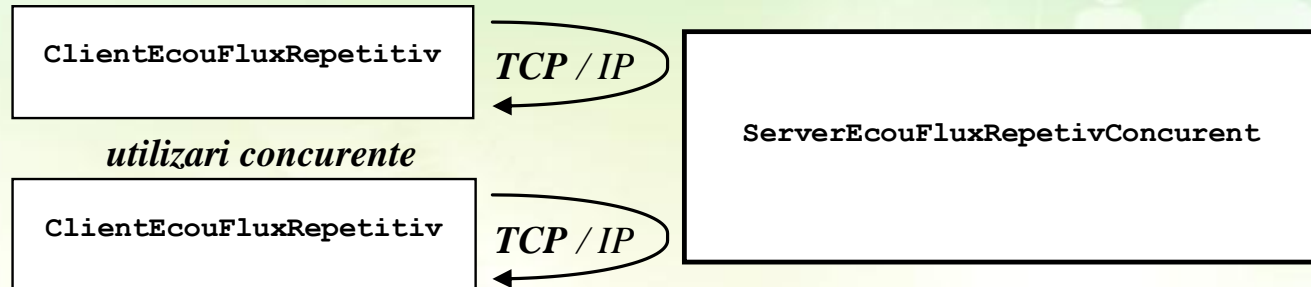
```



Socketuri flux (TCP) Java



Server TCP ecou care poate trata mai multi clienti concurrent



```

21 // Servirea clientului curent
22 while (true) {
23
24 // Citirea unei linii din fluxul de intrare TCP
25 String mesajPrimit = inRetea.readLine();
26 // Afisarea liniei citite la consola de iesire
27 System.out.println("Mesaj primit: " + mesajPrimit);
28
29 // Scrierea liniei in fluxul de iesire TCP, cu fortarea trimiterii
30 outRetea.println(mesajPrimit);
31 outRetea.flush();
32
33 if (mesajPrimit.equals(".")) break; // Testarea conditiei de oprire
34 }
35
36 // Inchiderea socketului (si implicit a fluxurilor)
37 conexiuneTCP.close();
38 System.out.println("Bye!");
39 }
40 catch (IOException ex) { System.err.println(ex); }
  
```