Medii de programare în Inteligența Artificială

Embedding JAVA with JESS

Lect. univ. dr. Mihaela Colhon

http://inf.ucv.ro/~ghindeanu

Manipularea obiectelor Java in Jess

Pe langa functiile Jess discutate pana acum, Jess are definite functii prin intermediul carora se pot crea si manipula obiecte Java. Astfel, putem sa implementam in Jess **aproape** orice putem face in Java cu exceptia faptului ca din Jess nu putem construi clase Java.

De exemplu, in loc sa contruim liste folosind formalismul Jess (prezentat in cursurile anterioare), putem sa le definim prin intermediul claselelor Java: **Map**, **Set** sau **HashMap**. Construirea de instante ale claselor Java se face prin intermediul functiei Jess **new**:

(bind ?nume_variabila_jess (new java.[nume_pachet].[nume_clasa]))

Tipul RU.EXTERNAL_ADDRESS

Jess atribute tipul RU.EXTERNAL_ADDRESS pentru a identifica obiectele in care sunt retinute instante ale claselor Java. La fel ca in Java si in Jess se poate evita identificarea claselor folosind numele pachetului din care fac parte prin folosirea functiei **import**, dupa cum urmeaza:

```
Jess> (import java.util.*)
TRUE
Jess> (bind ?hm (new HashMap))
<External-Address:java.util.HashMap>
```

Cod Java vs. cod Jess. Functia *call*

Cod Java	Cod Jess

import java.util.*;
HashMap hm = new HashMap();
hm.put("white", "255-255-255");
hm.put("black", "0-0-0");
Set s = hm.keySet();
Object[] arr = s.toArray();
	(import java.util.*)
	(bind ?hm (new HashMap))
	(call ?hm put "white" "255-255-255")
	(call ?hm put "black" "0-0-0")
	(bind ?s (?h keySet))
	(bind ?list (?s toArray))

O referinta Jess la un obiect Java poate invoca metodele definite pentru obiectul in cauza prin intermediul functiei **call** asa cum se poate observa in exemplul anterior. Primul argument al acestei functii este referinta la obiect urmat de numele metodei si de parametrii care sunt folositi in apelare. Acesti parametrii sunt convertiti automat in tipurile Java corespunzatoare.

Apelul functiilor Java din Jess

Mai mult, functia **call** poate fi omisa, apelul ei fiind considerat implicit atunci cand avem de-a face cu variabile Jess legate la obiecte Java. Deci, exemplul de mai sus poate fi in continuare simplificat din punctul de vedere al scrierii:

```
Jess> (import java.util.*)
TRUE
Jess> (bind ?hm (new HashMap))
<External-Address:java.util.HashMap>
Jess> (?hm put "white" "255-255-255")
Jess> (?hm put "black" "0-0-0")
sau mai mult:
Jess> (import java.util.*)
TRUE
Jess> ((bind ?hm (new HashMap)) put "white" "255-255-255")
Jess> (?hm put "black" "0-0-0")
```

Apelul metodelor statice in Jess

Metodele statice ale unei clase sunt un fel de funcţii globale şi se pot accesa fără să fie necesară crearea unei instanţe a clasei. Un exemplu de metodă statică este metoda *main*() a clasei principale a aplicaţiilor. Metodele statice nu pot utiliza atribute şi metode care nu sunt statice.

Atat in Java cat si in Jess metodele statice sunt apelate prin prefixarea lor cu numele clasei din care fac parte. Un bine cunoscut exemplu este functia **sleep** a clasei **Thread** care poate fi invocata astfel:

```
Jess> (call Thread sleep 1000)
```

va genera o pauza de o secunda a executiei.

In acest caz, nu mai trebuie explicit importat pachetul **java.lang** din care face parte clasa **Thread** deoarece acesta este importat automat de catre Jess.

Java in Jess

Iata un alt exemplu de functie in Jess care utilizeaza functia **System.out.println** si constanta **Short.MAX_VALUE** din Java:

```
Jess>(import java.io.*)
Jess>(deffunction is_short (?i)
        (if (> ?i (Short.MAX_VALUE)) then
        ((System.out) println “Prea mare pentru tipul short”)
        else
        ((System.out) println “Numarul poate fi stocat ca data short”)))
TRUE
Jess> (is_short 3)
Numarul poate fi stocat ca data short
```

Obiecte Java
Setting and Reading Java Bean Properties

Valorile datelor membru ale obiectelor Java se pot obtine prin intermediul functiei Jess **get-member**. In mod echivalent, instantierea unei date membru a unui obiect Java se poate face folosind functia **set-member**. De exemplu, un obiect de tipul **java.awt.Point** are doua date membru:

x coordonata relativa la axa Ox

y coordonata pe axa Oy

Astfel accesarea acestor membri se face conform codului:

```
Jess> (bind ?p (new java.awt.Point))
<External-Address:java.awt.Point>
Jess> (set-member ?p x 10)
10
Jess> (set-member ?p y 20)
20
Jess> (printout t "(" (get-member ?p x) " , " (get-member ?p y) ")"
crlf)
(10, 20)
```

Interfata grafica cu obiecte Java construita in Jess

O interfata de tip Graphical User Interface (GUI) se poate construi cu usurinta din Jess folosind instante de clase Java API. In slideul urmator prezentam un program Jess in care este definita o fereastra de dialog de tip **java.awt. Frame** populata cu un obiect de tip **java.awt.Label** si un buton **java.awt.Button**. Evenimentele generate la inchiderea ferestrei sau la apasarea butonului sunt gestionate din Jess de catre obiecte tip **Listener**. Enumeram mai jos clasele de tip **EventListener** in pachetul **jess.awt**:

- jess.awt.ActionListener
- jess.awt.AdjustmentListener
- jess.awt.ComponentListener
- jess.awt.ContainerListener
- jess.awt.FocusListener
- jess.awt.ItemListener
- jess.awt.KeyListener
- jess.awt.MouseListener
- jess.awt.MouseMotionListener
- jess.awt.TextListener
- jess.awt.WindowListener

Interfata grafica. Structura minimala

- Definim obiecte grafice java.awt:

```
(defglobal ?*nume_var* = (new java.awt.[nume_clasa]
        <lista_parametrii>))
```

- Definim o functie atasata unuie eveniment pentru obiectul grafic:

```
(deffunction perform-something (?evt)
        ...)
```

- Atasam functia definita la Listener-ul obiectului grafic:

```
(?*nume_var* addActionListener
        (new jess.awt.Listener perform-something
                (engine)))
```

Interfata grafica cu obiecte Java construita in Jess. Exemplu (1)

```
;; ******************************
(import java.awt.*)
;; ******************************
;; Declaratii variabile globale
(defglobal ?*f* = 0)
(defglobal ?*b* = 0)
;; ******************************
;; Definire functii
(deffunction create-frame ()
             (bind ?*f* (new Frame "Interfata Jess cu obiecte Java"))
             (?*f* setBackground (new Color 255 0 255))
             (?*f* setLayout (new GridLayout 2 1)))
```

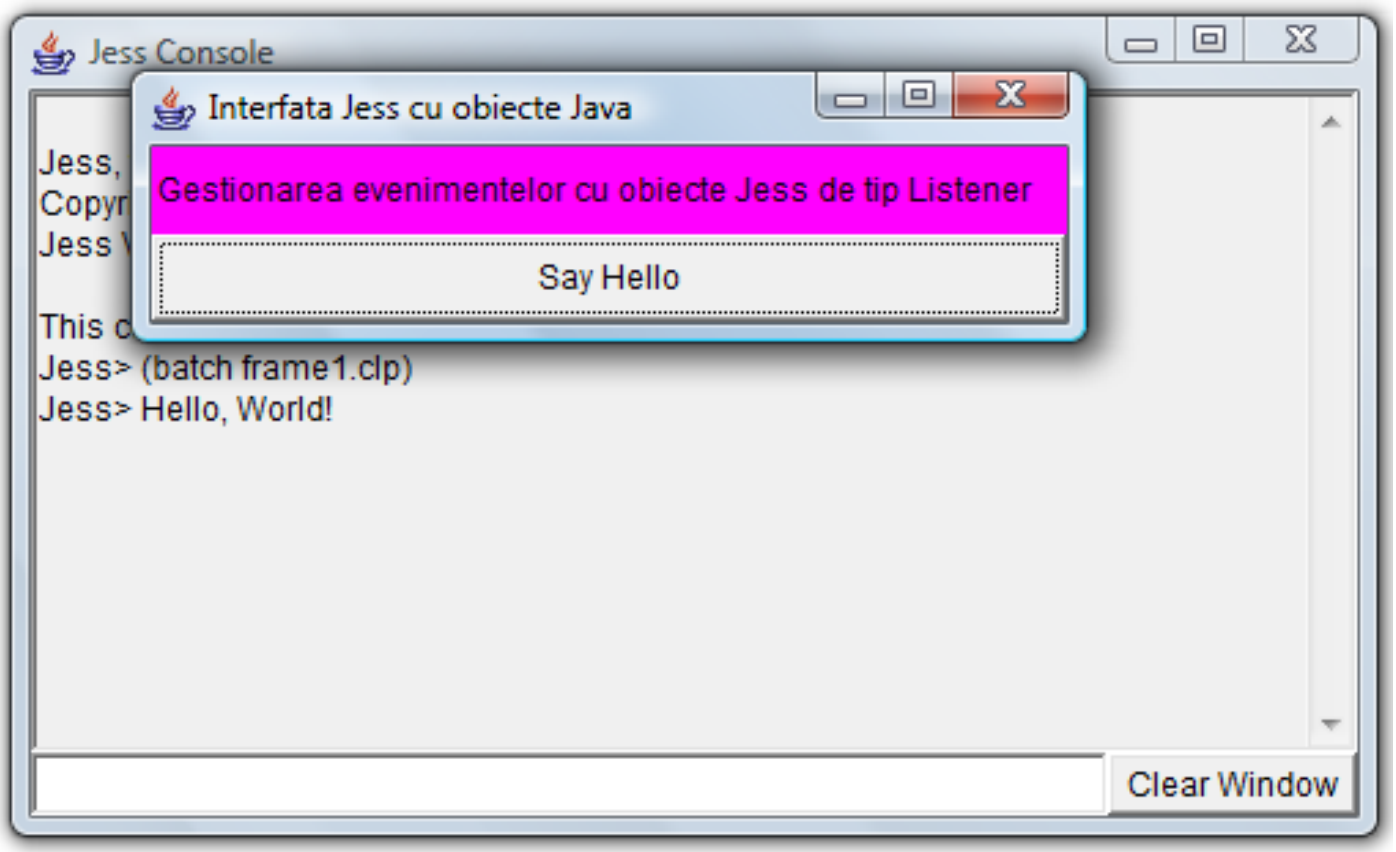
Interfata grafica cu obiecte Java construita in Jess. Exemplu (2)

```
(deffunction add-widgets ()
        (?*f* add (new Label "Gestionarea evt cu ob. Jess de tip Listener"))
        (bind ?*b* (new Button))
        (?*b* setLabel "Hello!")
        (?*f* add ?*b*))

(deffunction add-behaviours ()
        (?*f* addWindowListener (new jess.awt.WindowListener frame-handler
(engine)))
        (?*b* addActionListener  (new jess.awt.ActionListener button-handler
(engine))))

(deffunction show-frame ()
        (?*f* validate)
        (?*f* pack)
        (?*f* setSize 350 100)
        (?*f* show))
```

Interfata grafica cu obiecte Java construita in Jess. Exemplu (3)

```
(deffunction frame-handler (?event)
        ;; verificam daca ID-ul lui ?event este cel de WINDOW_CLOSING

        (if (= (?event getID) (get-member ?event WINDOW_CLOSING)) then
                (call (?event getSource) dispose)
                (call System exit 0)))

(deffunction button-handler (?event)
        (printout t "Hello, World!" crlf))

;; ******************************
;; executia programului
(create-frame)
(add-widgets)
(add-behaviours)
(show-frame)
```

Interfata grafica cu obiecte Java construita in Jess



Captarea exceptiilor in Jess

Metodele Java semnaleaza erorile rezultate in timpul apelului prin intermediului unui mecanism de aruncare al execeptiilor (in engleza, *throwing exceptions*). O astfel de exceptie este tot un obiect Java si mesajul acestuia de eroare poate fi captat de Jess in momentul in care este generat. Acelasi mecanism a fost folosit si pentru functiile Jess, iar pentru procesarea unor astfel de exceptii se poate folosi functia **(try ... catch ... [finally ...])** avand urmatoarea sintaxa:

```
(try <bloc_instructiuni>
catch <instructiuni_catch>
[finally <instructiuni_finally>])
```

Captarea exceptiilor in functia try

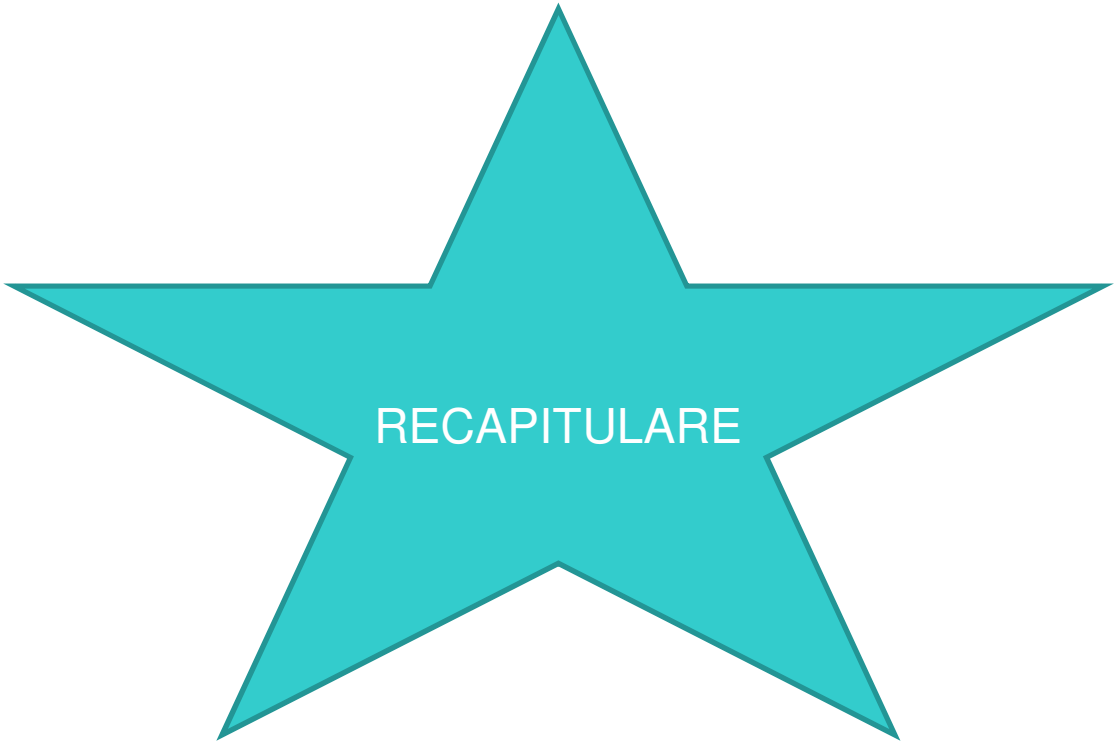
Functia **try** evalueaza expresiile din `<bloc_instructiuni>`. Daca una din acestea genereaza o eroare atunci abandoneaza executia pentru `<bloc_instructiuni>` si trece la executarea `<instructiuni_catch>`. Putem avea in continuarea blocului **catch** si un bloc suplimentar **finally**, la fel ca in Java. Obiectul rezultat in urma aparitiei exceptiei poate fi accesat in cadrul blocului **catch** prin intermediul variabilei **?ERROR**.

Deosebirea fata de blocul **try-catch** din Java consta in faptul ca in Jess nu se poate folosi decat un singur bloc **catch** iar diferentierea dintre tipurile de exceptii ce pot rezulta se face folosind metoda **instanceOf** apelata pentru variabila **?ERROR**.

Functia try. Exemplu

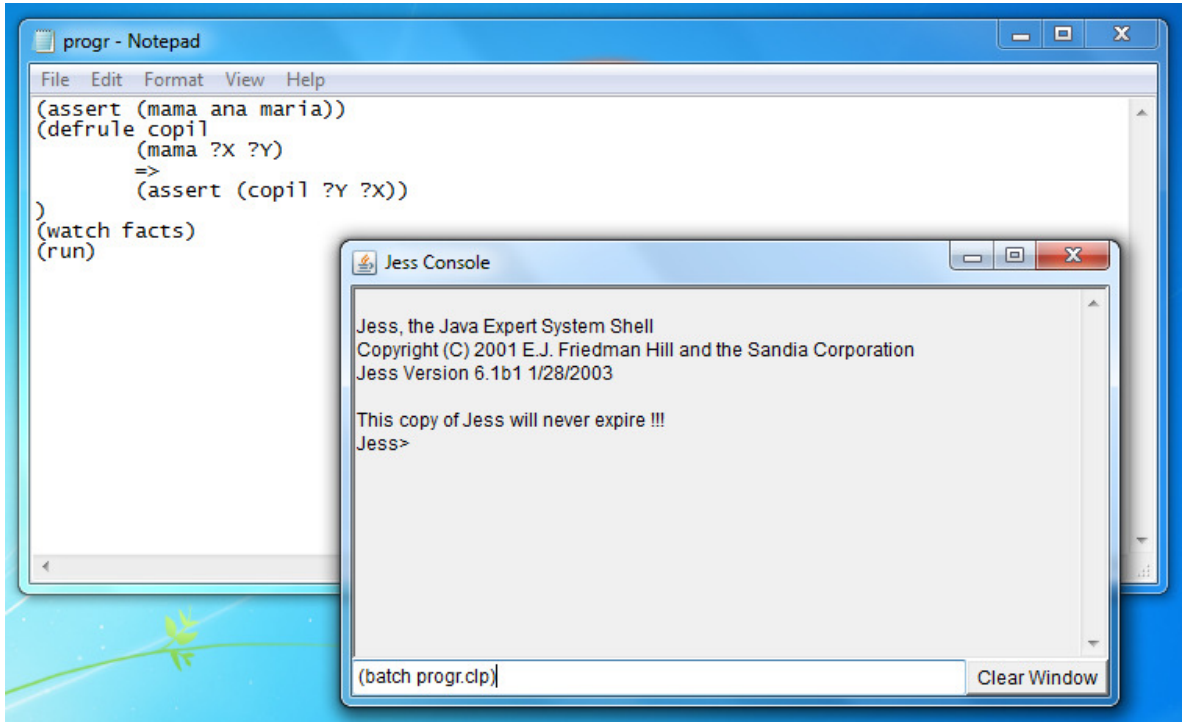
```
(import java.util.*)
(import java.io.*)

(bind ?file nil)
(try
        (printout t ”Dati numele fisierului” crlf)
        (bind ?name (read t))
        (bind ?file (new BufferedReader (new FileReader ?name)))
        (while (neq nil (bind ?line (?file readLine)))
                (printout t ?line crlf))
        catch
                (printout t "Eroare la procesarea fisierului" crlf)
        finally
                (if (neq nil ?file) then
                        (?file close)))
```

Arhitecturi Java+Jess



100% JESS

Exclusiv cod Jess

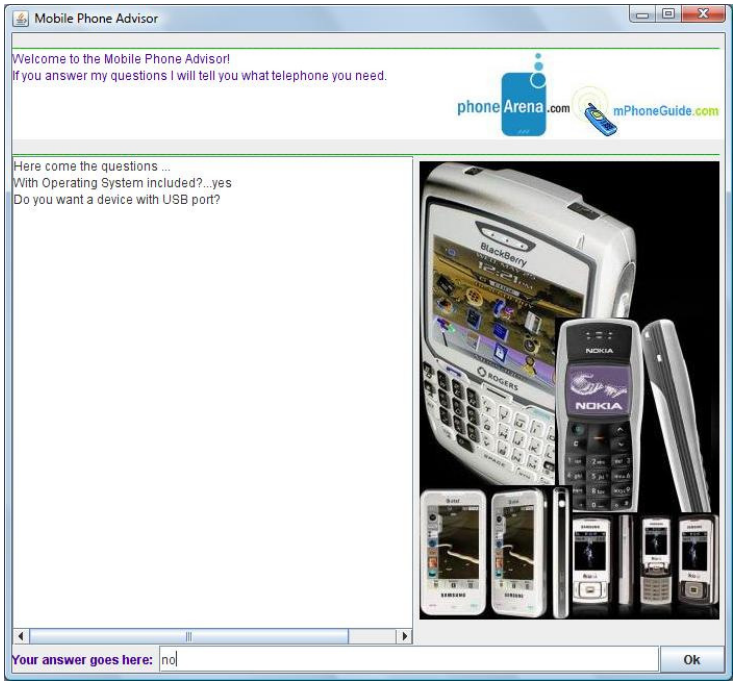


Jess+Java API

Cod Jess dar cu interfata Java API

```
(import java.awt.*)

...
(deftemplate templ_fact
        ...)
(deffacts group_facts
        (templ_fact ...)
        ...)
(defrule my_rule
        ....)
...
(reset)
(run)
```

program.clp

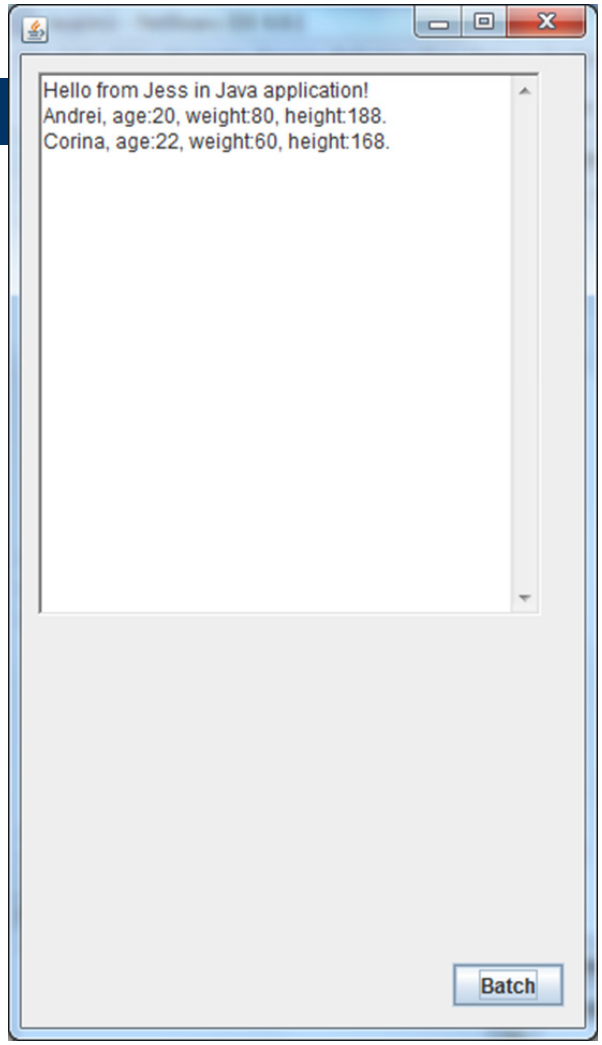


Jess+Java

Cod Jess dar cu apeluri Java

```
(import java.awt.*)
(import jess.awt.*)
...
(deftemplate templ_fact
        ...)
(deffacts group_facts
        (templ_fact ...)
        ...)
(defrule my_rule
        ....)
...
(call ...)
(reset)
(run)
```

program.clp



most Java

In marea majoritate cod Java, care incarca cod Jess in timpul executiei (laseaza in executie programe Jess).

Main.java

```
6   package javajess;
7
9   import jess.Deffacts;
    import jess.JessException;
    import jess.Fact;
    import jess.RU;
12  import jess.Rete;
    import jess.Value;
14
15
16  public class Main {
17
18
19      public static void main(String[] args) throws JessException{
20          //String pentru stocarea comenzilor Jess
21          String buff;
22          //cream o instanta a clasei de inferfata cu Jess
23          Rete engine = new Rete();
24          engine.executeCommand("(batch facts.clp)");
25          engine.reset();
26
27
28      }
29
30  }
31
```

100% Java

In totalitate cod Java

```
import jess.*
...
Rete engine = new Rete();
...
Deftemplate dt = new Deftemplate(”nume-templ”, ”comentariu”,
engine);
...
Fapt f = new Fact(”nume-fapt”, engine);
...
engine.reset();
engine.run();
```

program.java