

2. Introducere in limbajul Java

2.3. Elementele de baza ale limbajului Java

Programul, in sensul clasic (procedural-structurat) se ocupa cu **prelucrari** asupra unor **date**.

| | | |
|---|---|--|
| 1 | <code>int suma;</code> | // <u>declaratia (tipului) variabilei</u> - cod Java |
| 2 | <code>suma = 0;</code> | // <u>initializarea variabilei</u> |
| 3 | <code>for (int i=1; i<=10; i++) {</code> | |
| 4 | <code> suma = suma + i;</code> | // <u>utilizarea variabilei</u> (citire+scriere valoare) |
| 5 | <code>}</code> | |

Datele sunt reprezentate ca **variabile** (locatii de memorie cu nume). **O variabila are:**

- **numele ei**, care o identifica si este un *alias* pentru adresa numerica (de exemplu, **suma**)
- **valoarea** continuta (de exemplu, **suma** contine pe rand valorile: **0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55**)
- **locatia** in care e continuta valoarea (in cazul **suma**, locatia ocupa in Java 4B = 32b)
- **adresa** numerica (inaccesibila in anumite limbaje, cum este Java)
- **tipul de date** (de exemplu, **suma** este de tip **int**)

Tipul de date este o **descriere abstracta a unui grup de entitati asemanatoare**. El specifica **structura variabilelor** si **domeniul de definitie al valorilor**. Mai exact, tipul de date specifica:

- **spatiul de memorie alocat** pentru stocarea valorii,
- **gama** valorilor posibile,
- **formatul valorilor literale**/de tip imediat (de ex., sufixul **f** pentru valori de tip **float**),
- **conventiile privind conversiile** catre alte tipuri (**direct, implicit, prin extindere** sau **explicit, prin cast, prin trunchiere**),
- **valorile implicite** (daca este cazul),
- **operatorii asociati (permisi)** – tin de partea de **prelucrare** asupra datelor.

Tipurile de date primitive Java:

| Categorie | Tip | Valoare implicita | Spatiu memorie | Gama valori | Conversii explicite (cast, trunchiere) | Conversii implicite (extindere) |
|----------------------------------|---------|---------------------|---------------------------|--|--|------------------------------------|
| Valori intregi cu semn | byte | 0 | 8 biti (1B) | -128 ... 127 | La char | La short, int, long, float, double |
| | short | 0 | 16 biti (2B) | -32768 ... 32767 | La byte, char | La int, long, float, double |
| | int | 0 | 32 biti (4B) | -2147483648 ... 2147483647 | La byte, short, char | La long, float, double |
| | long | 0l sau 0L | 64 biti (8B) | -9223372036854775808 ... 9223372036854775807 | La byte, short, int, char | La float, double |
| Valori in virgula mobilă cu semn | float | 0.0f sau 0.0F | 32 biti (4B) | +/-1.4E-45 ... +/-3.4028235E+38, +/-infinity, +/-0, NaN | La byte, short, int, long, char | La double |
| | double | 0.0 = 0.0d sau 0.0D | 64 biti (8B) | +/-4.9E-324 ... +/-1.7976931348623157E+308, +/-infinity, +/-0, NaN | La byte, short, int, long, float, char | Nu exista (nu sunt necesare) |
| Caractere codificate UNICODE | char | \u0000 (null) | 16 biti (2B) | \u0000 ... \uFFFF | La byte, short | La int, long, float, double |
| Valori logice | boolean | false | 1 bit folosit din 32 biti | true, false | Nu exista (nu sunt posibile) | Nu exista (nu sunt posibile) |

Tipurile de date **referinta Java**, ca si pointerii din C, C++, etc., permit definirea unor variabile care contin adrese ale unor locatii. Spre deosebire insa de cazul pointerilor, **in Java nu se poate accesa valoarea adresei continuta** in variabila de tip referinta.

Conversia explicita este necesara in cazul in care poate aparea o **trunchiere (reducere de precizie) a valorii:**

- de la **valori intregi care ocupa mai mult spatiu** la **valori intregi care ocupa mai putin spatiu** (ex: int -> byte)
- de la **valori cu virgula** la **valori intregi**
- de la **double** la **float**
- de la **valori intregi** (care pot fi si pozitive si negative) la **valori char** (care pot fi doar pozitive)
- de la **valori char** la **valori intregi byte sau short**

Structuri de control al programului: - decizie:

```
<expresieBooleana> ? <expresie1> : <expresie2>
```

echivalenta cu:

```
if (<expresieBooleana>)
    <expresie1>          // executata daca <expresieBooleana> == true
else
    <expresie2>          // executata daca <expresieBooleana> == false
```

Expresia din paranteza trebuie sa fie logica, sa fie evaluata la o valoare de tip boolean, nu poate fi de tip intreg, ca in C, C++, etc.

- iteratii (bucle):

```
for (<initializare>; <expresieBooleana>; <actualizare>)
    <instructiuneExecutataRepetat>          // repetare cat timp <expresieBooleana> == true
```

```
while (<expresieBooleana>)                // repetare cat timp <expresieBooleana> == true
    <instructiuneExecutataRepetat>
```

```
do {
    <instructiuneExecutataRepetat>
} while (<expresieBooleana>);            // repetare cat timp <expresieBooleana> == true
```

- break si continue:

```
While (boolean expression) {
    statement1
    statement2
    if(boolean expression)
        break;
    statement3
}
statement4
```

```
While (boolean expression) {
    statement1
    statement2
    if(boolean expression)
        continue;
    statement3
}
statement4
```

- break si continue cu eticheta (nu exista goto eticheta ca in C, C++, etc.):

```
1   outsideLoop: for( ... ) {
2       ...
3       while( ... ) {
4           ...
5           if ( ... ) {
6               ...
7               break outsideLoop;
8           } // end if
9
10          if ( ... ) {
11              ...
12              continue outsideLoop;
13          } // end if
14          ...
15      } // end while
16      ...
17  } // end for
```

Functii - necesitatea existentei:

- tot codul intr-o metoda (se observa redundanta):

```
1  public class Raport01 {
2      public static void main(String[] args) {
3          final int LATIME = 50;          // variabila finala (constanta!!)
4
5          for (int i = 1; i <= LATIME; i++) System.out.print('-');
6          System.out.println();          // „traseaza o linie” de 50 de caractere
7          System.out.println("Prima parte a raportului");
8          for (int i = 1; i <= LATIME; i++) System.out.print('-');
9          System.out.println();          // „traseaza o linie” de 50 de caractere
10         System.out.println("A doua parte a raportului");
11         for (int i = 1; i <= LATIME; i++) System.out.print('-');
12         System.out.println();          // „traseaza o linie” de 50 de caractere
13     }
14 }
```

- delegarea catre o metoda de tip static (pentru eliminarea redundanțelor si modularizarea sarcinilor):

```

1 public class Raport02 {
2     private static void linie() { // definitia metodei
3         final int LATIME = 50;
4         for (int i = 1; i <= LATIME; i++) System.out.print('-');
5         System.out.println(); // „traseaza o linie” de 50 de caractere
6     }
7     public static void main(String[] args) {
8         linie(); // apelul metodei
9         System.out.println("Prima parte a raportului");
10        linie(); // apelul metodei
11        System.out.println("A doua parte a raportului");
12        linie(); // apelul metodei
13    }
14 }

```

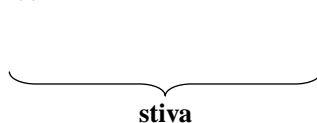
- utilizare parametri / primire argumente (pt. genericitatea codului si flexibilitatea utilizarii):

```

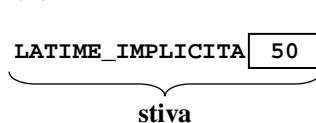
1 public class Raport03 {
2     private static void linie(int latime) { // definitia metodei
3         for (int i = 1; i <= latime; i++) System.out.print('-');
4         System.out.println(); // „traseaza o linie” de numar variabil de caractere
5     }
6     public static void main(String[] args) {
7         final int LATIME_IMPLICITA = 50;
8         linie(LATIME_IMPLICITA); // apelul metodei
9
10        System.out.println("Prima parte a raportului");
11        linie(LATIME_IMPLICITA - 5); // apelul metodei
12
13        System.out.println("A doua parte a raportului");
14        linie(LATIME_IMPLICITA); // apelul metodei
15    }
16 }

```

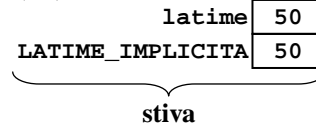
(I) Inaintea liniei 6



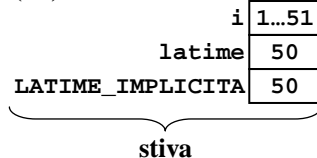
(II) Inaintea liniei 8



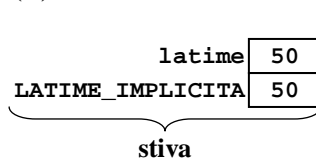
(III) Inaintea liniei 3



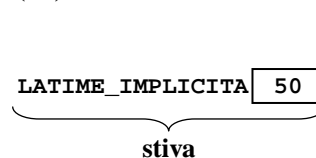
(IV) Inaintea liniei 4



(V) Inaintea liniei 5



(VI) Inaintea liniei 9

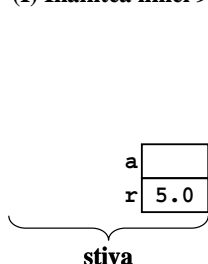
**Functii - returnarea unor valori:**

```

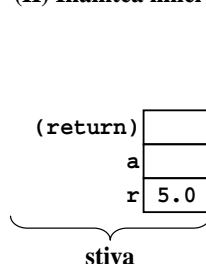
1 public class Cerc {
2     private static double arie(double raza) { // definitia metodei
3         final double PI = 3.14159; // variabila finala (constanta!!)
4         return 3.14159 * raza * raza; // returnarea unei valori
5     }
6     public static void main(String[] args) {
7         double r = 5.0; // variabila locala r
8         double a; // variabila locala a
9         a = arie(r); // apelul metodei
10        System.out.println("Un cerc de raza " + r + " are aria " + a + ".");
11    }
12 }

```

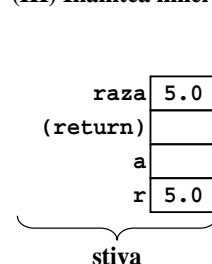
(I) Inaintea liniei 9



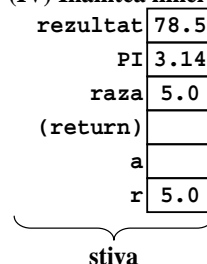
(II) Inaintea liniei 2



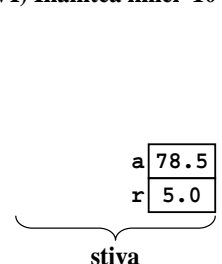
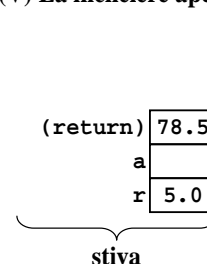
(III) Inaintea liniei 3



(IV) Inaintea liniei 5



(V) La incheiere apel (VI) Inaintea liniei 10



Functii - pasarea argumentelor prin valoare (efectul utilizarii unei copii a valorii primite)**1. Problema pasarii unei valori primitive**

```

1  public class C1 {
2      public static void inc(int i) { // declaratie (semnatura) metoda inc()
3          i++; // i este parametru formal (pe scurt, parametru)
4      }
5
6      public static void main(String[] args) {
7          int x = 10;
8          inc(x); // apel metoda inc()
9          System.out.println("x = " + x); // x este parametru actual (sau argument)
10     } // Rezultat: x = 10
11 }

```

2. Solutia pasarii unui tablou

```

1  public class C2 {
2      public static void inc(int[] i) { // primeste o copie a referintei cu aceeasi
3          // valoare, asa incat refera acelasi tablou
4          i[0]++; // este incrementat primul element al tabloului
5      }
6
7      public static void main(String[] args) {
8          int[] x = {10}; // tablou cu un element, referit de x
9          inc(x); // este pasata referinta (valoarea ei)
10         System.out.println("x[0] = " + x[0]); // Rezultat: x[0] = 11
11     }
12 }

```

3. Solutia pasarii unui obiect care contine un camp public (accesibil de catre orice cod exterior) – caz in care se poate vorbi de “lucrul cu” obiecte!

```

1  public class C3 {
2      public static void inc(ClasaInt i) { // primeste o copie a referintei cu aceeasi
3          // valoare, asa incat refera acelasi obiect
4          i.camp++; // e incrementat campul continut in obiect
5      }
6
7      public static void main(String[] args) {
8          ClasaInt x = new ClasaInt(); // obiect referit de x, continand camp tip int
9          x.camp = 10; // initializat cu valoarea 10
10         inc(x); // este pasata referinta (valoarea ei)
11         System.out.println("x.camp = " + x.camp); // Rezultat: x.camp = 11
12     }
13 }
14
15 class ClasaInt {
16     public int camp;
17 }

```

4. Solutia pasarii unui obiect care contine un camp privat (inaccesibil oricarui cod exterior) **si metode de acces** – caz in care se poate vorbi de “orientare spre” obiecte!

```

1  public class C4 {
2
3      public static void inc(ClasaInt i) { // primeste o copie a referintei cu aceeasi
4          // valoare, asa incat refera acelasi obiect
5          i.setCamp(i.getCamp()+1); // e incrementat campul incapsulat in obiect
6      }
7
8      public static void main(String[] args) {
9          ClasaInt x = new ClasaInt(); // obiect referit de x, continand camp tip int
10         x.setCamp(10); // initializat cu valoarea 10
11         inc(x); // este pasata referinta (valoarea ei)
12         System.out.println("x.getCamp() = " + x.getCamp()); // Rez.: x.getCamp() = 11
13     }
14 }
15
16 class ClasaInt {
17     private int camp;
18     public void setCamp(int c) { camp=c; }
19     public int getCamp() { return camp; }
20 }

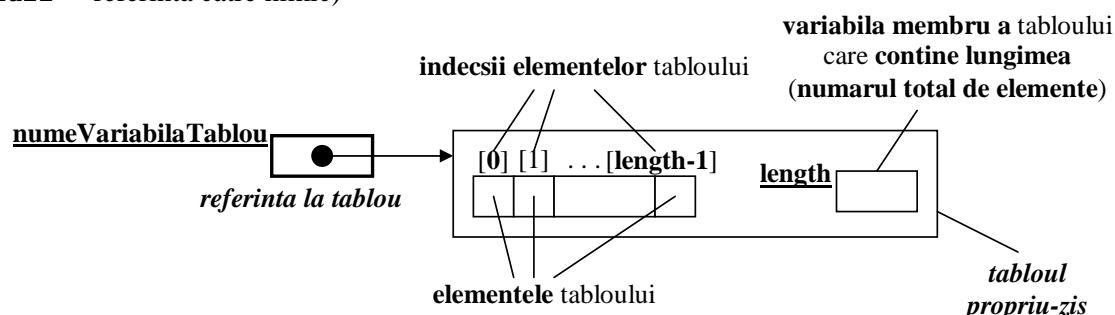
```

2.4. Tipuri referinta

Un **tablou Java** este o structura care **contine mai multe valori de acelasi tip, numite elemente**.

Lungimea unui tablou (numarul de elemente) este fixa, **stabilita in momentul crearii tabloului** (cu operatorul **new**).

Variabila tablou este o **simplic referinta la tablou, creata in momentul declararii ei** (cand e **initializata implicit** cu valoarea **null** – referinta catre nimic)



Pentru a obtine numarul de elemente ale unui tablou se foloseste:

```
// Obtinerea dimensiunii tabloului de argumente pasate de utilizator
int numarArgumentePasateDeUtilizator = args.length;
```

Pentru a se crea un tablou cu valorile 1, 2, 3 se foloseste **sintaxa simplificata**:

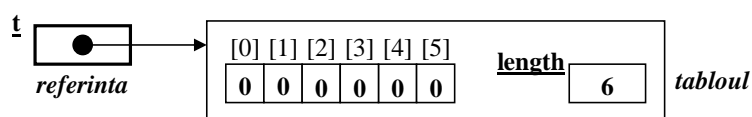
```
// Crearea unui tablou de 3 valori intregi, varianta simplificata
int[] tab = { 1, 2, 3 };
```

Acelasi efect se obtine folosind **sintaxa complexa pentru crearea unui tablou**:

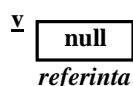
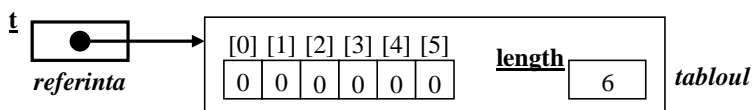
```
// Crearea unui tablou de 3 valori intregi, varianta complexa
int[] tab = new int[3]; // declararea variabilei si alocarea memoriei
tab[0]= 1; // popularea tabloului
tab[1]= 2; // popularea tabloului
tab[2]= 3; // popularea tabloului
```

```
1 int[] t; // declarare simpla
2 t = new int[6]; // alocare si initializare
3 int[] v; // declarare simpla
4 v = t; // copiere referinte
5 int[] u = { 1, 2, 3, 4 }; // declarare, alocare si initializare
6 t[1] = u[0]; // atribuire intre elemente
7 v = u; // copiere referinte
```

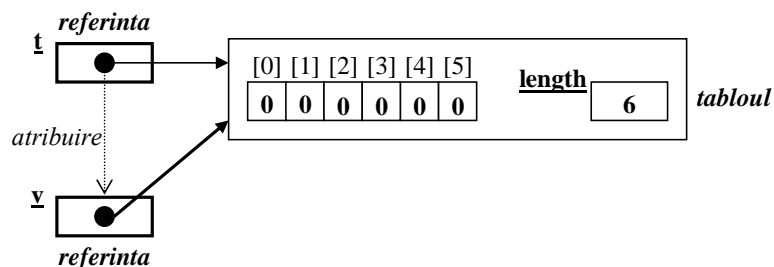
- dupa linia 2:



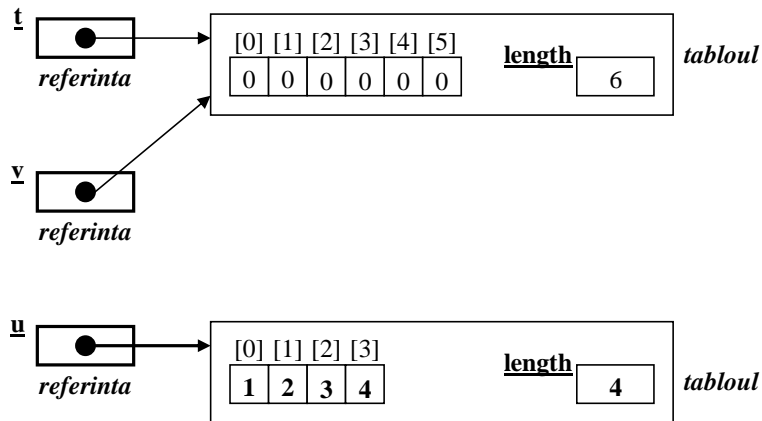
- dupa linia 3:



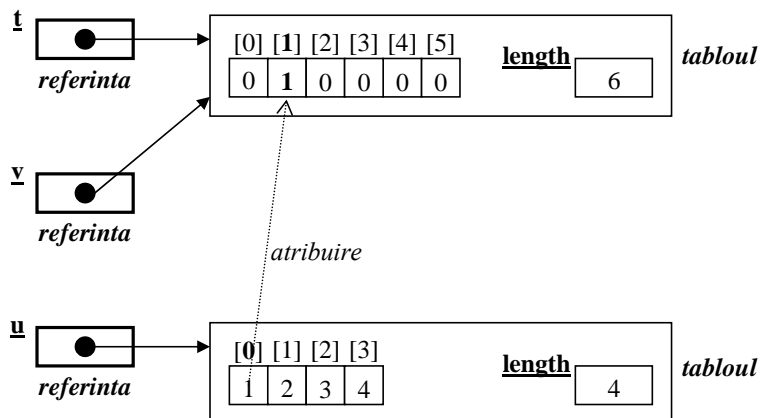
- dupa linia 4:



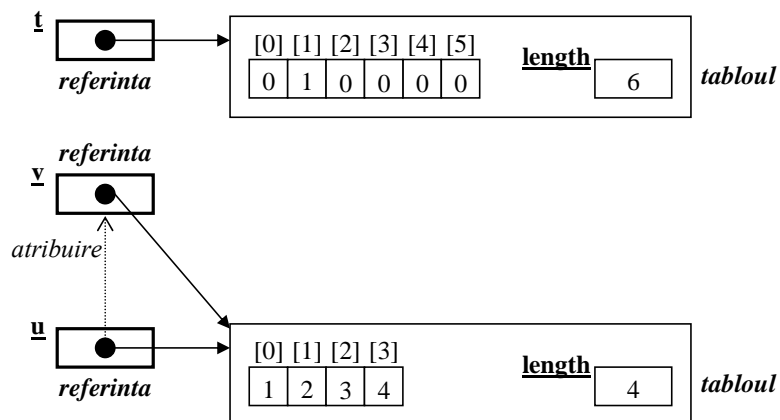
- dupa linia 5:



- dupa linia 6:



- dupa linia 7:



Lucrul cu tablouri - program histograma

```

1 public class Histograma {
2     public static void main(String[] args) {
3         int max = 3; // Valoarea maxima (valoarea minima este 0)
4         int N = 10; // Numarul valorilor de intrare (intrarilor)
5
6         int[] intrari = {1, 3, 1, 0, 3, 1, 2, 3, 2, 1}; // Tabloul intrarilor
7         System.out.print("Intrarile: "); // Afisarea tabloului
8         for (int i=0; i<N; i++) System.out.print(intrari[i] + " ");
9
10        int[] histo = new int[max+1]; // Tabloul histograma
11        for (int i=0; i<N; i++) histo[intrari[i]]++; // Popularea tabloului
12
13        System.out.println(); // Afisarea tabloului
14        for (int i=0; i<=max; i++) System.out.println(i + " apare de " + histo[i] + " ori");
15    } // Rezultatul este:
16 } // Intrarile: 1 3 1 0 3 1 2 3 2 1
17 // Valoarea 0 apare de 1 ori
18 // Valoarea 1 apare de 4 ori
19 // Valoarea 2 apare de 2 ori
20 // Valoarea 3 apare de 3 ori

```

Formatul pentru **declararea variabilelor de tip tablou cu elemente de tip referinta** este fie:

```
TipReferinta[] numeTablouElementeTipReferinta; // format Java
TipReferinta numeTablouElementeTipReferinta[]; // format C, C++
```

Formatul pentru **alocarea variabilelor de tip tablou cu elemente de tip referinta**:

```
numeTablouElementeTipReferinta = new TipReferinta[numarElementeTablou];
```

Clasa - tip (domeniu de definitie) al unor **variabile** numite **obiecte**.

- **structura complexa**, reunește **elemente de date** (campuri, atribute) **si algoritmi** (metode, operatii)
- **tip referinta Java** (obiectele sunt accesate prin **referinta**, care **contine adresa obiectului propriu-zis**)

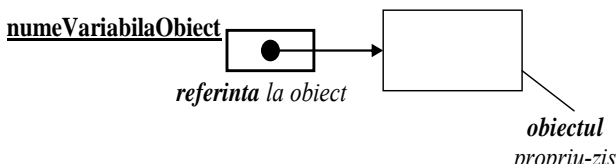
Declararea variabilelor obiect creaza o simpla referinta la obiect (implicit null)

```
NumeClasa numeVariabilaObiect;
```

numeVariabilaObiect null *referinta*

Crearea dinamica a structurii obiectului se face cu operatorul **new**:

```
numeVariabilaObiect = new NumeClasa(listaParametri);
```



Cazul clasei String care incapsuleaza siruri de caractere, din pachetul de clase implicite (java.lang)

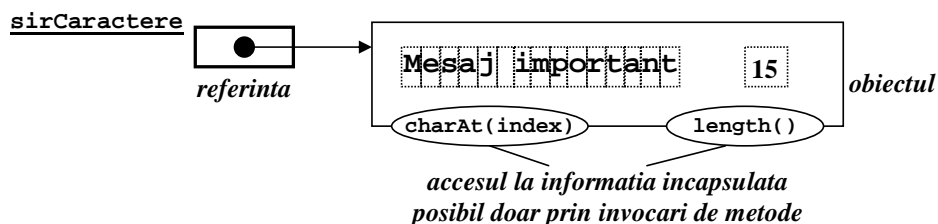
- **crearea unei referinte** la obiect de tip **string**, numita **sirCaractere**, initializata implicit cu **null**:

```
String sirCaractere;
```

sirCaractere null *referinta obiect de tip String*

- **crearea dinamica a unui obiect tip String** (obiectul incapsuleaza sirul de caractere "Mesaj important"):

```
sirCaractere = new String("Mesaj important"); // alocare si initializare
```



- **accesul la un caracter de index 0** (primul caracter):

```
sirDeCaractere.charAt(0) // prin metoda charAt()
```

- **accesul la informatia privind numarul de caractere al sirului incapsulat (lungimea sirului):**

```
sirDeCaractere.length() // prin metoda length()
```

Pentru comparatie, cazul unui tablou de caractere (in Java este diferit de un sir de caractere):

```
char[] tablouCaractere = {'M', 'e', 's', 'a', 'j', ' ', 'i', 'm', 'p', 'o', 'r', 't', 'a', 'n', 't', ','};
```

- **accesul la caracterul de index 0** (primul caracter):

```
tablouCaractere[0] // prin index si operator de indexare
```

- **accesul la informatia privind numarul de caractere (lungimea tabloului):**

```
tablouCaractere.length // prin camp length
```

2.5. Clasa Java pentru lucrul cu (siruri de) caractere

Clasa String - program de cautare a unor cuvinte cheie bazata pe parsing (analiza lexicala)

```

1  public class CautareCuvinteCheie1 {
2      public static void main(String[] args) {
3
4          String textAnalizat = "The string tokenizer class allows application " +
5              "to break a string into tokens. The tokenization method is much simpler " +
6              "than the one used by the StreamTokenizer class.";
7
8          String[] cuvinteCheie = { "string" , "token" };
9
10         // Pentru toate cuvintele cheie cautate
11         for (int i=0; i<cuvinteCheie.length; i++) {
12             String text = textAnalizat;
13             int pozitie=0;
14
15             // Daca un anumit cuvant cheie este gasit intr-un anumit text
16             // Varianta cu String.indexOf()
17             while ( text.indexOf(cuvinteCheie[i]) > -1 ) {
18                 pozitie = pozitie + text.indexOf(cuvinteCheie[i])+1;
19
20                 // Informeaza utilizatorul (indicand si pozitia)
21                 System.out.println("Cuvantul cheie \" + cuvinteCheie[i] +
22                     "\" a fost gasit in text pe pozitia " + pozitie + "\n");
23                 text = text.substring(text.indexOf(cuvinteCheie[i])+1);
24             }
25         }
26     }
27 }

```

Clasa String - program de analiza lexicala a cuvintelor si frazelor

```

1  public class StatisticiText {
2      public static void main(String[] args) {
3
4          String textAnalizat = "The string tokenizer class allows application " +
5              "to break a string into tokens. The tokenization method is much simpler " +
6              "than the one used by the StreamTokenizer class.";
7
8          String[] cuvinte = textAnalizat.split(" ");
9
10         // Numarul de cuvinte
11         System.out.println(" Textul contine " + cuvinte.length + " cuvinte:");
12
13         // Cuvintele
14         for (int i=0; i<cuvinte.length; i++) {
15             System.out.println(cuvinte[i]);
16         }
17
18         // Propozitiile
19         String propozitie = null;
20         int p = 0;
21         String text = textAnalizat;
22
23         System.out.println("\n Textul contine urmatoarele propozitii:\n");
24         for (int f = 0; (f = text.indexOf(".")) > -1; p++) {
25             propozitie = text.substring(0, f);
26             System.out.println(propozitie + ".\n");
27             text = text.substring(f+1, text.length());
28             if (text.indexOf(" ") == 0) {
29                 text = text.substring(1, text.length());
30             }
31         }
32         System.out.println("In total sunt " + p + " propozitii:\n");
33     }
34 }

```

Echivalente functionale:

```

1  char[] caractere = {'t', 'e', 's', 't'};
2  String sir = new String(caractere);
3  // echivalent cu String sir = String.valueOf(caractere);

1  char[] caractere = {'t', 'e', 's', 't', 'a', 'r', 'e'};
2  String sir = new String(caractere, 2, 5);
3  // echivalent cu String sir = String.valueOf(caractere, 2, 5);

```



```

1  String original = "sir";
2  String copie = new String(original);
3  // echivalent cu String copie = original.toString();
4  // echivalent cu String copie = String.valueOf(original);
5  // echivalent cu String copie = original.substring(0);

```

Complementaritati functionale:

```

String sir = "test";
byte[] octeti = sir.getBytes();
String copieSir = new String(octeti);

```

Exemple de lucru cu obiecte de tip String.

```

1  // variabile referinta
2  String a; // referinta la String initializata implicit cu null
3  String b = null; // referinta la String initializata explicit cu null
4
5  // constructie siruri de caractere utilizand constructori String()
6  String sirVid = new String(); // sirVid.length = 0, sirVid = ""
7
8  byte[] tabByte = {65, 110, 110, 97}; // coduri ASCII
9  String sirTablouByte = new String(tabByte); // sirTablouByte = "Anna"
10
11 char[] tabChar = {'T', 'e', 's', 't'};
12 String sirTabChar = new String(tabChar); // sirTabChar = "Test"
13
14 String s = "Sir de caractere";
15 String sir = new String(s); // sir = "Sir de caractere"
16
17 // constructie siruri de caractere utilizand metode de clasa
18 boolean adevarat = true;
19 String sirBoolean = String.valueOf(adevarat); // sirBoolean = "true"
20
21 char caracter = 'x';
22 String sirChar = String.valueOf(caracter); // sirChar = "x"
23
24 char[] tab2Char = {'A', 'l', 't', ' ', 't', 'e', 's', 't'};
25 String sirTab2Char = String.valueOf(tab2Char); // sirTabChar2="Alt test"
26
27 int numar = 10000;
28 String sirInt = String.valueOf(numar); // sirInt = "1000"
29
30 double altNumar = 2.3;
31 String sirDouble = String.valueOf(altNumar); // sirDouble = "2.3"

```

Codul:

```
x = "a" + 4 + "c";
```

este compilat ca:

```
x = new StringBuffer().append("a").append(4).append("c").toString();
```

Utilizarea metodei insert():

```

StringBuffer sb = new StringBuffer("Drink Java!");
sb.insert(6, "Hot ");
System.out.println(sb.toString());

```

Rezultatul executiei programului:

```
Drink Hot Java!
```

Program de inversare a unui sir folosind String si StringBuffer:

```

1  class ReverseString {
2      public static String reverseIt(String source) {
3          int i, len = source.length();
4          StringBuffer dest = new StringBuffer(len);
5
6          for (i = (len - 1); i >= 0; i--)
7              dest.append(source.charAt(i));
8          return dest.toString();
9      }
10 }
11 public class StringsDemo {
12     public static void main(String[] args) {
13         String palindrome = "ele fac cafele";
14         String reversed = ReverseString.reverseIt(palindrome);
15         System.out.println(reversed);
16     }
17 }

```

Rezultatul executiei programului:

```
elefac caf ele
```

2.6. Clase predefinite pentru incapsularea tipurilor primitive. Conversii

Exemple de lucru cu obiecte de tip `Integer`.

```

1   int    i, j, k;      // intregi ca variabile de tip primitiv
2   Integer m, n, o;    // intregi incapsulati in obiecte Integer
3   String s, r, t;     // siruri de caractere (incapsulate in obiecte)
4
5   // constructia intregilor incapsulati utilizand constructori ai clasei
6   i = 1000;
7   m = new Integer(i); // echivalent cu m = new Integer(1000);
8   r = new String("30");
9   n = new Integer(r); // echivalent cu n = new Integer("30");
10
11  // constructia intregilor incapsulati utilizand metode de clasa ale
12  t = "40";
13  o = Integer.valueOf(t); // echivalent cu o = new Integer("40");
14
15  // conversia intregilor incapsulati la valori numerice primitive
16  byte iByte = m.byteValue(); // diferit de 1000! (trunchiat)
17  int  iInt = m.intValue();    // = 1000
18  float iFloat = m.floatValue(); // = 1000.0F
19  double iDouble = m.doubleValue(); // = 1000.0
20
21  // conversia valorilor intregi primitive la siruri de caractere
22  String douaSute = Integer.toString(200); // metoda de clasa (statica)
23  String oMieBinary = Integer.toBinaryString(1000); // metoda de clasa
24  String oMieHex = Integer.toHexString(1000); // metoda de clasa
25
26  // conversia sirurilor de caractere la valori intregi primitive
27  int oSuta = Integer.parseInt("100"); // metoda de clasa (statica)

```

Tratarea exceptiilor In cazul in care argumentul nu are format intreg apelul metodei `parseInt()` genereaza o exceptie de tip `NumberFormatException` (definita in pachetul `java.lang`), care trebuie tratata exceptia cu un bloc:

```

try {
    // aici este plasata secventa de cod care poate genera exceptia
}
catch (NumberFormatException ex) {
    // aici este plasata secventa de cod care trateaza exceptia
}

```

```

1   public class VerificareArgumenteIntregi {
2       public static void main(String[] args) {
3           int i;
4           for ( i=0; i < args.length; i++ ) {
5               try {
6                   System.out.println(Integer.parseInt(args[i]));
7               }
8               catch (NumberFormatException ex) {
9                   System.out.println("Argumentul " +args[i]+ " nu are format numeric intreg");
10              }
11          }
12      }
13  }

1   public class ClasificareArgumenteConsola {
2       // stabilirea la lansare a valorilor, ca argumente ale programelor
3       public static void main(String[] args) {
4           int i;
5           for ( i=0; i < args.length; i++ ) {
6               try {
7                   int intreg = Integer.parseInt(args[i]);
8                   System.out.println("Argumentul " +intreg+ " are format numeric intreg");
9               }
10              catch (NumberFormatException ex1) {
11                  try {
12                      double real = Double.parseDouble(args[i]);
13                      System.out.println("Argumentul " +real+ " are format numeric real");
14                  }
15                  catch (NumberFormatException ex2) {
16                      System.out.println("Argumentul " +args[i]+ " nu are format numeric");
17                  }
18              }
19          }
20      }
21  }

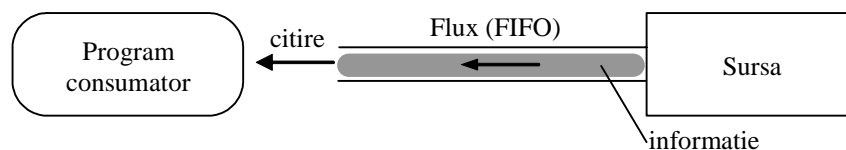
```

Programele pot avea nevoie de a:

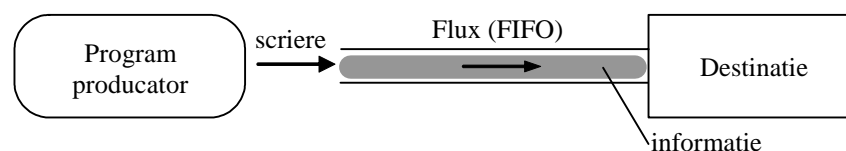
- **prelua** informatii de la **surse** externe,
- **trimit**e informatii catre **destinatii** externe.

Sursa/destinatia poate fi: *fișier pe disc, rețea (socket), memorie (program), dispozitiv IO (ecran, tastatura).*

Pentru **preluarea** informațiilor programul deschide un **flux de intrare de la o sursă** de informații și **citeste** secvențial:



Pentru **trimiterea** informației programul deschide un **flux de ieșire către o destinație** de informație și **scrie** secvențial:



In functie de **tipul de date transferate**, clasele din pachetul `java.io` se impart in **doua categorii**:

- **fluxuri de caractere** (date reprezentate in UNICODE pe 16b), avand ca radacini ale arborilor de clase derivate superclasele abstracte:

- `Reader` (de intrare) și
- `Writer` (de iesire)

- **fluxuri de octeti** (date reprezentate pe 8b), avand ca radacini ale arborilor de clase derivate superclasele abstracte:

- `InputStream` (de intrare) și
- `OutputStream` (de iesire)

In functie de **specializarea pe care o implementeaza**, subclasele claselor abstracte se impart in alte **doua categorii**:

- **fluxuri terminale** (*data sink*), care **nu** au ca sursa / destinatie alte fluxuri, ci:

- *fișierele,*
- *memoria (tablourile),*
- *rețeaua (socketurile),*
- *sirurile de caractere (String),*
- *alte programe (prin conducte - pipes)*

- **fluxuri de prelucrare** (*processing*), care **au** ca sursa / destinatie alte fluxuri, și au ca rol prelucrarea informațiilor:

- *buffer-are (stocare temporara),*
- *filtrare de diferite tipuri (conversie, contorizare, etc.)*
- *tiparire.*

Tipurile de fluxuri Java terminale:

| Tip de Terminal | Utilizare | Fluxuri de caractere | Fluxuri de octeti |
|-------------------------|--|------------------------------|---------------------------------------|
| Memorie | <i>Accesul secvențial la tablouri</i> | <code>CharArrayReader</code> | <code>ByteArrayInputStream</code> |
| | | <code>CharArrayWriter</code> | <code>ByteArrayOutputStream</code> |
| | <i>Accesul secvențial la siruri de caractere</i> | <code>StringReader</code> | <code>StringBufferInputStream</code> |
| | | <code>StringWriter</code> | <code>StringBufferOutputStream</code> |
| Canal / conducta (pipe) | <i>Conducte între programe</i> | <code>PipedReader</code> | <code>PipedInputStream</code> |
| | | <code>PipedWriter</code> | <code>PipedOutputStream</code> |
| Fisier | <i>Accesul la fișiere</i> | <code>FileReader</code> | <code>FileInputStream</code> |

1. Citirea dintr-un fișier a unui caracter prin intermediul unui flux de caractere (Unicode!):

```

1 // Crearea unui obiect referinta la fișier pe baza numelui fișierului
2 File inputFile = new File("nume1.txt");
3
4 // Crearea unui flux de intrare a caracterelor dinspre fișierul dat
5 FileReader in = new FileReader(inputFile);
6
7 // Citirea unui caracter din fișier
8 int c = in.read(); // Input
9
10 // Inchiderea fișierului
11 in.close();

```

2. Scrierea intr-un fisier a unui caracter prin intermediul unui flux de caractere:

```

1 // Crearea unui obiect referinta la fisier pe baza numelui fisierului
2 File outputFile = new File("nume2.txt");
3
4 // Crearea unui flux de iesire a caracterelor spre fisierul dat
5 FileWriter out = new FileWriter(outputFile);
6
7 char c = 'x';
8 // Scrierea unui caracter in fisier
9 out.write(c); // Output
10
11 // Inchiderea fisierului
12 out.close();

```

Tipurile de fluxuri Java de prelucrare.

| Tip de Prelucrare | Utilizare | Fluxuri de Caractere | Fluxuri de octeti |
|--------------------------|---------------------------------|----------------------|-----------------------|
| Buffer-are | Stocare temporară | BufferedReader | BufferedReader |
| | | BufferedWriter | BufferedOutputStream |
| Filtrare | Prelucrare | FilterReader | FilterInputStream |
| | | FilterWriter | FilterOutputStream |
| Conversie octet/caracter | Bridge byte-char | InputStreamReader | |
| | | OutputStreamWriter | |
| Concatenare | Prelucrare | | SequenceInputStream |
| Serializarea obiectelor | | | ObjectInputStream |
| | | | ObjectOutputStream |
| Conversia datelor | Acces la tip date primitiv Java | | DataInputStream |
| | | | DataOutputStream |
| Numararea | Numarare linii | LineNumberReader | LineNumberInputStream |
| Testare | Buffer de 1 byte/char | PushBockReader | PushbackInputStream |
| Imprimare | Tiparire | PrintWriter | PrintStream |

1. Citirea de la tastatura: pentru eficienta maxima, este recomandata inlantuirea (plasarea in cascada), astfel:

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
```

Citirea unui nume de la tastatura:

```

BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Introduceti numele: ");
String nume = in.readLine();

```

2. Afisarea argumentelor programului curent folosind un `PrintStream` (`System.out` este de tip `PrintStream`)

```

PrintStream ps = System.out;
ps.println("Argumentele programului: ");
for (int i=0; i<args.length; i++) {
    ps.print(args[i] + " ");
}
ps.println();

```

3. Citirea unui nume de la tastatura folosind inlantuirea (cascada), `DataInputStream`, `BufferedReader`:

```

DataInputStream in = new DataInputStream(new BufferedReader(System.in));
System.out.println("Introduceti numele: ");
String nume = in.readLine();

```

4. Afisarea argumentelor programului curent folosind un `DataOutputStream` in cascada cu `System.out`

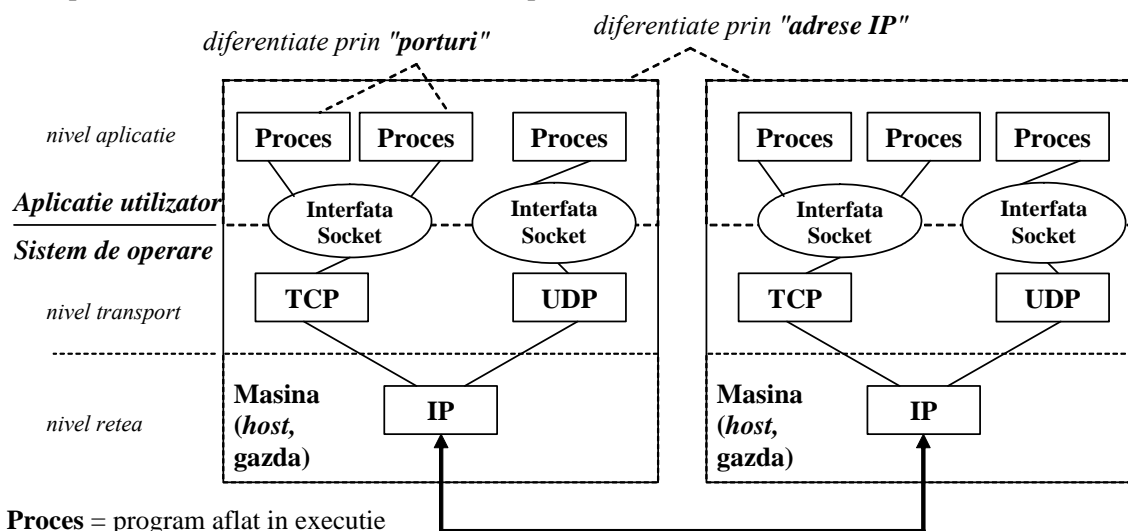
```

DataOutputStream dos = new DataOutputStream(System.out);
dos.writeBytes("Argumentele programului: \n");
for (int i=0; i<args.length; i++) {
    dos.writeBytes(args[i] + " ");
}
dos.writeChar('\n');
dos.flush();

```

Socket-ul = punct final al unei comunicatii între procese

- ofera un punct de acces la servicii de nivel transport (TCP sau UDP) în Internet

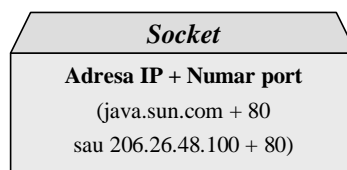


Java ofera socket-urile ca parte a unei biblioteci de clase standard, `java.net`.

Adresa socket într-o retea bazata pe IP consta din doua parti:

- **adresa IP**, pe 32 biti (4 octeti), reprezentata ca sir de 4 valori între 0 și 255 despartite prin puncte (ex. 206.26.48.100) are ca alias numele masinii și domeniului (ex. `java.sun.com`).

- **numarul de port** (identificatorul portului), pe 16 biti (2 octeti), distinct pentru fiecare tip de protocol (TCP și UDP)



Socket-ul = Adresa IP + Numarul de port

Clasa `InetAddress` încapsuleaza o adresa IP într-un obiect. Obiectul poate întoarce informatia utila dacă îi invocăm metodele. De exemplu, `equals()` întoarce adevarat dacă doua obiecte reprezinta aceeasi adresa IP.

Clasa `InetAddress` nu are constructor public. Pentru a crea obiecte ale acestei clase trebuie invocata una dintre metodele de clasa `getLocalHost()` sau `getByName()`.

Codul urmator:

```
1.a. byte[] octetiAdresaServer = { 200, 26, 48, 100 };
2.a. InetAddress adresaServer = InetAddress.getByAddress(octetiAdresaServer);
```

este echivalent cu:

```
1.b. String numeMasinaServer = "java.sun.com";
2.b. InetAddress adresaServer = InetAddress.getByName(numeMasinaServer);
```

și cu:

```
1.c. String adresaIPMasinaServer = "200.26.48.100";
2.c. InetAddress adresaServer = InetAddress.getByName(adresaIPMasinaServer);
```

Pentru a obtine obiectul `InetAddress` care încapsuleaza adresa IP locala se poate folosi:

```
InetAddress.getLocalHost()
```

O adresa IP speciala este **adresa IP loopback** (tot ce este trimis catre aceasta adresa IP se întoarce și devine intrare IP pentru gazda locala), cu ajutorul careia pot fi testate local programe care utilizeaza socket-uri.

Pentru a identifica adresa IP loopback sunt folosite numele "localhost" și valoarea numerica "127.0.0.1".

Pentru a obtine `InetAddress` care încapsuleaza adresa IP loopback pot fi folosite apelurile:

```
InetAddress.getByName(null)
InetAddress.getByName("localhost")
InetAddress.getByName("127.0.0.1")
```

Metoda `getAddress()` returneaza octetii adresei IP încapsulate, ceea ce poate fi util pentru filtrarea adreselor.

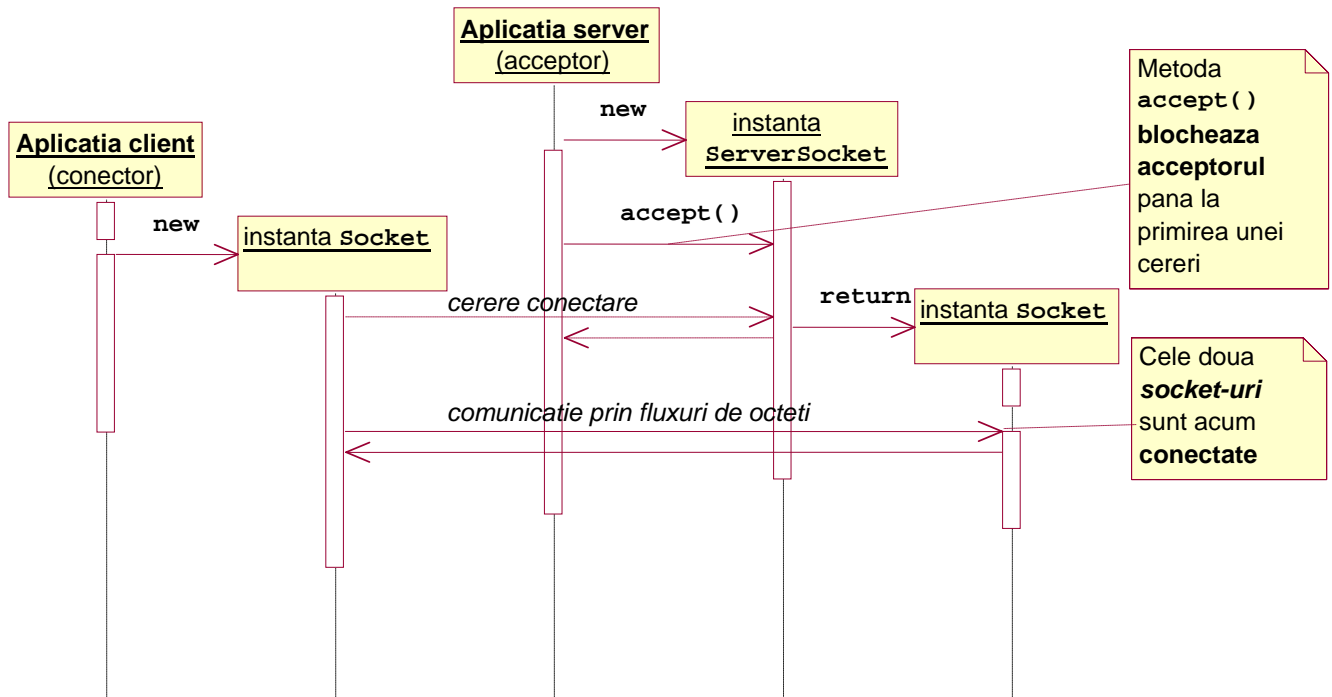
Clasa `ServerSocket` reprezinta *socket-ul* (aflat eventual pe un server bazat pe TCP) care asteapta si accepta cereri de conexiune (eventual de la un client bazat pe TCP).

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

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

Masina acceptor (de obicei serverul) 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`.

Secventa tipica a mesajelor schimbate intre cele doua masini:



Odata conexiunea stabilita, metodele `getInputStream()` si `getOutputStream()` ale clasei `Socket` trebuie utilizate pentru a obtine fluxuri de octeti, de intrare respectiv iesire, pentru comunicatia intre aplicatii.

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

```

1 // Stabilirea adresei serverului
2 String adresaServer = "localhost";
3
4 // Stabilirea portului serverului
5 int portServer = 2000;
6
7 // Crearea socketului (implicit este realizata conexiunea cu serverul)
8 Socket socketTCPClient = new Socket(adresaServer, portServer);
  
```

Dupa utilizare, `socket-ul` este inchis. Secventa tipica pentru inchiderea socket-ului:

```

1 // Inchiderea socketului (implicit a fluxurilor TCP)
2 socketTCPClient.close();
  
```

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

```

1 // Stabilirea portului serverului
2 int portServer = 2000;
3
4 // Crearea socketului server (care accepta conexiunile)
5 ServerSocket serverTCP = new ServerSocket(portServer);
  
```

Secventa tipica pentru crearea socket-ului pentru tratarea conexiunii TCP cu un client:

```

1 // Blocare in asteptarea cererii de conexiune - in momentul acceptarii
2 // cererii se creaza socketul care serveste conexiunea
3
4 Socket conexiuneTCP = serverTCP.accept();
  
```

Secventa tipica pentru crearea fluxurilor de octeti asociate socket-ului:

```

1 // Obtinerea fluxului de intrare octeti TCP
2 InputStream inTCP = socketTCPClient.getInputStream();
3
4 // Obtinerea fluxului de intrare caractere dinspre retea
5 InputStreamReader inTCPCaractere = new InputStreamReader(inTCP);
6 // Adaugarea facilitatilor de stocare temporara
7 BufferedReader inRetea = new BufferedReader(inTCPCaractere);
8
9 // Obtinerea fluxului de iesire octeti TCP
10 OutputStream outTCP = socketTCPClient.getOutputStream();
11
12 // Obtinerea fluxului de iesire spre retea,
13 // cu facilitate de afisare (similare consolei standard de iesire)
14 PrintStream outRetea = new PrintStream(outTCP);

```

Secventa tipica pentru trimiterea de date:

```

1 // Crearea unui mesaj
2 String mesajDeTrimis = "Continut mesaj";
3
4 // Scrierea catre retea (trimiterea mesajului)
5 outRetea.println(mesajDeTrimis);
6
7 // Fortarea trimiterii
8 outRetea.flush();

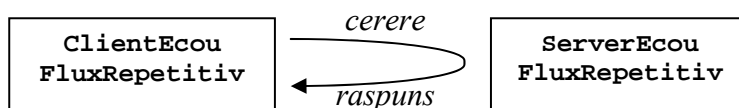
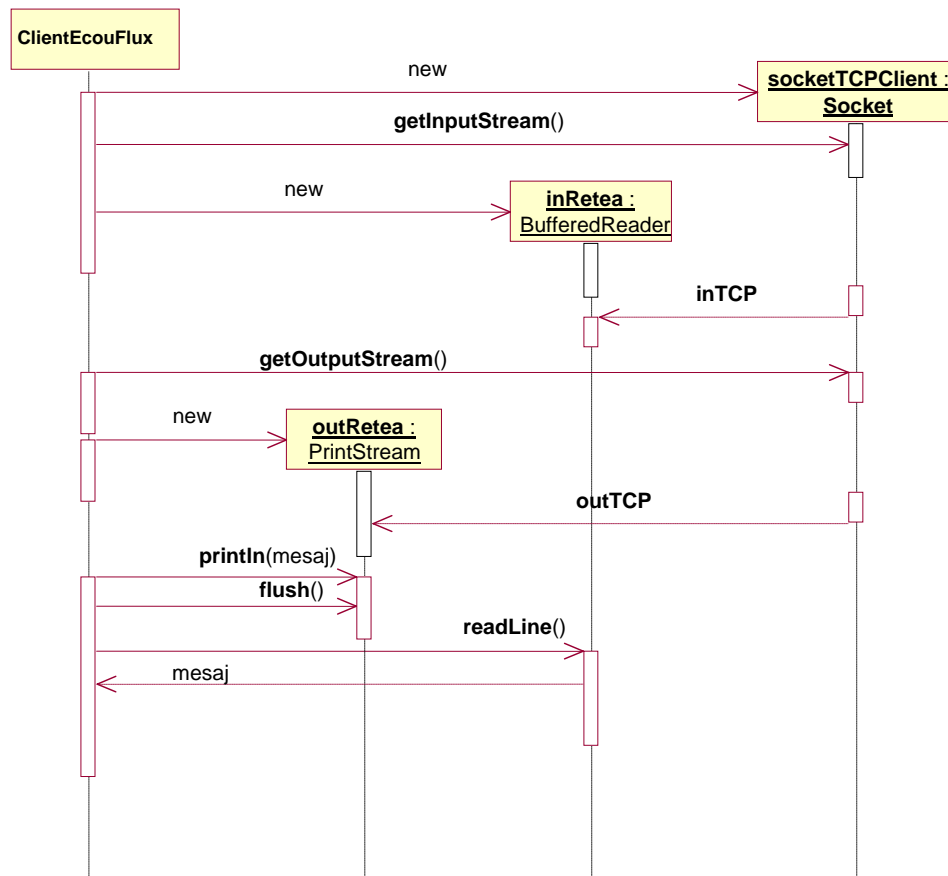
```

Secventa tipica pentru primirea de date:

```

1 // Citirea dinspre retea (receptia unui mesaj)
2 String mesajPrimit = inRetea.readLine();
3
4 // Afisarea mesajului primit
5 System.out.println(mesajPrimit);

```

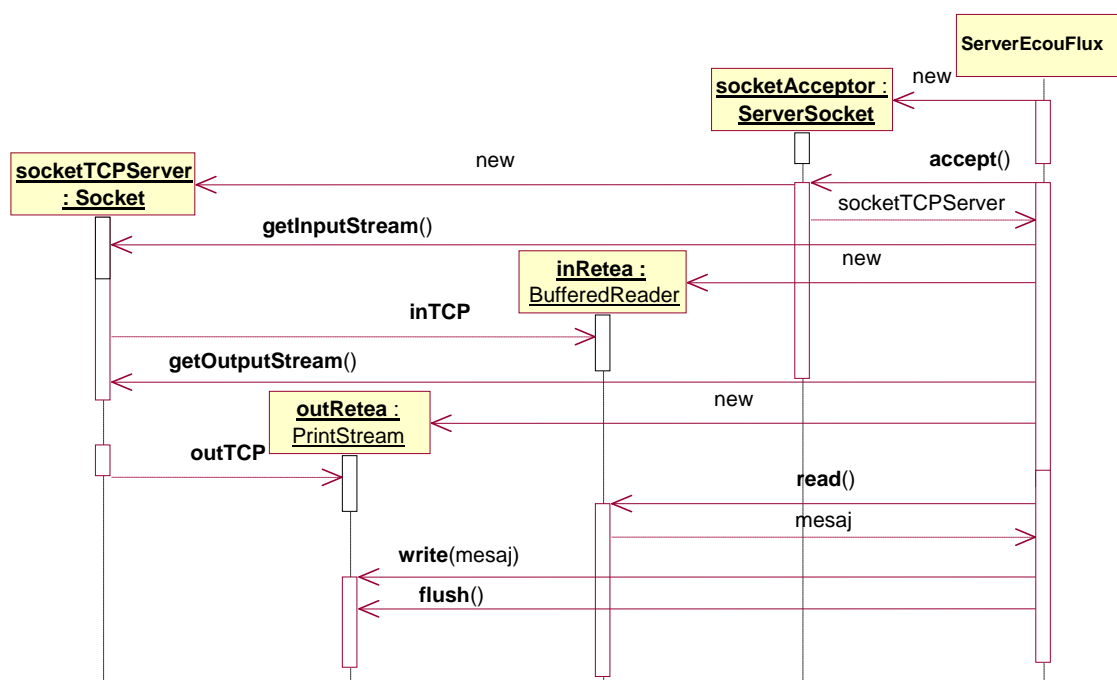
**Schimbul de mesaje la client.**

```

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
11         int portServer = Integer.parseInt(JOptionPane.showInputDialog(
12             "Introduceti numarul de port al serverului: "));
13
14         Socket conexiuneTCP = new Socket(adresaServer, portServer); // Creare socket
15
16         System.out.println("Conexiune TCP cu serverul " + adresaServer +
17             ":" + portServer + "...");
18         System.out.println("Pentru oprire introduceti '.' si <Enter>");
19
20         PrintStream outRetea = new
21             PrintStream(conexiuneTCP.getOutputStream()); // Creare fluxuri
22
23         BufferedReader inRetea = new BufferedReader(
24             new InputStreamReader(conexiuneTCP.getInputStream()));
25
26         while (true) {                    // Cat timp conditia de oprire nu este indeplinita
27
28             String mesajTrimis = JOptionPane.showInputDialog("Se trimite: "); // Mesaj
29
30             outRetea.println(mesajTrimis);           // Scrierea in fluxul de iesire TCP
31             outRetea.flush();
32
33             String mesajPrimit = inRetea.readLine(); // Citirea din fluxul de intrare TCP
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         conexiuneTCP.close(); // Inchiderea socketului (si implicit a fluxurilor)
41         System.out.println("Bye!");
42     }
43 }

```

Schimbul de mesaje la server.



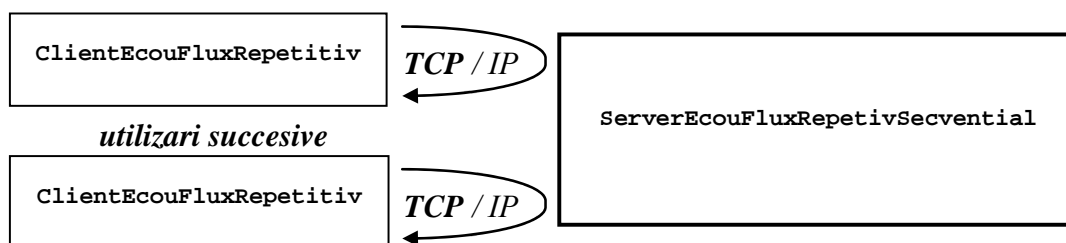
Serverul ecou TCP raspunde mai multor mesaje trimise succesiv, mesajul format dintr-un punct (".") semnaland serverului terminarea mesajelor de trimis (clientul urmand sa isi termine executia):


```

1 import java.net.*;
2 import java.io.*;
3 import javax.swing.JOptionPane;
4
5 public class ServerEcouFluxRepetitiv {           // 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 serverTCP = 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 conexiuneTCP = serverTCP.accept();
18
19         System.out.println("Conexiune TCP pe portul " + portServer + "...");
20
21         // Crearea fluxurilor de caractere conectate la fluxurile de octeti
22         // obtinute de la socketul TCP
23         PrintStream outRetea = new PrintStream(conexiuneTCP.getOutputStream());
24
25         BufferedReader inRetea = new BufferedReader(
26             new InputStreamReader(conexiuneTCP.getInputStream()));
27
28         while (true) {
29             // Citirea unei linii din fluxul de intrare TCP
30             String mesajPrimit = inRetea.readLine();
31
32             // Afisarea liniei citite la consola de iesire
33             System.out.println("Mesaj primit: " + mesajPrimit);
34
35             // Scrierea liniei in fluxul de iesire TCP, cu fortarea trimiterii
36             outRetea.println(mesajPrimit);
37             outRetea.flush();
38
39             // Testarea conditiei de oprire a servirii
40             if (mesajPrimit.equals(".")) break;
41         }
42
43         // Inchiderea socketului (si implicit a fluxurilor)
44         conexiuneTCP.close();
45         System.out.println("Bye!");
46     }
47 }

```

Serverul ecou TCP poate fi transformat astfel incat sa poata trata mai multi clienti (de exemplu, ClientEcouFluxRepetitiv) succesiv.



```

1 Import java.net.*;
2 import java.io.*;
3 import javax.swing.JOptionPane;
4
5 public class ServerEcouFluxRepetitivSecvential {
6
7     public static void main (String args[]) throws IOException {
8
9         int portServer = Integer.parseInt(JOptionPane.showInputDialog(
10             "Introduceti numarul de port al serverului: "));
11
12         // Crearea socketului server (care accepta conexiunile)
13         ServerSocket serverTCP = new ServerSocket(portServer);
14

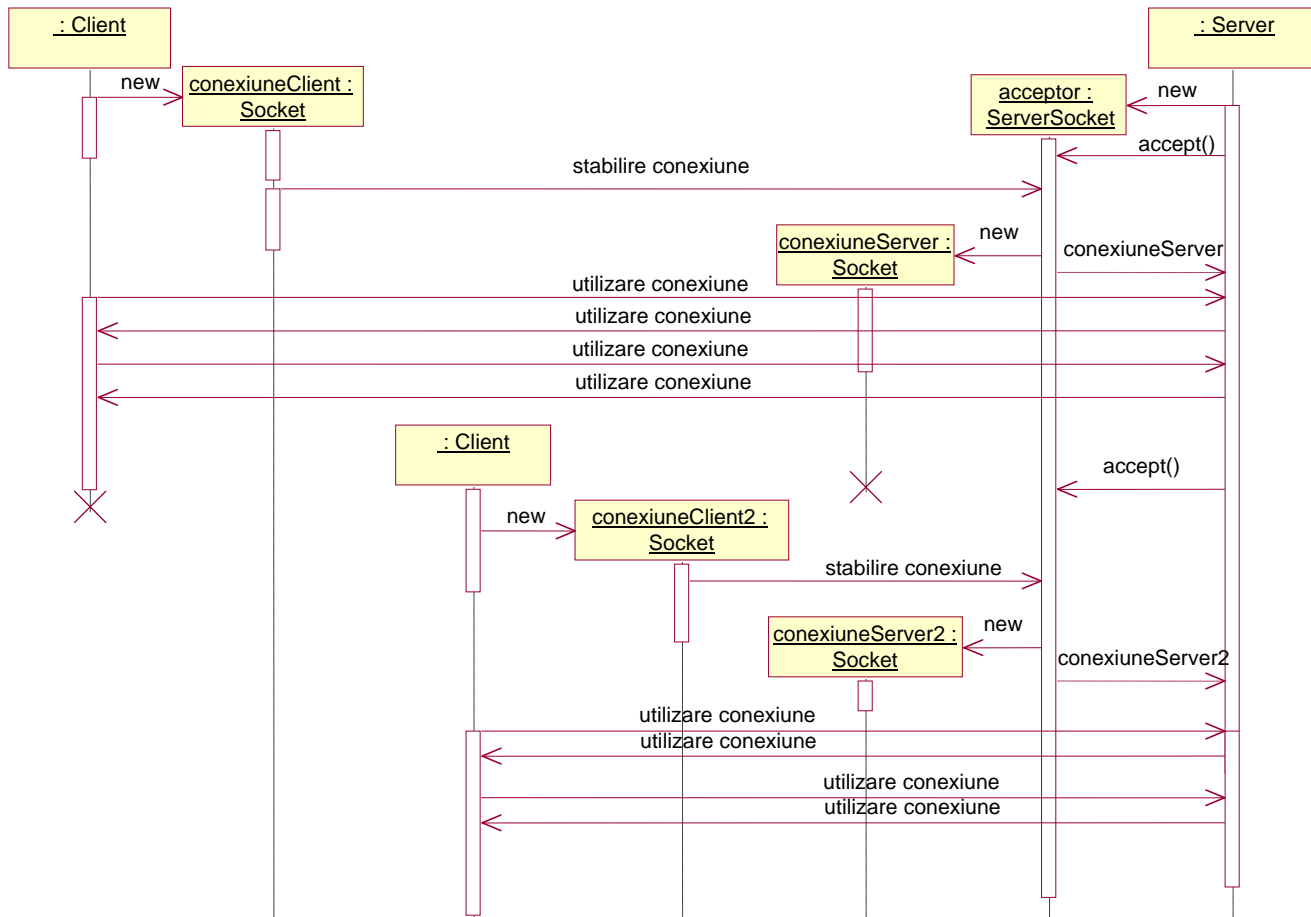
```

```

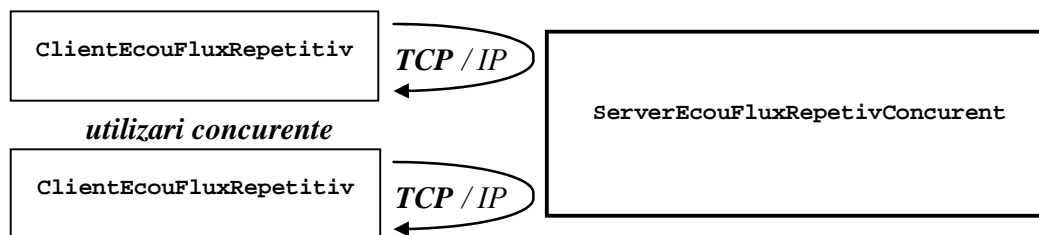
15 // Servirea mai multor clienti succesivi (in mod secvential)
16 while (true) {
17     System.out.println("Server in asteptare pe port "+portServer+"...");
18
19     // Blocare in asteptarea cererii de conexiune - in momentul
20     // acceptarii cererii se creaza socketul care serveste conexiunea
21     Socket conexiuneTCP = serverTCP.accept();
22     System.out.println("Conexiune TCP pe portul " + portServer + "...");
23
24     // Crearea fluxurilor de caractere conectate la fluxurile de octeti
25     // obtinute de la socketul TCP
26     PrintStream outRetea = new
27         PrintStream(conexiuneTCP.getOutputStream());
28     BufferedReader inRetea = new BufferedReader(
29         new InputStreamReader(conexiuneTCP.getInputStream()));
30
31     // Servirea clientului curent
32     while (true) {
33         // Citirea unei linii din fluxul de intrare TCP
34         String mesajPrimit = inRetea.readLine();
35
36         // Afisarea liniei citite la consola de iesire
37         System.out.println("Mesaj primit: " + mesajPrimit);
38
39         // Scrierea liniei in fluxul de iesire TCP, cu fortarea trimiterii
40         outRetea.println(mesajPrimit);
41         outRetea.flush();
42
43         // Testarea conditiei de oprire a servirii
44         if (mesajPrimit.equals(".")) break;
45     }
46
47     // Inchiderea socketului (si implicit a fluxurilor)
48     conexiuneTCP.close();
49     System.out.println("Bye!");
50 }
51 }
52 }

```

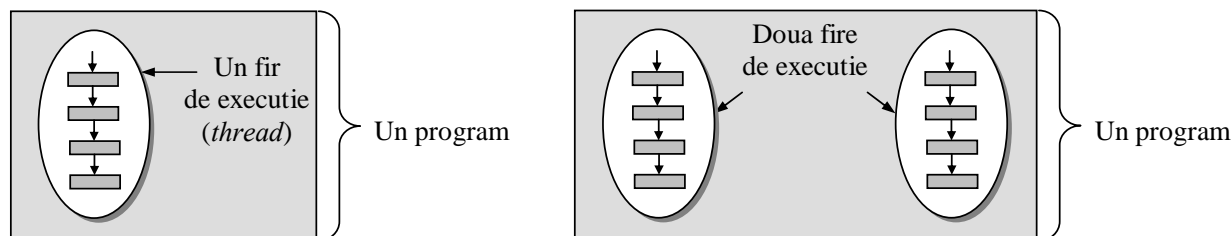
Secventa de mesaje schimbate la client, la server si intre client si server este urmatoarea:



Serverul ecou TCP poate fi transformat astfel incat sa poata trata mai multi clienti in acelasi timp (concurrent).



Fire de executie Java



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();
```

2. Ca alternativa, se poate declara o clasa care implementeaza interfata Runnable, interfata care contine doar declaratia unei metode run() (declaratie identical celei din clasa Thread, deoarece clasa Thread implementeaza interfata Runnable). **Se creaza o instanta a noii clase, este pasata constructorului la crearea unei 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 apoi a unei instante a clasei Thread:

```
FirR r = new FirR();
Thread fir = new Thread(r);
```

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

```
fir.start();
```

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 de 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 de tip Thread
// care sa refere explicit firul
// si nici variabila de tip FirR
```

Lucrul cu fire de executie - FirSimplu extinde **Thread**, metoda principala lanseaza metoda **run()** ca nou fir

```

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

```

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  }

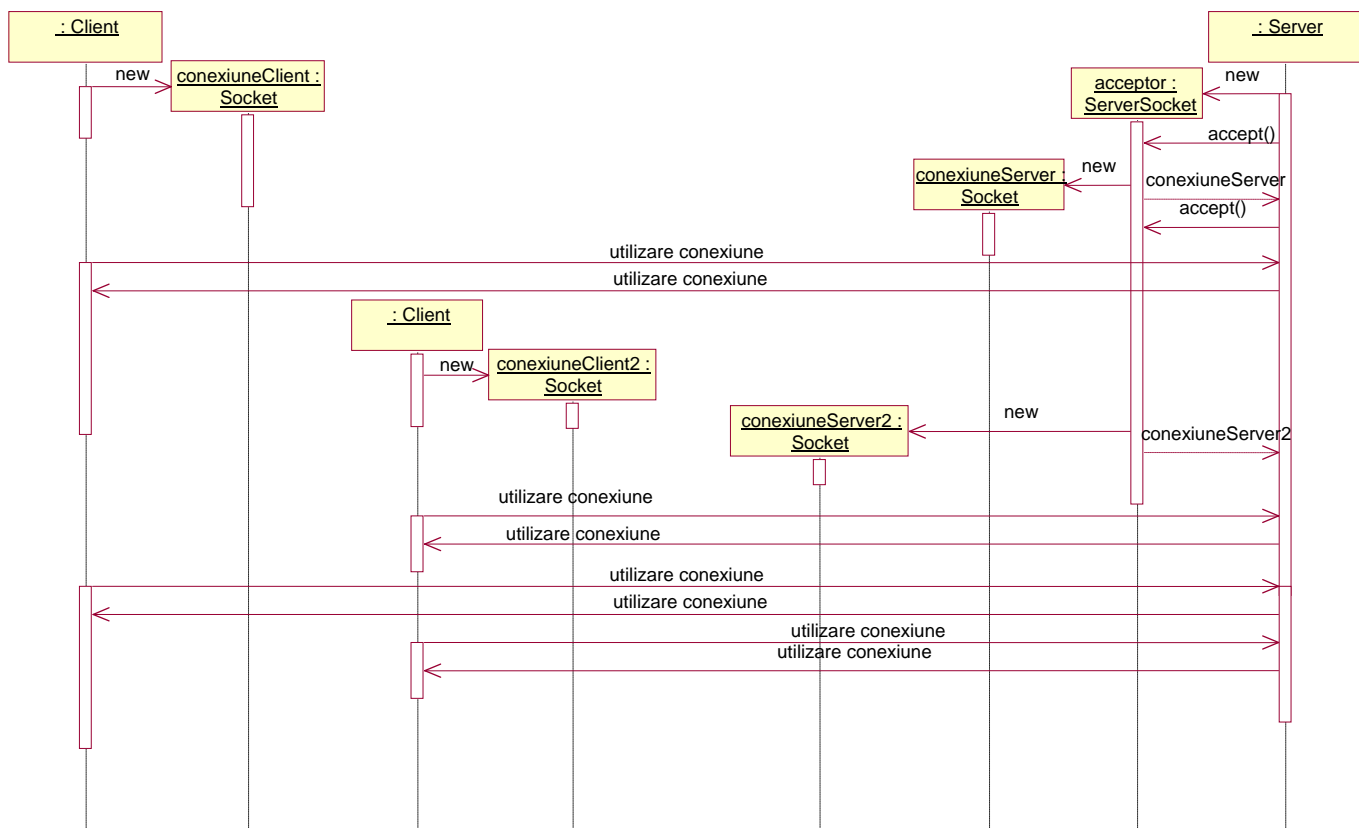
```

```

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

```

Rezultatele a doua executii succesive:



Codul necesar pentru stabilirea conexiunilor cu clientii ce urmeaza a fi tratati de obiecte fir de executie:

```

1 // initializare portServer
2
3 // Crearea socketului server (care accepta conexiunile)
4
5 ServerSocket serverTCP = new ServerSocket(portServer);
6
7 // Servirea mai multor clienti in paralel (in mod concurrent)
8
9 while (true) {
10
11 // Blocare in asteptarea cererii de conexiune
12
13 Socket conexiuneTCP = serverTCP.accept();
14
15 // Crearea si lansarea firului de executie pentru tratarea noului client
16
17 ClasaFiruluiDeTratare firExecutie = new ClasaFiruluiDeTratare(conexiuneTCP);
18 firExecutie.start();
19
20 } // Gata pentru a accepta urmatorul client

```

Codul necesar pentru comunicatia cu un client:

```

1 Socket socketTCP; // Atribut
2
3 public ClasaFiruluiDeTratare(Socket s) { // Constructor
4     socketTCP = s; // Initializarea atributului
5 }
6
7 public void run() { // Implementarea firului de executie
8
9 // Crearea fluxurilor conectate la fluxurile obtinute de la socketul TCP
10
11 PrintStream out = new PrintStream(conexiuneTCP.getOutputStream());
12 BufferedReader in = new BufferedReader(
13     new InputStreamReader(conexiuneTCP.getInputStream()));
14
15 // Tratarea clientului curent
16 while (true) {
17 // Citiri din fluxul de intrare TCP cu in.readLine();
18
19 // Scrieri in fluxul de iesire TCP cu out.println(); out.flush();
20 } // Incheierea tratarii clientului
21
22 // Inchiderea socketului
23 conexiuneTCP.close();
24
25 }

```

O varianta mai simpla poate fi utilizarea unei singure clase, care sa extinda clasa Thread, si:

- in metoda main() sa implementeze stabilirea conexiunilor cu clientii
- in metoda run() sa implementeze codul necesar pentru comunicatia cu un client.

Server ecou care poate trata mesaje sosite de la mai multi clienti in paralel:

```

1 import java.net.*;
2 import java.io.*;
3 import javax.swing.JOptionPane;
4
5 public class ServerEcouFluxRepetivConcurrent extends Thread {
6     private Socket conexiuneTCP;
7
8     public ServerEcouFluxRepetivConcurrent(Socket socketTCP) {
9         conexiuneTCP = socketTCP;
10    }
11
12    public static void main(String args[]) throws IOException {
13        int portServer = Integer.parseInt(JOptionPane.showInputDialog(
14            "Introduceti numarul de port al serverului: "));
15
16        // Crearea socketului server (care accepta conexiunile)
17        ServerSocket serverTCP = new ServerSocket(portServer);
18
19        System.out.println("Server in asteptare pe portul "+portServer+"...");
20

```

```

21 // Servirea mai multor clienti in acelasi timp (in mod concurrent)
22 while (true) {
23 // Blocare in asteptarea cererii de conexiune - in momentul
24 // acceptarii cererii se creaza socketul care serveste conexiunea
25 Socket socketTCP = serverTCP.accept();
26 System.out.println("Conexiune TCP pe portul " + portServer + "...");
27
28 new ServerEcouFluxRepetivConcurrent(socketTCP).start(); // run()
29 }
30 }
31
32 public void run() { // Fir de servire client
33 try {
34 // Crearea fluxurilor de caractere conectate la fluxurile de octeti
35 // obtinute de la socketul TCP
36 PrintStream outRetea = new
37 PrintStream(conexiuneTCP.getOutputStream());
38 BufferedReader inRetea = new BufferedReader(
39 new InputStreamReader(conexiuneTCP.getInputStream()));
40
41 // Servirea clientului curent
42 while (true) {
43 // Citirea unei linii din fluxul de intrare TCP
44 String mesajPrimit = inRetea.readLine();
45
46 // Afisarea liniei citite la consola de iesire
47 System.out.println("Mesaj primit: " + mesajPrimit);
48
49 // Scrierea liniei in fluxul de iesire TCP, cu fortarea trimiterii
50 outRetea.println(mesajPrimit);
51 outRetea.flush();
52
53 // Testarea conditiei de oprire a servirii
54 if (mesajPrimit.equals(".")) break;
55 }
56
57 // Inchiderea socketului (si implicit a fluxurilor)
58 conexiuneTCP.close();
59 System.out.println("Bye!");
60 }
61 catch (IOException ex) {
62 System.err.println(ex);
63 }
64 }
65 }

```

Program care creaza socket-uri pentru scanarea conexiunilor TCP de pe masina locala sau alte masini IP.

```

1 import java.net.*;
2 import java.io.*;
3
4 public class ScannerPorturiTCPSuperioare {
5 public static void main (String args[]) {
6     BufferedReader inConsola = new BufferedReader(new
7         InputStreamReader(System.in));
8     Socket socketTCP;
9     String adresaIP = null;
10    System.out.print("Introduceti adresa IP dorita: ");
11    try {
12        adresaIP = inConsola.readLine();
13    }
14    catch (IOException ex) {
15        System.err.println(ex);
16    }
17
18    for (int portTCP=1024; portTCP < 65536; portTCP++) {
19        try {
20            socketTCP = new Socket(adresaIP, portTCP);
21            System.out.println("Existenta un server TCP pe portul " + socketTCP.getPort()
22                + " la adresa " + socketTCP.getInetAddress().getHostAddress());
23            socketTCP.close();
24        }
25        catch (UnknownHostException ex) { System.err.println(ex); break; }
26        catch (IOException ex) { System.err.print("."); }
27    }
28 }
29 }

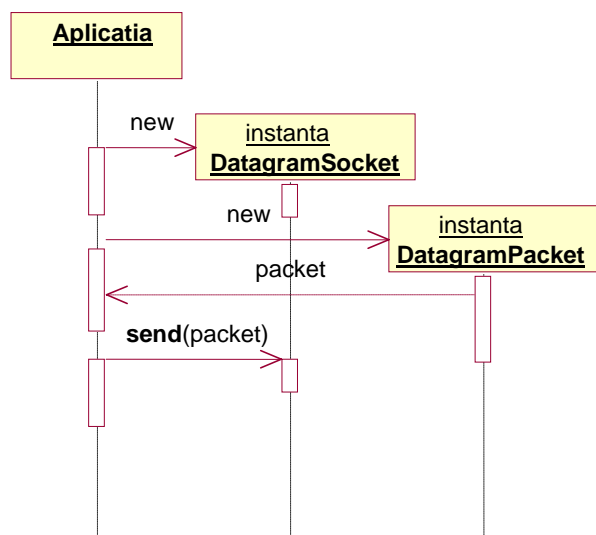
```

Java ofera, in pachetul `java.net`, mai multe **clase pentru lucrul cu socket-uri datagrame** (UDP).

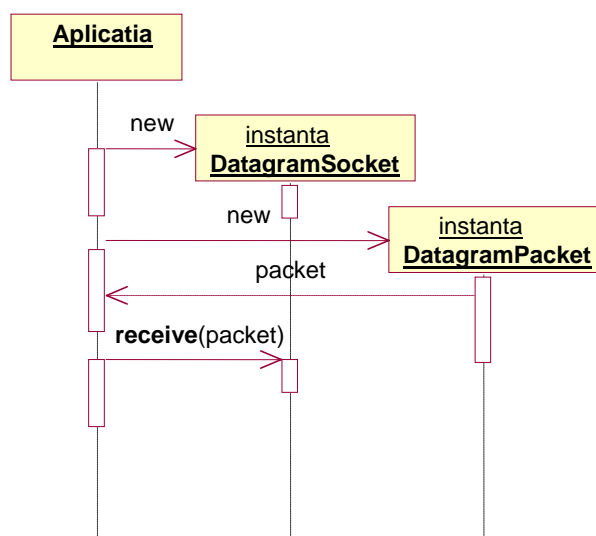
Clasa `DatagramPacket` reprezinta un **pachet UDP** (o **datagrama**). Pachetele datagrama sunt utilizate pentru livrare fara conexiune si includ in mod normal informatii privind adresele IP si porturile sursa si destinatie.

Clasa `DatagramSocket` reprezinta **socket-ul UDP**, prin care se trimite sau se primesc pachete datagrama peste retele IP prin intermediul protocolului UDP.

Un `DatagramPacket` este trimis printr-un `DatagramSocket` apeland la metoda `send()` a clasei `DatagramSocket`, pasand ca parametru respectivul `DatagramPacket`.



Un `DatagramPacket` este primit printr-un `DatagramSocket` apeland la metoda `receive()` a clasei `DatagramSocket`, pasand ca parametru un `DatagramPacket` pregatit pentru receptie.



Clasa `MulticastSocket` poate fi utilizata pentru trimiterea si receptia unui `DatagramPacket` catre, respectiv dinspre, un **grup multicast**. Clasa `MulticastSocket` este o subclasa a `DatagramSocket` care adauga functionalitati legate de multicast.

Constructorii uzuali ai clasei `DatagramPacket`:

`DatagramPacket`(byte[] buf, int length)

Construieste un obiect `DatagramPacket` pregatindu-l pentru **receptia** unui pachet de lungime `length`, furnizandu-i tabloul de octeti `buf` in care sa fie plasate datele pachetului (`length` trebuie sa fie mai mic sau egal cu `buf.length`).

`DatagramPacket`(byte[] buf, int length, InetAddress address, int port)

Construieste un obiect `DatagramPacket` pregatindu-l pentru **trimiterea** unui pachet de lungime `length` catre numarul de port UDP specificat (`port`) al gazdei cu adresa specificata (`address`), furnizandu-i tabloul de octeti `buf` din care sa fie preluate datele pachetului (`length` trebuie sa fie mai mic sau egal cu `buf.length`).

Declaratiile si descrierea catorva metode ale clasei DatagramPacket:

| | |
|-------------|---|
| InetAddress | getAddress() Returneaza adresa IP sub forma de obiect InetAddress a masinii catre care pachetul curent va fi trimis sau de la care pachetul curent va fi receptionat. |
| byte[] | getData() Returneaza tabloul de octeti (<i>bufferul</i>) care contine datele pachetului curent. |
| int | getLength() Returneaza numarul de octeti (<i>lungimea bufferului</i>) de date de trimis sau de receptionat. |
| int | getOffset() Returneaza indexul (<i>offsetul</i>) la care sunt plasate datele de trimis sau de receptionat in tabloul de octeti. |
| int | getPort() Returneaza numarul de port UDP al masinii catre care pachetul curent va fi trimis sau de la care pachetul curent va fi receptionat. |
| void | setAddress(InetAddress iaddr) Stabileste adresa IP sub forma de obiect InetAddress a masinii catre care pachetul curent va fi trimis. |
| void | setData(byte[] buf) Stabileste tabloul de octeti (<i>bufferul</i>) care contine datele pachetului curent (implicit indexul <code>offset=0</code>). |
| void | setData(byte[] buf, int offset, int length) Stabileste tabloul de octeti (<i>bufferul</i>) care contine datele pachetului curent specificandu-i indexul <code>offset</code> de la care sa inceapa plasarea datelor pachetului in tablou. |
| void | setLength(int length) Stabileste numarul de octeti (<i>lungimea bufferului</i>) de date de trimis sau de receptionat. |
| void | setPort(int iport) Stabileste numarul de port UDP al masinii catre care pachetul curent va fi trimis. |

Secventa tipica pentru crearea unui pachet datagrama (UDP) pentru timiterere:

```

1 // Stabilirea adresei serverului
2 String adresaIP = "localhost";
3
4 // Stabilirea portului serverului
5 int portUDP = 2000;
6
7 // Crearea tabloului de octeti (bufferului) de date
8 String mesaj = "mesaj de trimis";
9 byte[] bufferDate = mesaj.getBytes();
10
11 // Crearea pachetului de trimis
12 try {
13     DatagramPacket pachetUDP = new DatagramPacket(bufferDate,
14         bufferDate.length, InetAddress.getByAddress(adresaIP), portUDP);
15 }
16 catch (UnknownHostException ex) {
17     System.err.println(ex);
18 }

```

Secventa tipica pentru crearea unui pachet datagrama (UDP) pentru receptie:

```

1 // Crearea tabloului de octeti (bufferului) de date
2 byte[] bufferDate = new byte[4096];
3
4 // Crearea pachetului de primit
5 try {
6     DatagramPacket pachetUDP = new DatagramPacket(bufferDate,
7         bufferDate.length);
8 }
9 catch (UnknownHostException ex) {
10     System.err.println(ex);
11 }

```

Secventa tipica pentru citirea din pachetul UDP a datelor primite:

```

1 // Obtinerea datelor pachetului ca tablou de octeti
2 bufferDate = pachetUDP.getData();
3
4 // Reconstructia mesajului text
5 String mesajPrimit =
6     new String(bufferDate, 0, pachetUDP.getLength());

```


Principalii constructori ai clasei DatagramSocket:

| | |
|--|---|
| DatagramSocket() | Construieste un <i>socket</i> datagrama si il leaga la un port UDP disponibil pe masina locala. |
| DatagramSocket(int port) | Construieste un <i>socket</i> datagrama si il leaga la portul UDP specificat pe masina locala. |
| DatagramSocket(int port, InetAddress laddr) | Construieste un <i>socket</i> datagrama, legat pe masina locala la portul UDP si adresa IP specificate. |

Declaratiile si descrierea catorva metode ale clasei DatagramSocket:

| | |
|-------------|--|
| InetAddress | getLocalAddress() Obtine adresa IP locala la care este legat <i>socketul</i> curent. |
| int | getLocalPort() Returneaza numarul de port local la care este legat <i>socketul</i> curent. |
| void | receive(DatagramPacket p) Receptioneaza un pachet datagrama prin <i>socketul</i> curent. |
| void | send(DatagramPacket p) Trimite un pachet datagrama prin <i>socketul</i> curent. |
| void | close() Inchide <i>socketul</i> datagrama curent. |
| void | connect(InetAddress address, int port) Conecteaza <i>socketul</i> curent la adresa IP si numarul de port specificate (acestea actioneaza ca un filtru, prin <i>socketul</i> conectat putand fi trimise sau primite pachete doar catre respectiv de la adresa IP si numarul de port specificate). Implicit <i>socketul</i> este neconectat. |
| InetAddress | getInetAddress() Returneaza adresa IP la care <i>socketul</i> curent este conectat (null daca nu este conectat). |
| int | getPort() Returneaza numarul de port UDP la care <i>socketul</i> curent este conectat (-1 daca nu este conectat). |
| void | disconnect() Deconecteaza <i>socketul</i> curent. |
| boolean | isConnected() Returneaza o valoare logica indicand daca <i>socketul</i> curent este conectat. |
| boolean | getBroadcast() Returneaza o valoare logica indicand daca SO_BROADCAST este validat. |
| void | setBroadcast(boolean on) Valideaza/invalideaza SO_BROADCAST. |
| int | getSoTimeout() Obtine valoarea optiunii SO_TIMEOUT. |
| void | setSoTimeout(int timeout) Stabileste valoarea optiunii SO_TIMEOUT la un <i>timeout</i> specificat, in milisecunde. |
| int | getReceiveBufferSize() Returneaza valoarea optiunii SO_RCVBUF pentru <i>socketul</i> curent, adica dimensiunea <i>bufferului</i> utilizat de platforma pentru pachetele primite de <i>socketul</i> curent. |
| void | setReceiveBufferSize(int size) Stabileste optiunea SO_RCVBUF pentru <i>socketul</i> curent la valoarea specificata. |
| int | getSendBufferSize() Returneaza valoarea optiunii SO_SNDBUF pentru <i>socketul</i> curent, adica dimensiunea <i>bufferului</i> utilizat de platforma pentru pachetele trimise de <i>socketul</i> curent. |
| void | setSendBufferSize(int size) Stabileste optiunea SO_SNDBUF pentru <i>socketul</i> curent la valoarea specificata. |
| int | getTrafficClass() Obtine valoarea octetului clasa de trafic (QoS) sau tip de serviciu (ToS) in antetul datagrama IP care incapsuleaza datagrama UDP trimise prin <i>socketul</i> curent. |
| void | setTrafficClass(int tc) Stabileste valoarea octetului clasa de trafic (QoS) sau tip de serviciu (ToS) in antetul datagrama IP care incapsuleaza datagrama UDP trimise prin <i>socketul</i> curent. |

Secventa tipica pentru crearea socket-ului unei aplicatii client:

```
// Crearea socketului
DatagramSocket socketUDP = new DatagramSocket();
```

Secventa tipica pentru crearea socket-ului unei aplicatii server:

```
// Stabilirea portului serverului
int portServer = 2000;

// Crearea socketului
DatagramSocket socketUDP = new DatagramSocket(portServer);
```

Socket-ul utilizat pentru trimiterea de date:

```
// Trimiterea pachetului UDP
socketUDP.send(pachetDeTrimis);
```

Socket-ul utilizat pentru primirea de date:

```
// Receptia unui pachet UDP - blocare in asteptare
socketUDP.receive(pachetPrimit);
```

Dupa utilizare, socket-ul poate fi inchis:

```
// Inchiderea socketului
socketUDP.close();
```

Identificarea porturilor UDP locale pe care sunt conexiuni prin incercarea de a crea pe ele socket-uri UDP

```
1 import java.net.*;
2 public class ScannerPorturiUDPLocale {
3     public static void main (String args[]) {
4         DatagramSocket serverUDP;
5
6         for (int portUDP=1; portUDP < 65536; portUDP++) {
7             try {
8                 // Urmatoarele linii vor genera exceptie prinsa de blocul catch
9                 // in cazul in care e deja un server pe portul portUDP
10                serverUDP = new DatagramSocket(portUDP);
11                serverUDP.close();
12            }
13            catch (SocketException ex) {
14                System.err.println("Exista un server UDP pe portul " + portUDP);
15            }
16        }
17    }
18 }
```

Program server ecou cu pierderi (10% din pachetele primite)

```
1 import java.net.*;
2 import java.io.*;
3
4 /**
5  * Server ecou care pierde in mod aleator 10% din pachete
6  */
7 public class ServerEcouCuPierderiDatagramme {
8
9     public static void main(String args[]) throws IOException {
10        BufferedReader inConsola = new BufferedReader(new
11            InputStreamReader(System.in));
12        System.out.print("Introduceti numarul de port al serverului: ");
13        int portUDP = Integer.parseInt(inConsola.readLine());
14
15        // Crearea socket-ului
16        DatagramSocket socketUDP = new DatagramSocket(portUDP);
17
18        // Tratarea clientului
19        while (true) {
20
21            // Primirea pachetului
22            byte bufferDate[] = new byte[65535];
23            DatagramPacket pachetUDP = new DatagramPacket(bufferDate,
24                bufferDate.length);
25            socketUDP.receive(pachetUDP);
26            System.out.println("S-au primit " + pachetUDP.getLength() +
27                " octeti, <<" + pachetUDP.getData() + ">>");
28
29            // Decizia asupra modului de tratare a pachetului
30
```

```

31 // Trimiterea pachetului
32 if (Math.random() < .9) {
33     System.out.println("Pachetul va fi trimis in ecou.");
34
35     DatagramPacket pachetUDPEcou = new DatagramPacket(
36         pachetUDP.getData(), pachetUDP.getLength(),
37         pachetUDP.getAddress(), pachetUDP.getPort());
38     socketUDP.send(pachetUDPEcou);
39     System.out.println("Raspuns trimis.");
40 }
41
42 // Aruncarea pachetului
43 else {
44     System.out.println("Pachetul a fost aruncat.");
45 }
46 }
47 }
48 }

```

Client pentru server de ecou cu pierderi (retransmite la intarzieri >10 secunde ale pachetelor ecou de la server)

```

1 import java.net.*;
2 import java.io.*;
3
4 public class ClientEcouCuRetransmisieDatagrame {
5     protected DatagramSocket socketUDP;
6     protected DatagramPacket pachetUDP;
7     protected Asteptare timer;
8
9     protected void trimitePachet() throws IOException {
10        socketUDP.send(pachetUDP);
11        System.out.println("Pachet trimis.");
12
13        // Crearea firului timer Asteptare
14        timer = new Asteptare(10, this);
15        // Pornirea firului timer Asteptare
16        timer.start();
17    }
18
19    protected void primestePachet() throws IOException {
20        byte bufferDate[] = new byte[65535];
21        DatagramPacket pachetUDP = new DatagramPacket(bufferDate,
22            bufferDate.length);
23
24        socketUDP.receive(pachetUDP);
25        ByteArrayInputStream inTablou = new ByteArrayInputStream(
26            pachetUDP.getData(), 0, pachetUDP.getLength());
27        DataInputStream inDateFormatate = new DataInputStream(inTablou);
28        String textPrimit = inDateFormatate.readUTF();
29        System.out.println("S-a primit " + textPrimit);
30
31        // Oprirea firului timer
32        timer.stop();
33    }
34
35    public static void main(String args[]) throws IOException {
36        BufferedReader inConsola = new BufferedReader(new
37            InputStreamReader(System.in));
38        System.out.print("Introduceti adresa IP a serverului: ");
39        String adresaServer = inConsola.readLine();
40        System.out.print("Introduceti numarul de port al serverului: ");
41        int portServer = Integer.parseInt(inConsola.readLine());
42        System.out.print("Introduceti mesajul de trimis: ");
43        String textDeTrimis = inConsola.readLine();
44
45        ClientEcouCuRetransmisieDatagrame client = new
46            ClientEcouCuRetransmisieDatagrame();
47
48        // Creare socket
49        client.socketUDP = new DatagramSocket();
50
51        // Constructie pachet
52        ByteArrayOutputStream outTablou = new ByteArrayOutputStream();
53        DataOutputStream outDateFormatate = new DataOutputStream(outTablou);
54        outDateFormatate.writeUTF(textDeTrimis);
55        byte[] bufferDate = outTablou.toByteArray();
56        client.pachetUDP = new DatagramPacket(bufferDate, bufferDate.length,
57            InetAddress.getByName(adresaServer), portServer);

```

```
56
57     while (true) {
58         // Trimitere pachet
59         client.trimitePachet();
60
61         // Primire pachet
62         client.primestePachet();
63
64         System.out.println("Pauza 2 secunde...");
65         try {
66             Thread.sleep(2000);
67         } catch (InterruptedException ex) {
68             ex.printStackTrace();
69         }
70     }
71 }
72 }
```

Programul utilizeaza clasa Asteptare, care ilustreaza lucrul cu fire de executie.

```
1 import java.net.*;
2 import java.io.*;
3 class Asteptare extends Thread {
4     private ClientEcouCuRetransmisieDatagramme client;
5     private int durata;
6
7     public Asteptare (int durata,
8                     ClientEcouCuRetransmisieDatagramme client) {
9         this.client = client;
10        this.durata = durata;
11    }
12
13    public void run() {
14        try {
15            Thread.sleep (durata*1000);
16
17            System.out.println("Retransmisie...");
18            try {
19                // Retransmisie pachet
20                client.trimitePachet();
21            } catch (IOException ex) {
22                ex.printStackTrace();
23            }
24
25        } catch (InterruptedException ex) {
26            ex.printStackTrace ();
27        }
28    }
29 }
```

Alte aspecte ale socket-urilor datagramme (UDP):

- socket-uri datagramme conectate (care permit filtrarea pachetelor primite, si nu necesita specificarea in pachet a adresei destinatiei)
- segmentarea si reasamblarea la nivel aplicatie (pentru cazul in care mesajele de lungime mare sunt avantajate de transmisia in bucati mai mici)
- controlul traficului (tratarea QoS prin servicii diferite, inluzand marcare a pachetelor la nivel aplicatie)

Single Tier (Mainframe-based)



Two-Tier

Three-Tier (RPC based)



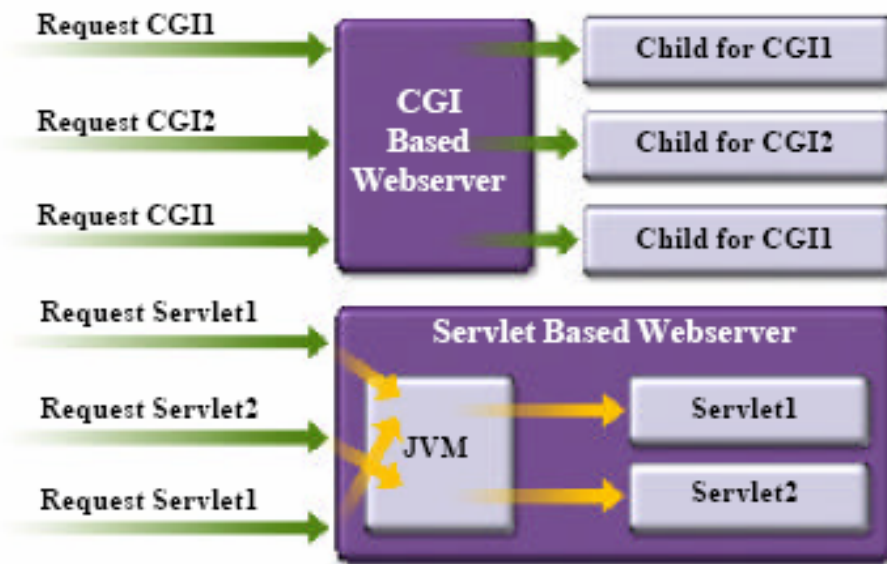
Three-Tier (Remote Object based)

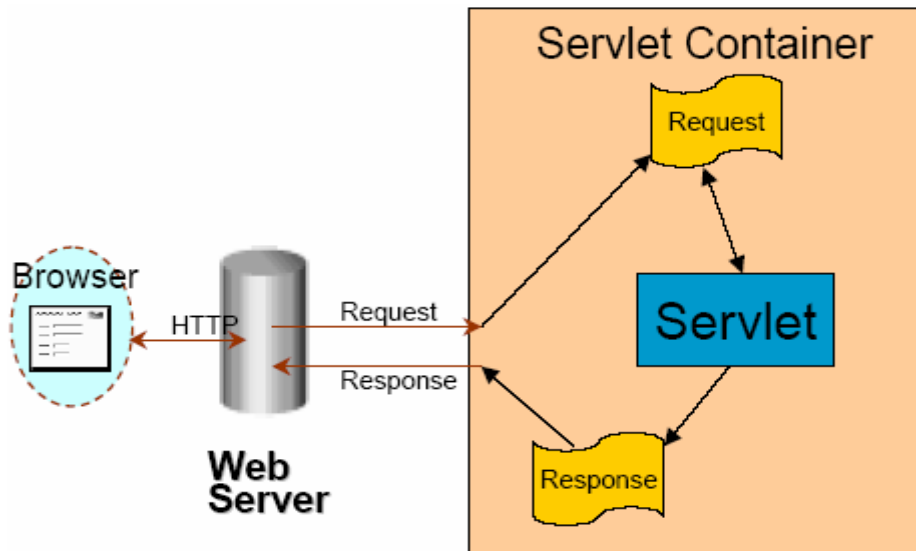


Three-Tier (Web Server)

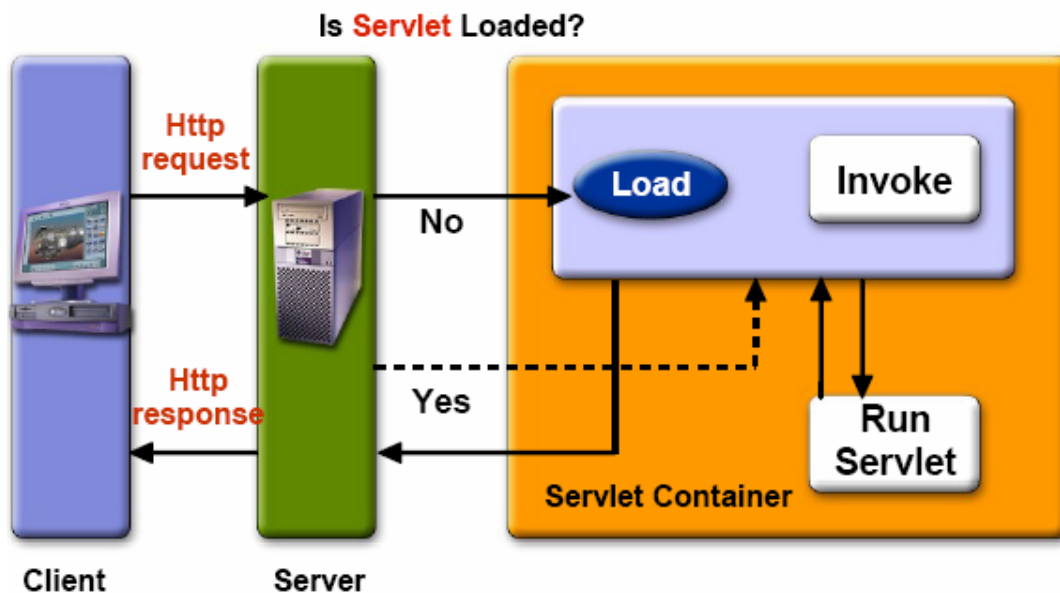


Servlet vs. CGI





Servlet Life-Cycle



doGet() and doPost() Methods

