Interfata grafica pentru un program scris in Sicstus Prolog

Intrebari: irina.ciocan@gmail.com

Cuprins

1. Introducere. De ce e utila interfata. Ce metode de realizare exista.
2. Cum facem comunicarea in prolog.
3. Cum facem comunicarea in Java
4. NetBeans IDE
5. Elemente de input pentru interfata grafica
6. Diverse
7. Intrebari/discutie

De ce avem nevoie de interfata?

- Este mai user-friendly
- Utilizatorul nu mai are nevoie sa cunoasca sintaxa Prolog si sa stie cum sa faca o interogare
- Putem oferi mai multe informatii – de exemplu, imagini, videoclipuri. Putem sa afisam datele cu text formatat (colorat, scris aldin, italic etc.)

Legatura Prolog-Java

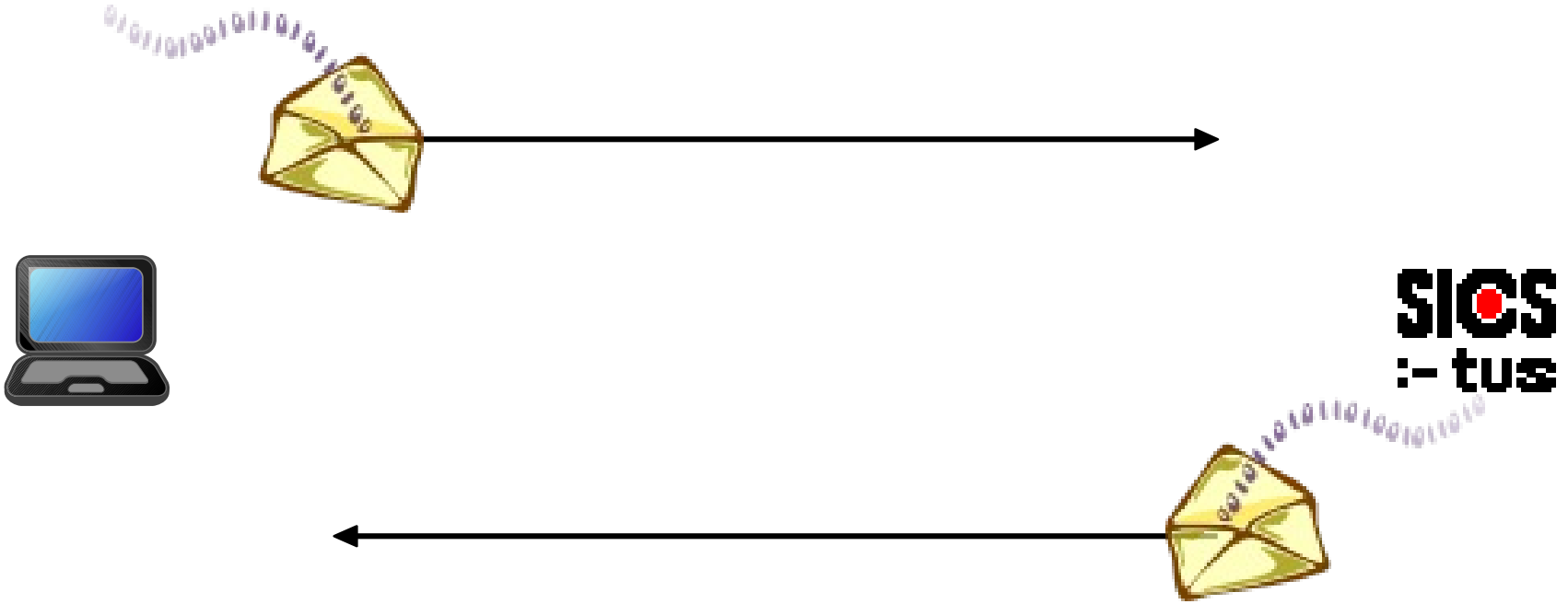
- S-a ales limbajul Java pentru realizarea interfetei in acest curs. Se mai putea realiza in C++ sau C#.
- Legatura cu interfata se poate face in doua moduri: fie prin intermediul Jasper, fie prin comunicare pe Sockets.
- Comunicarea pe Sockets prezinta si avantajul ca baza de cunostinte (fisierele prolog) pot fi pe un host dedicat, iar interfata poate rula pe alt calculator *(de exemplu, un Ionel poate, comod, de acasa, sa acceseze sistemul expert al firmei in care lucreaza, si atunci Ionel e fericit.)*

Exemplu de interfata

- Se bazeaza pe ideea prezentata in exemplul celor de la Sicstus
  https://sicstus.sics.se/~perm/sicstus/socketdemo_0.2/socketdemo.jar

- Daca vreti sa cititi si despre varianta cu jasper:
  https://sicstus.sics.se/sicstus/docs/3.7.1/html/sicstus_12.html

Cum functioneaza?



Ce este un Socket?

- Capat al comunicarii dintre 2 procese
- Adresa formata din ip si port
- Daca procesele sunt ambele pe acelasi host, ip-ul e chiar 127.0.0.1 (localhost-ul)

Sockets in Java

- In Java comunicarea se face intre doua elemente de tip Socket.
- Unul dintre socketi trebuie, insa, sa fie creat cu ajutorul unui element de tip ServerSocket (modelul client-server)
- ServerSocket-ul asteapta conexiuni
- Socket-ul initiaza o conexiune care e acceptata de ServerSocket

Java	Prolog(Sicstus 4)
Socket newSocket = serverSocket.accept();	socket_client_open(localhost: Port, Stream, [type(text)])

Rulare program

- Se ruleaza doar programul in JAVA
- Programul Prolog e apelat din interiorul programului Java printr-un exec.

Rulare Sicstus din linia de comanda

- https://sicstus.sics.se/sicstus/docs/3.7.1/html/sicstus_4.html
- **-f** *pornire rapida (fara citirea fisierului de initializare)*
- **-l fisier_prolog** *incarca acest fisier prolog direct la pornire (voi veti pune fisierul prolog cu sistemul expert)*
- **--goal scop** *(unde scop e predicatul pe care vrem sa il apelam initial)*
- **-a lista_argumente** *(alte argumente pe care prologul le poate obtine cu ajutorul predicatului* ***prolog_flag(argv, ListaArgumente))***

library(sockets)

- `:-use_module(library(sockets)).`
- Diferente destul de mari (in cadrul acestei biblioteci) intre Sicstus 3 si Sicstus 4
- Sicstus 3:

  https://sicstus.sics.se/sicstus/docs/3.7.1/html/sicstus_28.html
- Sicstus 4:
  https://sicstus.sics.se/sicstus/docs/4.3.0/html/sicstus/lib_002dsockets.html#lib_00

Un exemplu simplu - programul in Prolog

```
inceput:-prolog_flag(argv, [PortSocket|_]), %preiau numarul
portului, dat ca argument cu -a
%portul este atom, nu constanta numerica, asa ca trebuie sa il
convertim la numar
atom_chars(PortSocket,LCifre),
number_chars(Port,LCifre),%transforma lista de cifre in numarul
din
socket_client_open(localhost: Port, Stream, [type(text)]),
proceseaza_text_primit(Stream,0).
```

`prolog_flag(?Flag,?Valoare)` – in cazul de fata e folosit cu Flagul instantiat (**argv**) si obtine valoarea acestuia, sub forma unei liste de argumente (cele pasate dupa optiunea -a din linia de comanda). Noi am dat un singur argument, numarul portului, deci acesta e si primul din lista de argumente

https://sicstus.sics.se/sicstus/docs/4.0.5/html/sicstus/mpg_002dref_002dprolog_005fflag.html

Un exemplu simplu - programul in Prolog

```
atom_chars(PortSocket,LCifre), number_chars(Port,LCifre)
```

Deoarece numarul Socketului e primit ca atom si nu sub forma de constanta numerica, trebuie sa realizam o conversie. Transformam atomul cu `atom_chars` in lista de caractere(cifre), iar cu `number_chars` transformam lista de caractere mai departe in numar

```
socket_client_open(localhost: Port, Stream, [type(text)])
```

- Deschide o conexiune de tip client, care trebuie acceptata de un server. Socketul de tip server e implementat in partea de Java.
- Adresa socketului trebuie sa fie de forma host:port (unde pe post de host se da un IP, sau un nume de host – cum e localhost; iar pentru port se da un numar de port)
- Datele transmise pe stream sunt de tip text (caractere). - O alta optiune era sa fie de tip binar.

Un exemplu simplu - programul in Prolog

```
proceseaza_text_primit(Stream,C):-
read(Stream,CevaCitit),
proceseaza_termen_citit(Stream,CevaCitit,C).

proceseaza_termen_citit(Stream,salut,C):-
write(Stream,'salut, bre!\n'),
flush_output(Stream),
C1 is C+1,
proceseaza_text_primit(Stream,C1).
```

- Citeste cate un termen de pe stream si il proceseaza.
- Va reamintesc ca `read` citeste pana la caracterul punct.
- Procesarea e facuta de predicatul `proceseaza_termen_citit(+Stream,+Termen, +Contor)`
- Contorul numara al catelea termen a fost citit de pe stream. Nu este obligatoriu de pus in program. Dar poate fi util pentru debugging.
- Raspunsurile catre interfata Java sunt date cu `write(+Stream,+Termen).`

Un exemplu simplu - programul in Prolog

```
proceseaza_termen_citit(Stream,'ce mai faci?',C):-
write(Stream,'ma plictisesc...\n'),
flush_output(Stream),
C1 is C+1,
proceseaza_text_primit(Stream,C1).

proceseaza_termen_citit(Stream, X + Y,C):-
Rez is X+Y,
write(Stream,Rez),nl(Stream),
flush_output(Stream),
C1 is C+1,
proceseaza_text_primit(Stream,C1).

proceseaza_termen_citit(Stream, X, _):-
(X == end_of_file ; X == exit),
close(Stream).

proceseaza_termen_citit(Stream, Altceva,C):-
write(Stream,'nu inteleg ce vrei sa spui: '),write(Stream,Altceva),nl(Stream),
flush_output(Stream),
C1 is C+1,
proceseaza_text_primit(Stream,C1).
```

Un exemplu simplu - programul in Prolog

- Fiecare raspuns e succedat de o linie noua ('\n'). Acest lucru nu e obligatoriu, dar eu folosesc acest carcater drept separator intre mesaje.
- Fiecare tip de mesaj e procesat de cate o regula a predicatului `proceseaza_termen_citit. Exista` si o regula pentru cazul general, ca sa nu se termine cu esec predicatul la primirea unui tip de mesaj netratat:

```
proceseaza_termen_citit(Stream, Altceva,C)
```

- Este obligatoriu sa existe un `flush_output(Stream)` dupa fiecare mesaj. Fara apelarea acestui predicat, ceea ce s-a scris pe Stream poate ramane in buffer, si sa nu mai ajunga la interfata.
- Predicatul `proceseaza_termen_citit` este recursiv, si are ca pas de oprire cazul in care se citeste de pe socket termenul 'exit' sau finalul de stream(`end_of_file`).

Un exemplu simplu - programul in Java

Clase folosite:

- ExempluInterfataProlog
- ConexiuneProlog
- CititorMesaje
- ExpeditorMesaje
- Fereastra

ExempluInterfataProlog

- Este clasa pricipala (cea care contine si functia main)
- Initiaza conexiunea cu ajutorul unui element de tip ConexiuneProlog. Creaza un obiect de tip Fereastra si il deschide. Realieaza legatura intre cele doua.

ExempluInterfataProlog

```
public class ExempluInterfataProlog {
    static final int PORT=5002;
    public static void main(String[] args) {
        ConexiuneProlog cxp;
        try {
            final Fereastra fereastra=new Fereastra("Exemplu Interfata Prolog");

            cxp=new ConexiuneProlog(PORT,fereastra);

            fereastra.setConexiune(cxp);
            fereastra.setVisible(true);
            fereastra.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    try {
                        fereastra.conexiune.opresteProlog();
                        fereastra.conexiune.expeditor.gata=true;
                    } catch (InterruptedException ex) {

Logger.getLogger(ExempluInterfataProlog.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            });
      […….....mai multe catch-uri lipsa – le aveti oricum complete in fisierul
exemplu ….....]
    }

}
```

Setam valoarea portului prin care se va comunica. Aceasta valoare e mai tarziu transmisa si catre programul Prolog

ExempluInterfataProlog

```
public class ExempluInterfataProlog {
    static final int PORT=5002;
    public static void main(String[] args) {
        ConexiuneProlog cxp;
        try {
            final Fereastra fereastra=new Fereastra("Exemplu Interfata Prolog");

            cxp=new ConexiuneProlog(PORT,fereastra);

            fereastra.setConexiune(cxp);
            fereastra.setVisible(true);
            fereastra.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    try {
                        fereastra.conexiune.opresteProlog();
                        fereastra.conexiune.expeditor.gata=true;
                    } catch (InterruptedException ex) {

Logger.getLogger(ExempluInterfataProlog.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            });
      […….....mai multe catch-uri lipsa – le aveti oricum complete in fisierul
exemplu ….....]
    }

}
```

Cream obiectele de tip Conexiune si de tip Fereastra, si realizam legatura intre ele

ExempluInterfataProlog

```
public class ExempluInterfataProlog {
    static final int PORT=5002;
    public static void main(String[] args) {
        ConexiuneProlog cxp;
        try {
            final Fereastra fereastra=new Fereastra("Exemplu Interfata Prolog");

            cxp=new ConexiuneProlog(PORT,fereastra);

            fereastra.setConexiune(cxp);
            fereastra.setVisible(true);
            fereastra.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    try {
                        fereastra.conexiune.opresteProlog();
                        fereastra.conexiune.expeditor.gata=true;
                    } catch (InterruptedException ex) {

Logger.getLogger(ExempluInterfataProlog.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            });
      […….....mai multe catch-uri lipsa – le aveti oricum complete in fisierul
exemplu ….....]
    }

}
```

Deschide efectiv fereastra

ExempluInterfataProlog

```
public class ExempluInterfataProlog {
    static final int PORT=5002;
    public static void main(String[] args) {
        ConexiuneProlog cxp;
        try {
            final Fereastra fereastra=new Fereastra("Exemplu Interfata Prolog");

            cxp=new ConexiuneProlog(PORT,fereastra);

            fereastra.setConexiune(cxp);
            fereastra.setVisible(true);
            fereastra.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    try {
                        fereastra.conexiune.opresteProlog();
                        fereastra.conexiune.expeditor.gata=true;
                    } catch (InterruptedException ex) {

Logger.getLogger(ExempluInterfataProlog.class.getName()).log(Level.SEVERE, null, ex);
                    }
                }
            });
      […….....mai multe catch-uri lipsa – le aveti oricum complete in fisierul
exemplu ….....]
    }

}
```

Parte de cod care se realizeaza la inchiderea ferestrei: aici se inchide conexiunea catre prolog

ConexiuneProlog

- Este clasa care realizeaza conexiunea cu procesul prolog
- Ea porneste procesul prolog
- Foloseste clasele `CititorMesaje` si `ExpeditorMesaje` pentru transmiterea mesajelor.
- Are o referinta catre fereastra deschisa pentru a transmite date spre elementele de display din fereastra

ConexiuneProlog

```
public class ConexiuneProlog {
    final String
caleExecutabilSicstus="C:\\Users\\Irina\\Desktop\\SICStus Prolog
4.0.2\\SICStus Prolog 4.0.2\\bin\\sicstus.exe";
    final String nume_fisier="exemplu_prolog.pl";

    final String scop="inceput.";


    Process procesSicstus;
    ExpeditorMesaje expeditor;
    CititorMesaje cititor;
    Fereastra fereastra;
    int port;


    public Fereastra getFereastra(){
        return fereastra;
    }
```

Aici veti pune calea catre executabilul sicstus.exe

Sau il puteti adauga in variabila de mediu “path” si puteti folosi doar numele executabilului in loc de toata calea

ConexiuneProlog

```
public class ConexiuneProlog {
    final String
caleExecutabilSicstus="C:\\Users\\Irina\\Desktop\\SICStus Prolog
4.0.2\\SICStus Prolog 4.0.2\\bin\\sicstus.exe";
    final String nume_fisier="exemplu_prolog.pl";

    final String scop="inceput.";


    Process procesSicstus;
    ExpeditorMesaje expeditor;
    CititorMesaje cititor;
    Fereastra fereastra;
    int port;


    public Fereastra getFereastra(){
        return fereastra;
    }
```

Predicatul principal al fisierului prolog – cel care porneste comunicarea din partea cealalta

ConexiuneProlog

```
   public ConexiuneProlog(int _port, Fereastra _fereastra) throws IOException,
InterruptedException{
        InputStream processIs, processStreamErr;
        port=_port;
        fereastra=_fereastra;
        //acces la mediul curent de rulare
        ServerSocket servs=new ServerSocket(port);
        //Socket sock_s=servs.accept();
        cititor=new CititorMesaje(this,servs);
        cititor.start();
        expeditor=new ExpeditorMesaje(cititor);
        expeditor.start();
        Runtime rtime= Runtime.getRuntime();
        String comanda=caleExecutabilSicstus+" -f -l "+nume_fisier+" --goal
"+scop+" -a "+port;
        procesSicstus=rtime.exec(comanda);

        //InputStream-ul din care citim ce scrie procesul
        processIs=procesSicstus.getInputStream();
        //stream-ul de eroare
        processStreamErr=procesSicstus.getErrorStream();
    }
```

Instante pentru CititorMesaje si ExpeditorMesaje. Ambele extind Thread. Aici se pornesc si firele de executie pentru ele (start())

ConexiuneProlog

```
    public ConexiuneProlog(int _port, Fereastra _fereastra) throws IOException,
InterruptedException{
        InputStream processIs, processStreamErr;
        port=_port;
        fereastra=_fereastra;
        //acces la mediul curent de rulare
        ServerSocket servs=new ServerSocket(port);
        //Socket sock_s=servs.accept();
        cititor=new CititorMesaje(this,servs);
        cititor.start();
        expeditor=new ExpeditorMesaje(cititor);
        expeditor.start();
        Runtime rtime= Runtime.getRuntime();
        String comanda=caleExecutabilSicstus+" -f -l "+nume_fisier+" --goal
"+scop+" -a "+port;
        procesSicstus=rtime.exec(comanda);

        //InputStream-ul din care citim ce scrie procesul
        processIs=procesSicstus.getInputStream();
        //stream-ul de eroare
        processStreamErr=procesSicstus.getErrorStream();
    }
```

Pornirea procesului Prolog

Reamintim parametrii din linia de comanda

- **-f** *pornire rapida (fara citirea fisierului de initializare)*
- **-l fisier_prolog** *incarca acest fisier prolog direct la pornire (voi veti pune fisierul prolog cu sistemul expert)*
- **--goal scop** *(unde scop e predicatul pe care vrem sa il apelam initial)*
- **-a lista_argumente** *(alte argumente pe care prologul le poate obtine cu ajutorul predicatului* ***prolog_flag(argv, ListaArgumente))***

CititorMesaje

- Extinde clasa Thread (va rula in paralel cu programul principal)
- Foloseste Pipe-uri pentru comunicare (PipedOutputStream si **PipedInputStream**)
- Transmite mesajele catre Prolog pe stream-ul socketului

CititorMesaje

```
public class CititorMesaje extends Thread {
    ServerSocket servs;
    volatile Socket s=null;//volatile ca sa fie protejat la
accesul concurent al mai multor threaduri
    volatile PipedInputStream pis=null;
    ConexiuneProlog conexiune;

    //setteri sincronizati
    public synchronized void setSocket(Socket _s){
        s=_s;
        notify();
    }

    public final synchronized void
setPipedInputStream(PipedInputStream _pis){
        pis=_pis;
        notify();
    }
```

CititorMesaje

```
//getteri sincronizati
    public synchronized Socket getSocket() throws InterruptedException
    {
        if (s==null){
            wait();//asteapta pana este setat un socket
        }
        return s;
    }

    public synchronized PipedInputStream getPipedInputStream() throws
InterruptedException{
        if(pis==null){
            wait();
        }
        return pis;
    }

    //constructor
    public CititorMesaje(ConexiuneProlog _conexiune, ServerSocket _servs) throws
IOException{
        servs=_servs;
        conexiune=_conexiune;
    }
```

CititorMesaje

```
  public void run(){
        try {
            //apel blocant, asteapta conexiunea
            //conexiunea clinetului se face din prolog
            Socket s_aux=servs.accept();
            setSocket(s_aux);
            //pregatesc InputStream-ul pentru a citi de pe
Socket
            InputStream is=s_aux.getInputStream();

            PipedOutputStream pos=new PipedOutputStream();
            setPipedInputStream(new PipedInputStream(pos));//leg
un pipedInputStream de capatul in care se scrie
```

CititorMesaje

```
            int chr;
            String str="";
            while((chr=is.read())!=-1) {//pana nu citeste EOF
                pos.write(chr);//pun date in Pipe, primite de la Prolog
                str+=(char)chr;
                if(chr=='\n'){//linie noua (\n)=caracter separator intre
mesaje
                    final String sirDeScris=str;
                    str="";
                    SwingUtilities.invokeLater(new Runnable() {
                        public void run(){

conexiune.getFereastra().getDebugTextArea().append(sirDeScris);
                        }
                    });
                }
            }
        } catch (IOException ex) {

Logger.getLogger(CititorMesaje.class.getName()).log(Level.SEVERE, null,
ex);
        }
```

ExpeditorMesaje

- Extinde clasa Thread (va rula in paralel cu programul principal)
- Foloseste Pipe-uri pentru comunicare (**PipedOutputStream** si PipedInputStream)
- Transmite mesajele catre Prolog pe stream-ul socketului

ExpeditorMesaje

```
public class ExpeditorMesaje extends Thread{
    Socket s;
    CititorMesaje cm;
    volatile PipedOutputStream pos=null;
    PipedInputStream pis;
    OutputStream ostream;
    volatile boolean gata=false;

    //setteri sincronizati
    public final synchronized void
setPipedOutputStream(PipedOutputStream _pos){
        pos=_pos;
        notify();
    }
```

ExpeditorMesaje

```
    //getteri sincronizati
    public synchronized PipedOutputStream getPipedOutputStream() throws
InterruptedException{
        if(pos==null){
            wait();
        }
        return pos;
    }

    //constructor
    public ExpeditorMesaje(CititorMesaje _cm) throws IOException{
        cm=_cm;
        pis=new PipedInputStream();
        setPipedOutputStream(new PipedOutputStream(pis));
    }

    //functie care trimite un mesaj(string) pe socket
    public void trimiteMesajSicstus(String mesaj) throws InterruptedException{
        PipedOutputStream pos= getPipedOutputStream();
        PrintStream ps=new PrintStream(pos);
        ps.println(mesaj+" .");
        ps.flush();
    }
```

ExpeditorMesaje

```
public void run(){
        try {
            s=cm.getSocket();
            ostream=s.getOutputStream();
            int chr;
            while((chr=pis.read())!=-1){
                ostream.write(chr);
            }


        } catch (IOException ex) {
            Logger.getLogger(ExpeditorMesaje.class.getName()).log(Level.SEVERE,
null, ex);
        } catch (InterruptedException ex) {
            Logger.getLogger(ExpeditorMesaje.class.getName()).log(Level.SEVERE,
null, ex);
        }

    }
}
```

Clasa Fereastra

- Extinde clasa JFrame
- Realizeaza interfata grafica efectiva
- Textbox pentru introducerea mesajului catre Prolog
- Buton de trimitere
- JTextArea in care se afiseaza raspunsul prologului
- Are o referinta catre obiectul de tip ConexiuneProlog

Fereastra

```
public javax.swing.JTextArea getDebugTextArea(){
    return textAreaDebug;
}


public void setConexiune(ConexiuneProlog _conexiune){
    conexiune=_conexiune;
}
```

Fereastra

```
    private void
jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed
        try {
            PipedOutputStream pos=
conexiune.expeditor.getPipedOutputStream();
            PrintStream ps=new PrintStream(pos);
            ps.println("salut.");
            ps.flush();
        } catch (InterruptedException ex) {

Logger.getLogger(Fereastra.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }//GEN-LAST:event_jButton1ActionPerformed
```

Fereastra

```
    private void
okButtonActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_okButtonActionPerformed
        String valoareParametru=tfParametru.getText();
        tfParametru.setText("");
        try {

conexiune.expeditor.trimiteMesajSicstus(valoareParametru);
        } catch (InterruptedException ex) {

Logger.getLogger(Fereastra.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }//GEN-LAST:event_okButtonActionPerformed
```

Idei de debugging

Ce facem cand nu functioneaza?

1) 😭
2) Verificam in Sicstus ca Programul e corect (nu avem erori de sintaxa).
3) Verificam partea de comunicarea in retea cu ajutorul Stream-urilor, folosind stdin-ul drept stream de input.

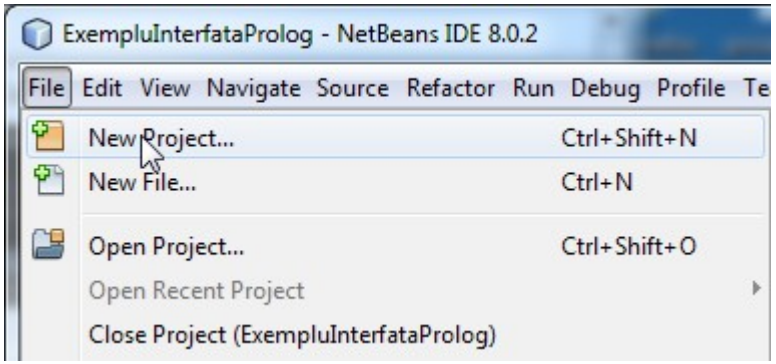
```
| ?- seeing(IC),proceseaza_text_primit(IC,0).
|: salut.
salut, bre!
|: bau.
nu inteleg ce vrei sa spui: bau
|:
```

4) Apelam manual linia de comanda care porneste Sicstusul cu fisierul *.pl* deja incarcat, ca sa vedem daca ia bine regulile si argumentele, ca sa vedem daca acestea sunt incarcate in Sicstus.

```
"[Cale executabil]sicstus.exe" -f -l exemplu_prolog.pl --goal inceput. -a 5002
```

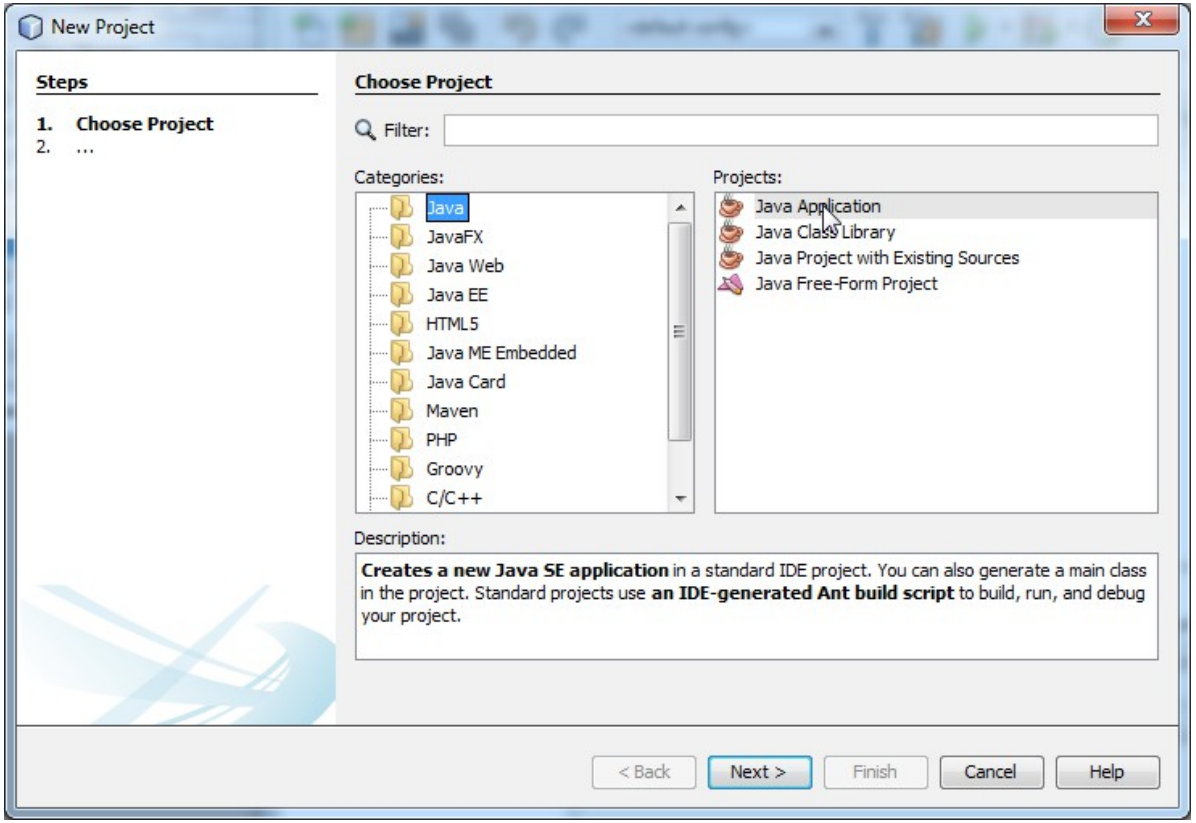
5) Verificam din Java daca are acces la fisierul pl.
6) Breakpoint-uri si watch-uri in partea de Java
7) Afisari in consola

Creare proiect NetBeans

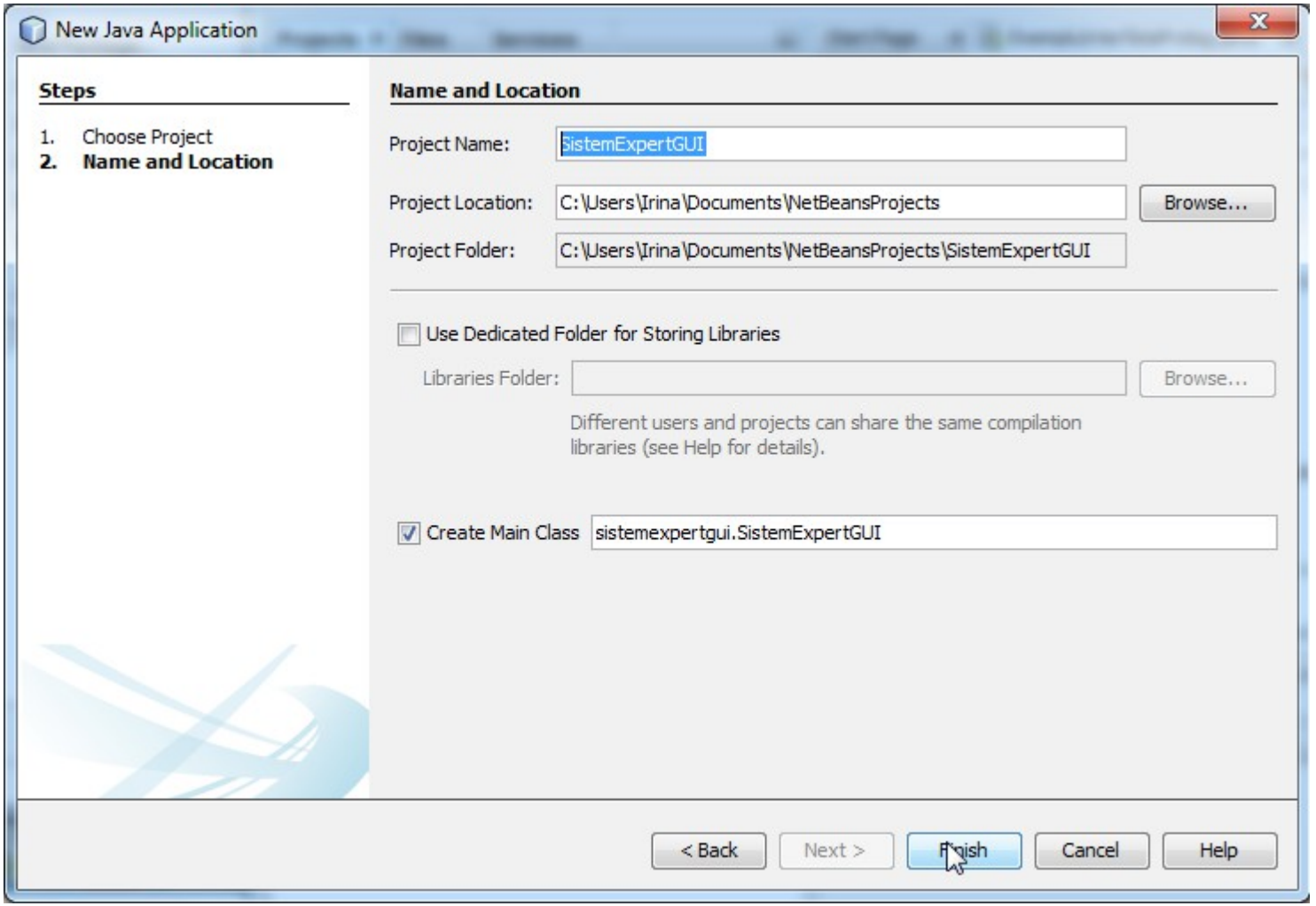


Cream un proiect nou

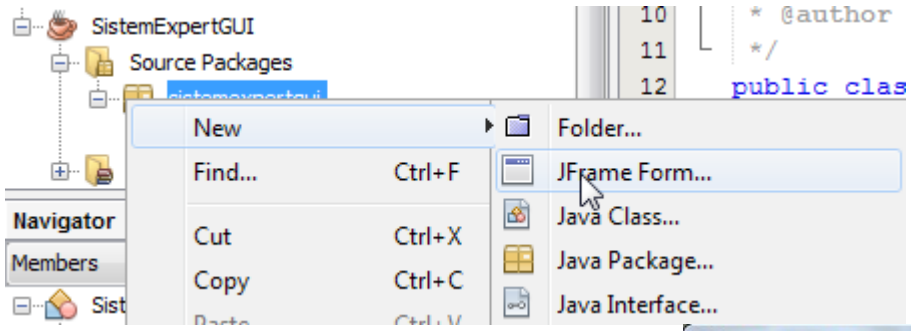
Alegem Java Application

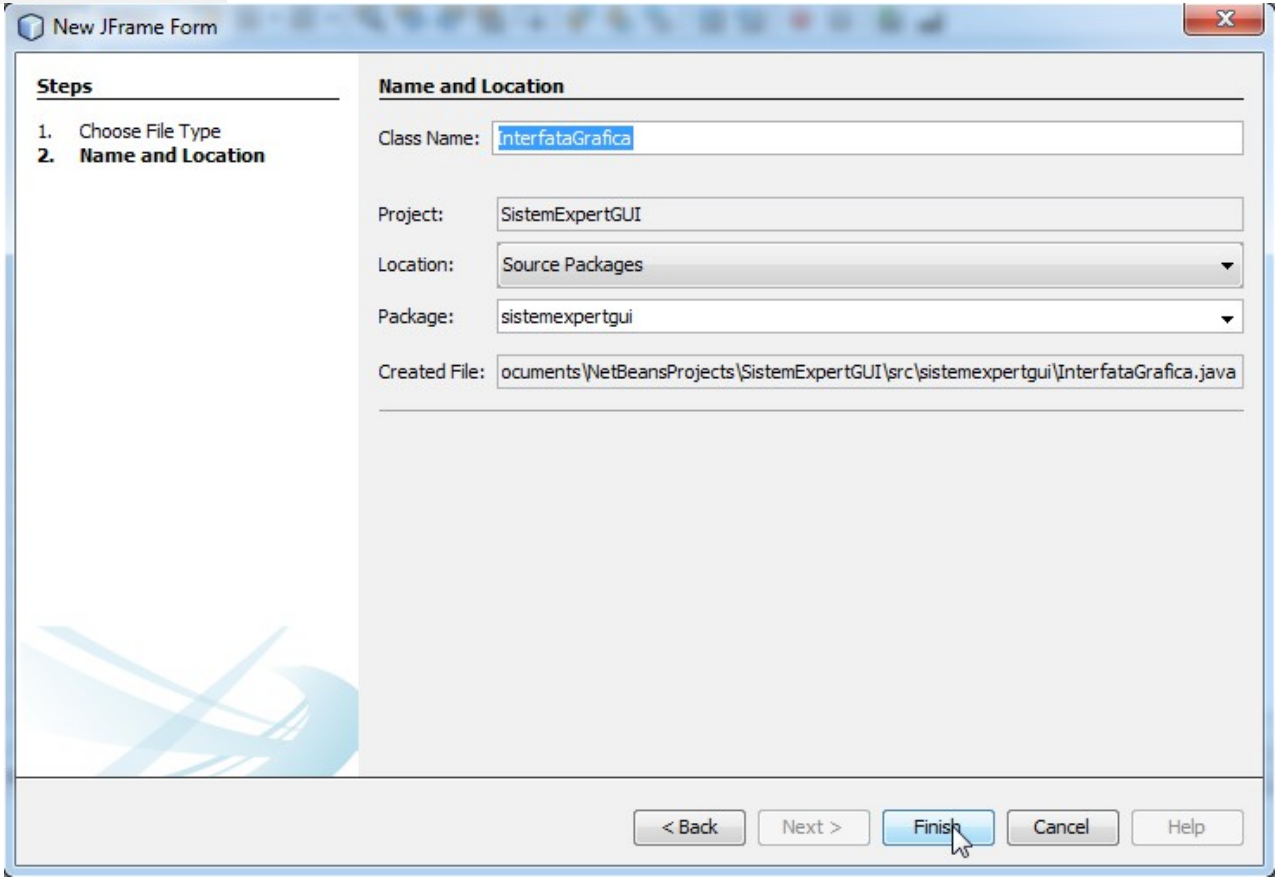


Creare proiect NetBeans

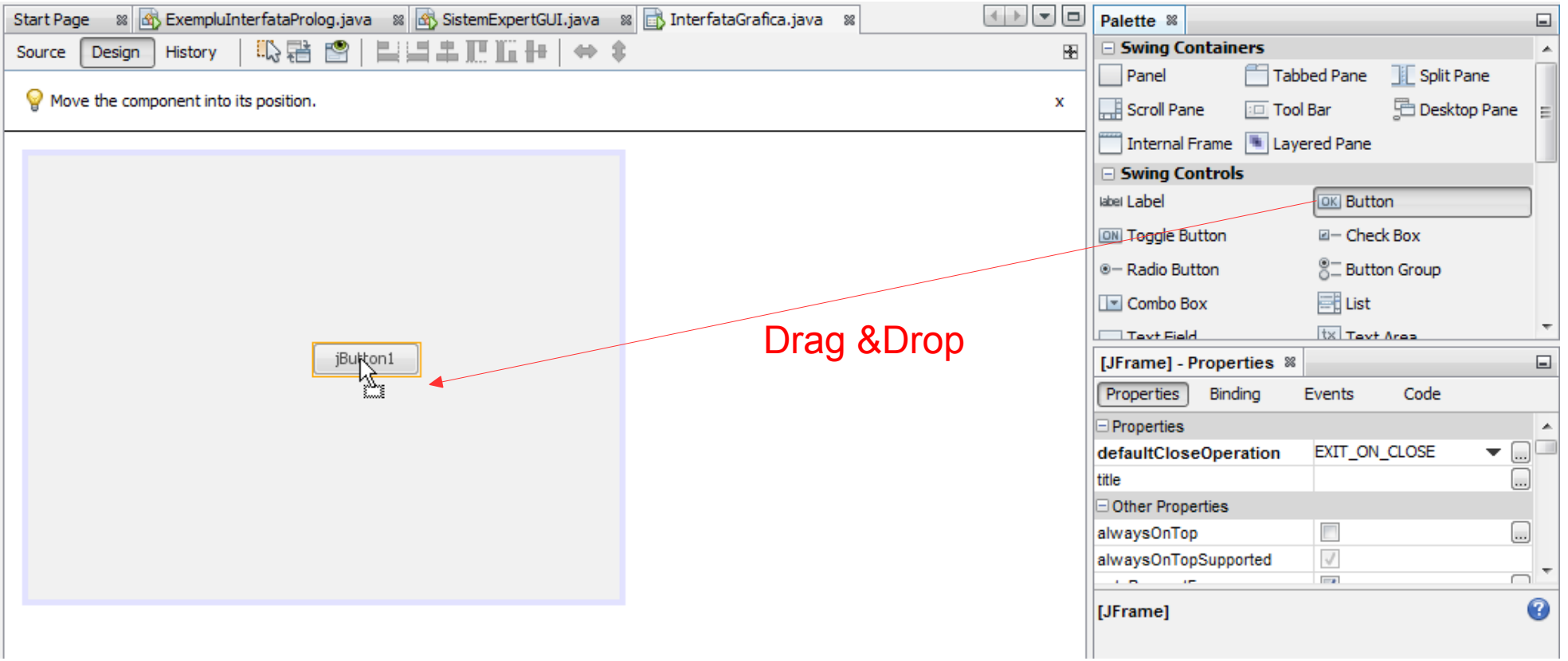


Creare proiect NetBeans



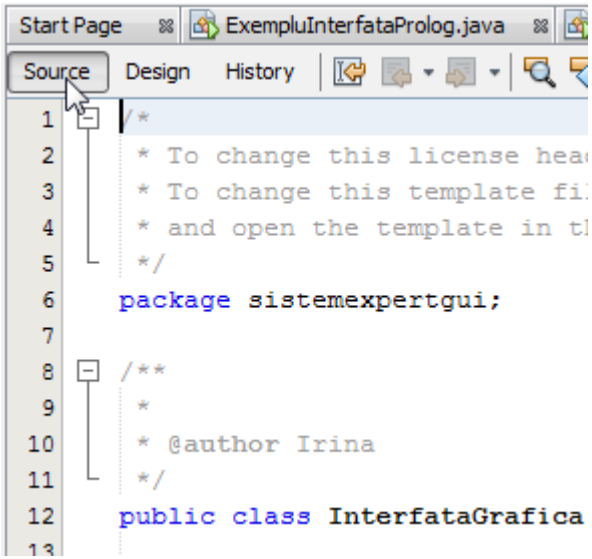


Proiect Netbeans – Creare Interfata

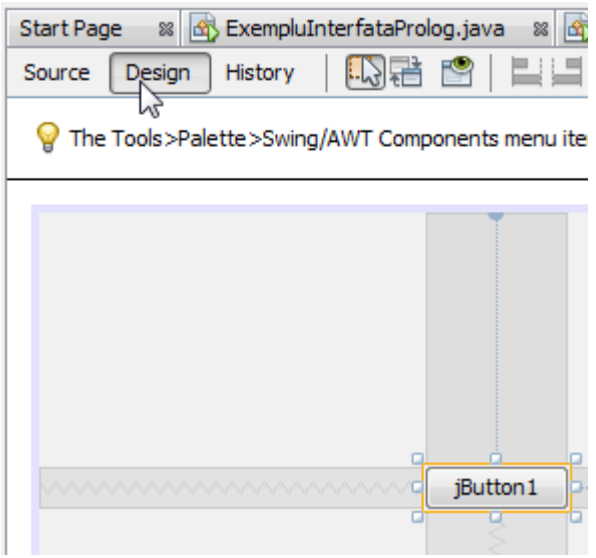


Proiect Netbeans – Creare Interfata

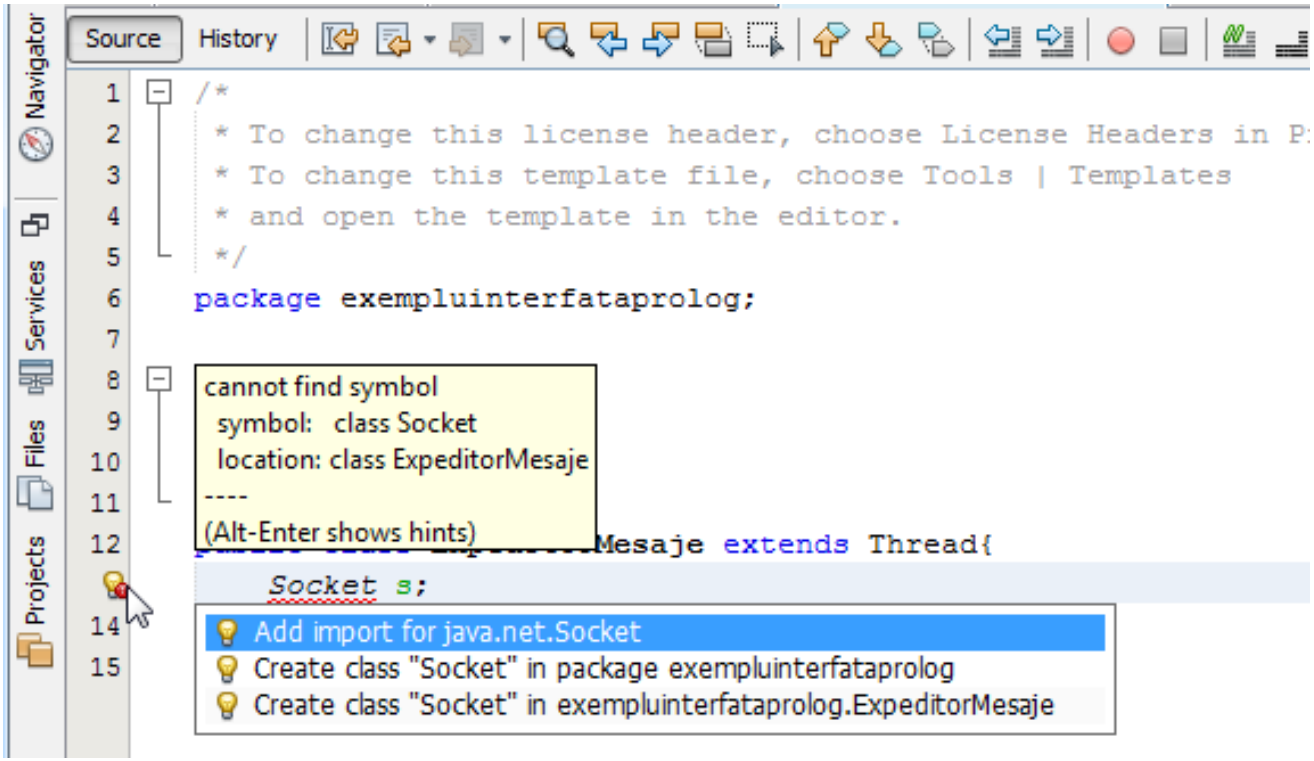
Tabul pentru codul sursa



Tabul pentru design



Proiect Netbeans – Erori



Intrebari?

Thank you! Thank you! Thank you! Thank you!

Va multumesc pentru rabdare