Programarea orientata pe obiecte

Curs 2 – addon

-Clase si obiecte in Java

-Interfetele ca abstractizare

Clase si obiecte – ex din lumea reala

- Card bancar:
  - Concept abstract
  - Asigura functionalitate: verifica PIN, interogheaza sold, debiteaza cont
- Cardul BRD Sexy
  - Concept mai particularizat, card bancar, de la BRD, pe care poti sa iti pui propria poza
- Cardul lui Popescu, de la BRD, cu poza lui
  - Obiect real

Clasa (de fapt o ierarhie de clase) Later, cardul e de fapt o **interfata**

Obiect(O particularizare a unei clase)

Clase si obiecte – Exemplu cod C

*main.c

```c
#include <stdio.h>
#include <stdlib.h>

int **a;
struct node{
                    char word[20],
                    struct nod *next;
          };
node *create(void *letter)
{
    node *p;
    if(!(p=malloc(sizeof(node)))) return NULL;
    p->letter=letter;
    p->next=NULL;
    return p;
}
node *insert(node *p, void *letter)
{
    node *new_n;
        new_n=create(letter);
        new_n->next = p->next;
        p->next = new_n;
    return new_n;
}

int main()
{
    FILE *f, *g;
    char w;
    int n=0, m, i, j, max=0;
    f = fopen("in.txt", "r");
    g = fopen("out.txt", "w");

    a=calloc(100,sizeof(int*));

    for (j=0; j<100; j++) a[j] = calloc(1, sizeof(char));

    while(!feof(f)) {
                        m=0;
                        fprintf(f, "%c", &w); {m++;
```

Clase si obiecte – Exemplu cod Java

AssertTest.java | ModelGeneralBL.java

Source | History

```
23   */
24  public class ModelGeneralBL {
25
26      /**
27       * Se normalizeaza valorile Kappa.
28       *
29       * Cele care au Activ selectat pe 0 nu intra in calcul
30       *
31       *
32       * @param model
33       */
34      public static void NormalizeKappa(ModelGeneral model)
35      {
36          int i;
37          double sum;
38          sum=0;
39          for(i=0;i<Singletons.GetNrCategorii();i++)
40          {
41              if(model.getIsKappaActiv(i)>0)
42                  sum+=model.getCoefK(i);
43          }
44          if(sum>0)
45          {
46              for(i=0;i<Singletons.GetNrCategorii();i++)
47                  if(model.getIsKappaActiv(i)>0)
48                      model.setCoefK(i, model.getCoefK(i)/sum);
49          }
50      }
51      /**
52       * Se calculeaza suma coeficientilor kappa in functie daca sunt activi sau nu
53       * @param model
54       * @return
55       */
56      public static double GetSumOfActiveKappaCoef(ModelGeneral model)
57      {
58          double sum=0;
59          int i;
60          for(i=0;i<model.GetNoCategorii();i++)
61              if(model.getIsKappaActiv(i)==1)
62              sum+=model.getCoefK(i);
63          return sum;
64      }
65      /**
66       * Se verifica daca suma coeficientilor kappa e in marja de eroare
67       * @param sum
```

Clase si obiecte – Organizarea codului in Java

- Codul se gaseste DOAR in clase (`class`), DOAR in interiorul metodelor. (metode <=> functii din C)
- Clasele sunt grupate in pachete (`package`)
- Package – Class – Method – **actual code**
- Corespondenta 1-1 intre:

Clasa in java	fisierul in care e declarata
Pachet format din mai multe clase	directorul unde sunt fisierele ce declara clasele

- Memoria alocata apartine DOAR unui obiect

Clasele (1)

- Clasa (`class`) este conceptul de **incapsulare**
- Incapsuleaza codul cu memoria (variabilele) pe care lucreaza acel cod.
- Ascunde memoria (`private`)
- Expune lumii doar anumite “portite” de intrare. (`public`)

Clasele – anatomia unei clase (1)

AssertTest.java - Editor

AssertTest.java

Source | History

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package asserttest;
6
7  /**
8   *
9   * @author visoft
10  */
11 public class AssertTest {
12
13     //O variabila de exemplificare.
14     //Se observa keyword-ul private
15     private Integer variabila;
16
17     /**
18      * Comentarii foarte importante despre ce face metoda.
19      * Observati stilul de comentare, incepe cu /**
20      */
21     public void test(String[] args) {
22         // TODO code application logic here
23         int i,j;
24         i=5;
25         j=2;
26         double k;
27         k=i/j;
28         assert(k==2.5):"wrong k";
29
30     }
31 }
```

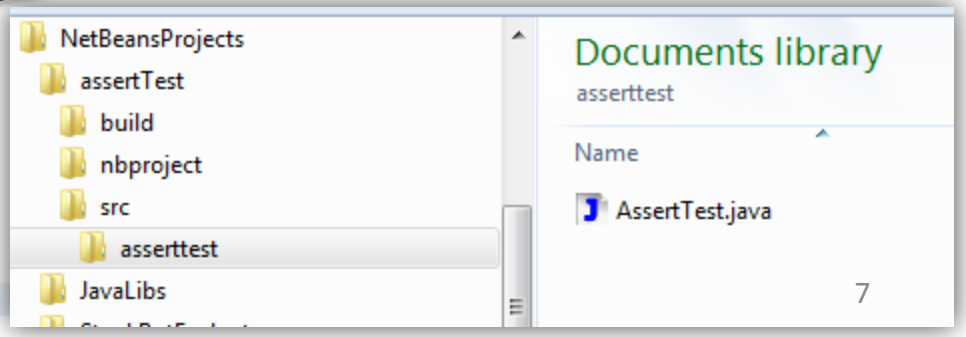
Pachetul din care face parte clasa

Numele clasei

Comentarii. Foarte importante. Se puncteaza la proiecte, colocviu, examen!

Variabile (inca nu am efectiv memoria alocata)

Cod



Clasele – anatomia unei clase (2)

- `public, private, protected, class` – keywords
- variables, methods, static variables, methods,
- etc.

- De citit din cursul III, d-nul prof Joldos

Clasele – Sumar

- Caramizile OOP. Declarate cu `class`.
- **Clasa obiectului <=> Tipul variabilei**
- Incapsuleaza cod si date
- Au “porti” de comunicare (`public`). Restul datelor/codului (`private`) este inaccesibil din exterior.
- Codul este grupat in metode (<=> functiile din C)
- Pachete – Clase – Metode – Cod
- Clasa == fisierul cu acelasi nume
- 1 clasa pe fisier (exista si clase imbricate)**.**

Obiecte (1)

- “Instantierea unei clase”
- Traducere: Clasa reprezinta un sablon, obiectul reprezinta particularizarea sablonului
  - Card debit ➔ Cardul lui Popescu, de la BRD
  - Avion ➔ Cursa Bucuresti Moscova de la ora 7.30
  - Student ➔ Popescu Ion
  - RowSet ➔Rezultatul “Select * from Vizitatori” executat pe baza de date MySQL

Obiecte (2)

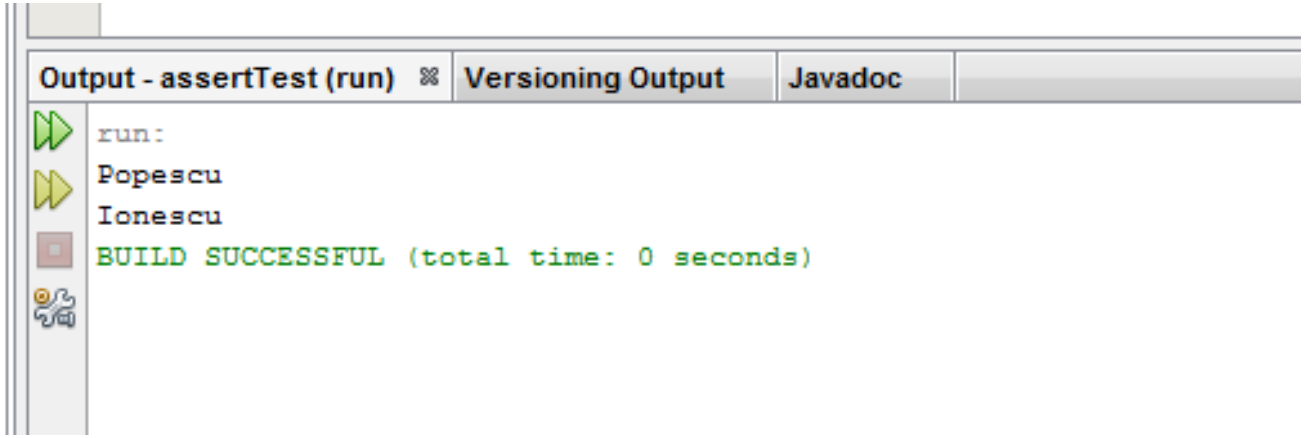
- Obiectul contine zona de memorie **declarata** in clasa
- O clasa poate genera n obiecte.
- Un obiect apartine DOAR unei clase (inca nu stim ce e aia mostenire).
- Nu exista obiecte fara clasa!!!!
- Exista clase fara obiect.
- Codul manipuleaza practic obiecte

In java, orice variabila == obiect

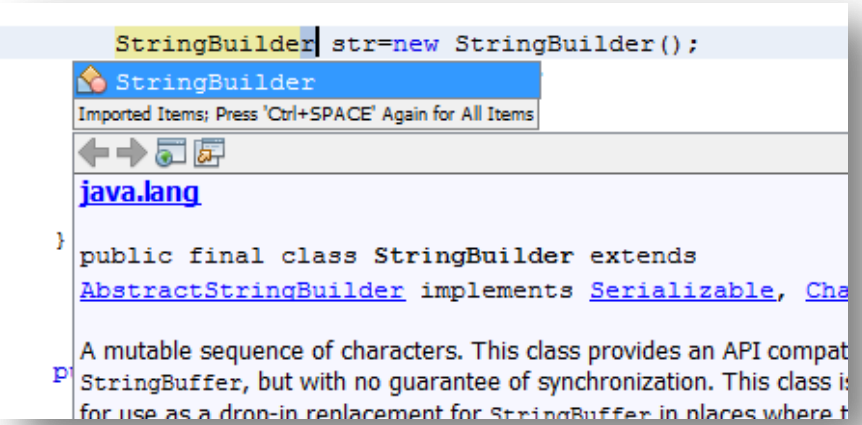
(well, cu exceptia primitivelor)

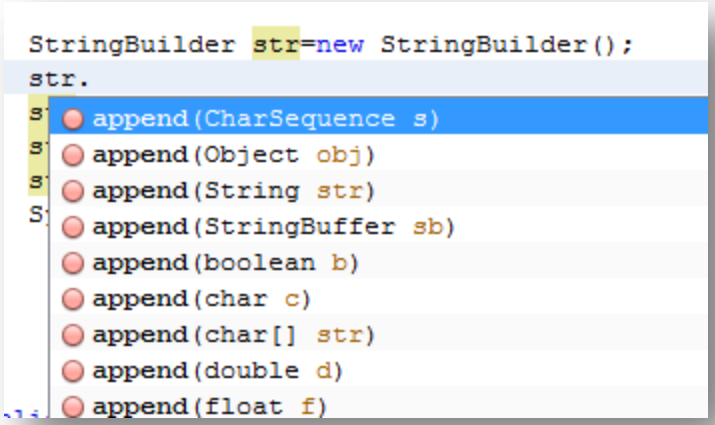
Obiecte (3)

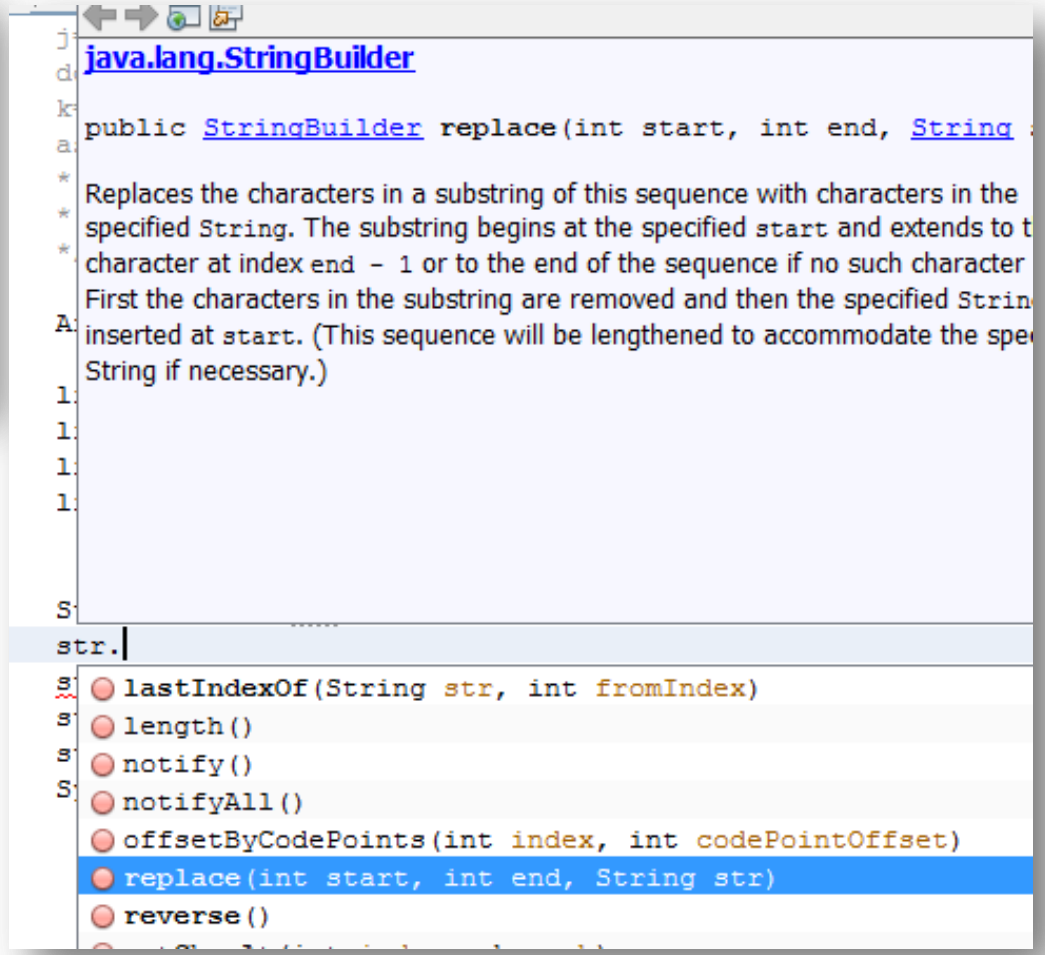
```
34
35          ArrayList<String> listaMea; // ArrayList<String> ==tipul variabilei
36                                      // listaMea == numele variabilei
37          listaMea=new ArrayList(10); // Aloc memorie. new <=> malloc din C
38          listaMea.add("Popescu");    // Apeles codul grupat in metoda add
39          listaMea.add("Ionescu");
40          listaMea.add("Iliescu");
41
42
43          StringBuilder str=new StringBuilder();
44          str.append(listaMea.get(0));
45          str.append("\n");
46          str.append(listaMea.get(1));
47          System.out.println(str);
48
```



Obiecte – De unde stiu ce si cum? (1)

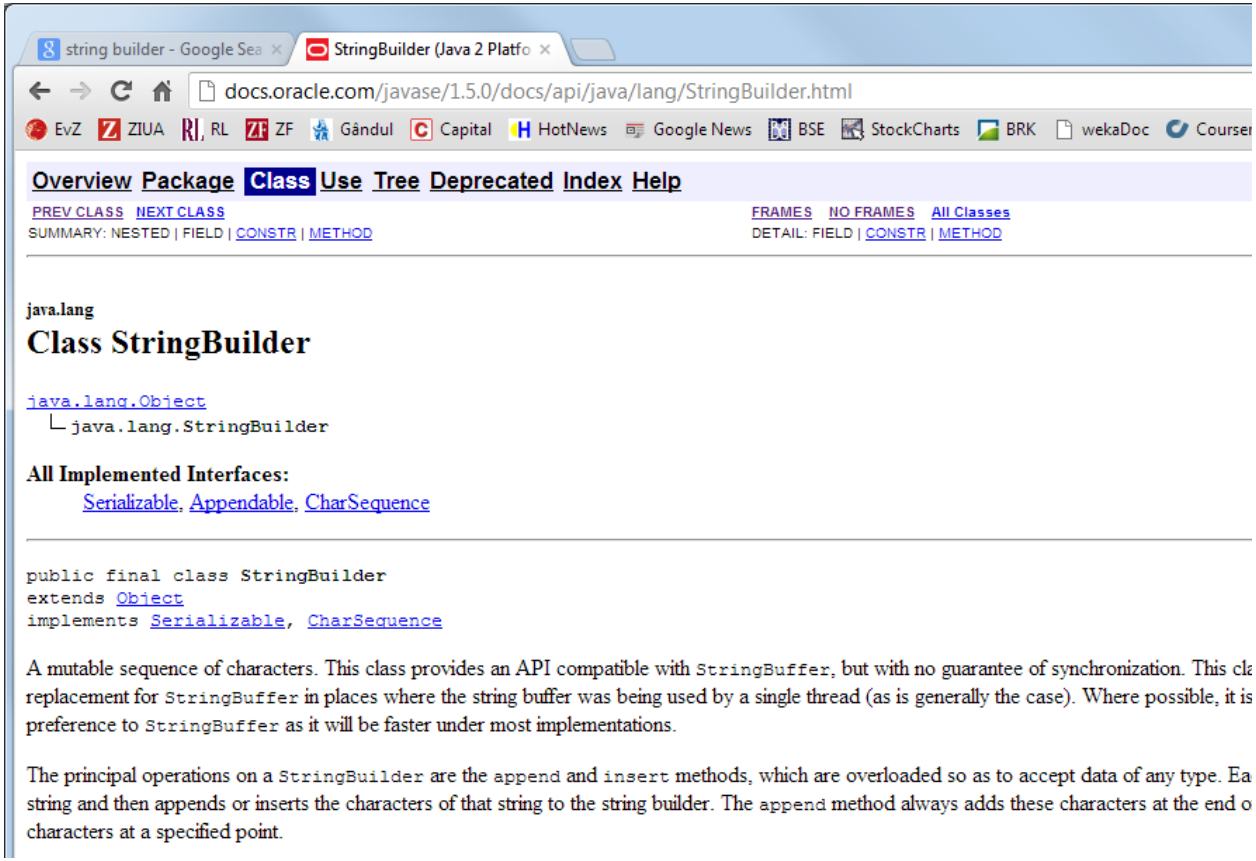






Obiecte – De unde stiu ce si cum? (2)

- Java documentation



Obiecte – De unde stiu ce si cum? (3)

- Cand scrieti clasa, scrieti si comentariile.
- Exista utilitar special care genereaza documentatie html din comentariile voastre (javadoc)
- IDE-urile mai complexe automatizeaza toate operatiile
  - Generare de comentarii specifice javadoc
  - Rularea utilitarului
  - Afisarea comentariilor atunci cand scriem cod
- Lipsa comentarii == cod ilizibil
  - Puncte scazute la examen,
  - Firma nu iti mai da bonus

Obiecte – Sumar

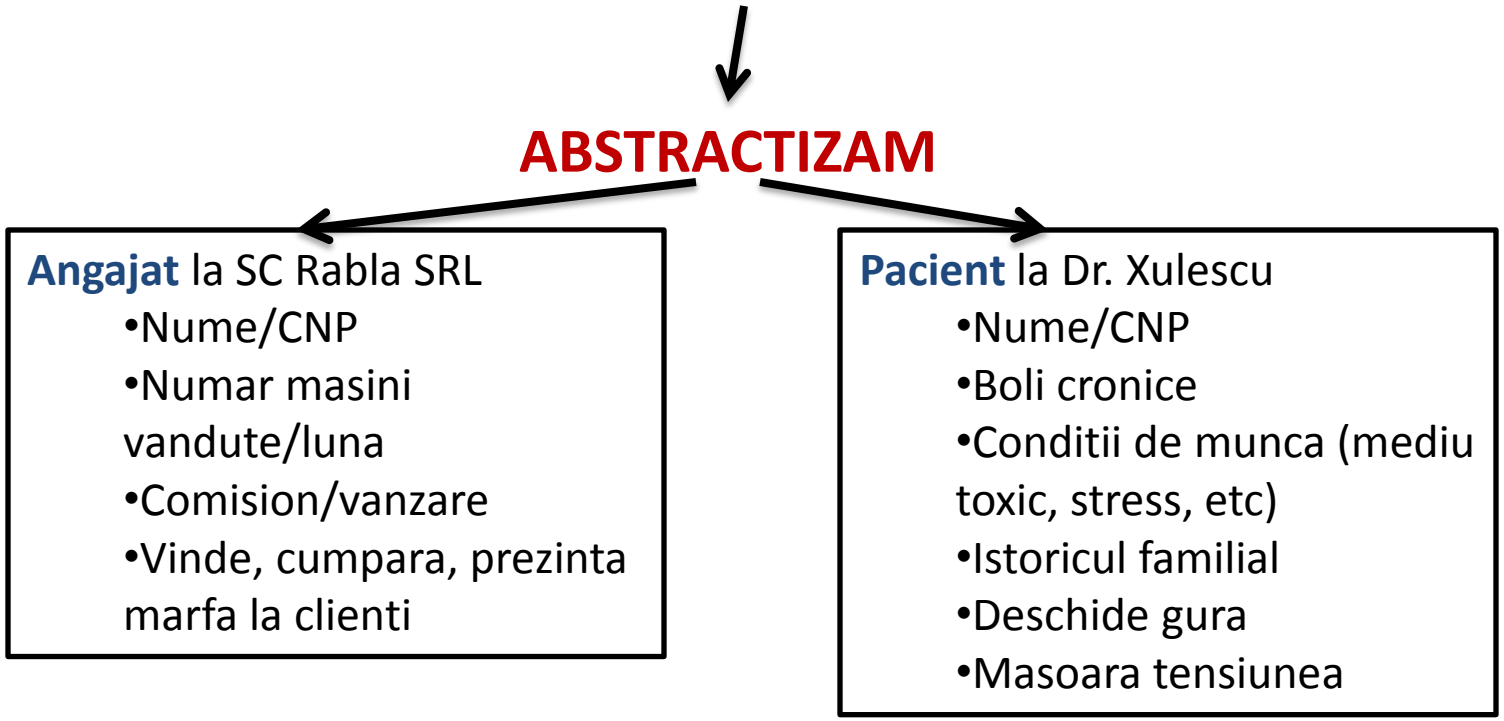
- Orice variabila in Java este un obiect
- Orice obiect are o clasa (<=> tipul obiectului)
- Orice obiect, e o **referinta** la zona de memorie corespunzatoare obiectului.
- Zona de memorie contine variabilele declarate in clasa.
- Cineva trebuie sa aloce memorie, sa puna valori la variabilele declarate acolo, etc.
  - Operatorul new(), constructorul clasei,
  - Metode care returneaza obiecte gata create
  - Boxing, unboxing (Integer, Float, String)
- Exceptie: tipurile primitive (int, float, double, char)

Abstractizarea

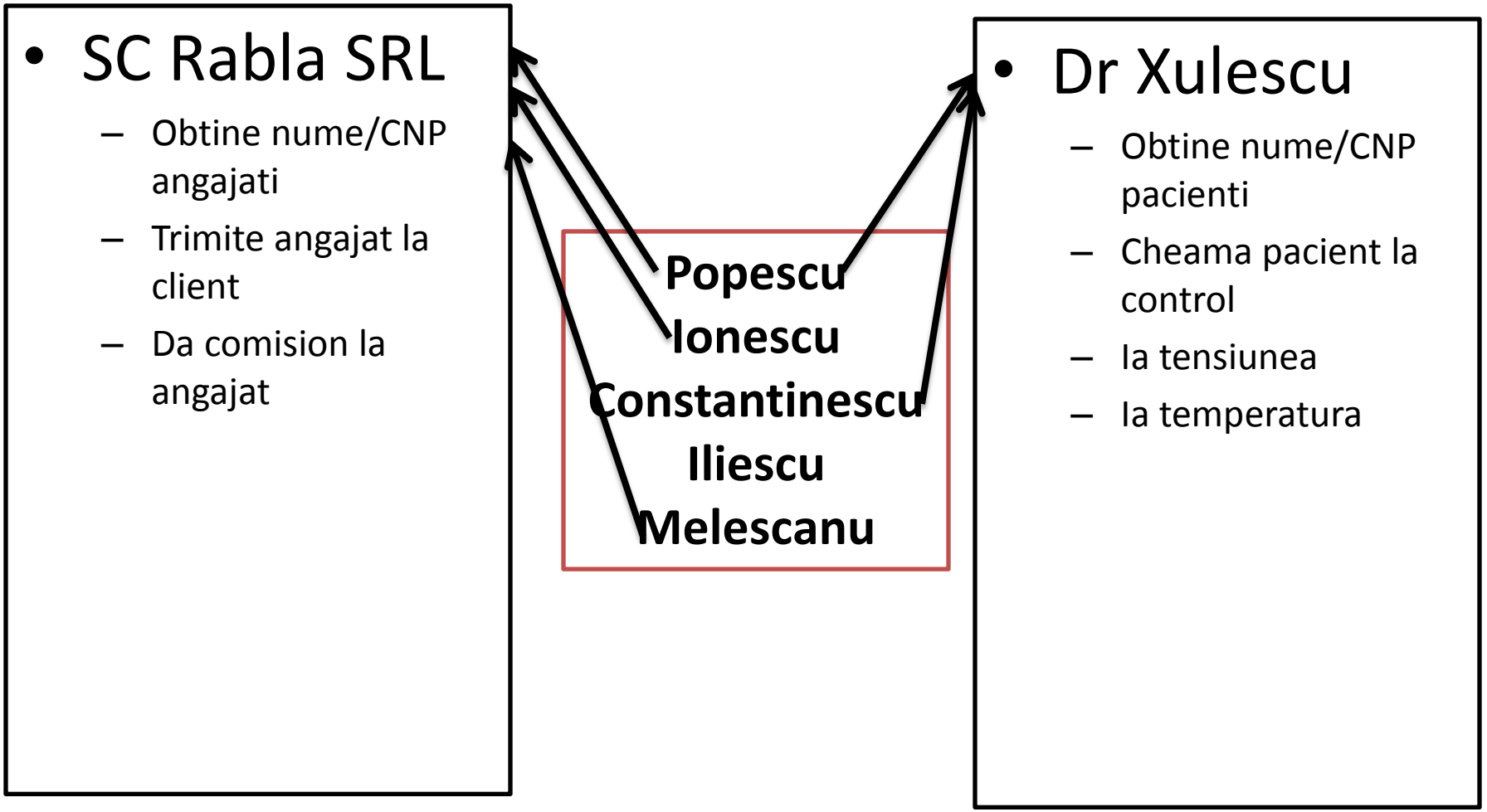
- Exemple din lumea reala
- **Interfata** ca metoda de abstractizare
- Exemplu de interfata: JDBC
- Sumar interfete

Abstractizarea – Exemple din lumea reala (1)

- Popescu, 53 de ani, agent de vanzari la o firma de masini, cardiac, cu 2 copii si 1 caine. Salar pe baza de comision.



Abstractizarea – Exemple din lumea reala (2)



Abstractizarea – Exemple din lumea reala (3)

- Mai mult: Abstractizam relatiile la
  - Angajat
    - Da nume/CNP
    - Preia sarcina de servici
    - Executa sarcina
    - Incaseaza salar
  - Pacient
    - Da nume/CNP
    - Da istoric medical
    - etc.

Interfata ca metoda de abstractizare lumea reala (1)

- **Angajatul** se obliga sa ofere anumite servicii, sa presteze o anumita munca.
- **Pacientul** se obliga sa isi dea numele/CNP, sa se lase examinat etc.

- Firma stie ca toti angajatii ofera servicii
- Medicul stie ca toti pacientii se lasa examinati

- Un om poate fi **pacient**, **angajat**, **proprietar**, **chirias**, etc.

Interfata ca metoda de abstractizare lumea reala (2)

- Un angajator poate abstractiza toti oamenii din firma privindu-i ca **angajati**.
- La fel si un medic, abstractizeaza toti oamenii care vin la consult ca **pacienti**.

- Faptul ca cineva spune ca este **angajat/pacient/proprietar** inseamna ca se obliga sa ofere anumite servicii definite de statusul care si-l asuma.
- Cineva care poate interactiona cu **proprietari**, poate interactiona cu oricine care isi asuma rolul de proprietar

Interfata ca metoda de abstractizare Exemplu software (3)

- Un SGBD (Baza de date) poate fi Oracle DB, MySQL, MS SQL Server, etc.
- Trebuie sa ofere servicii:
  - Connect to DB
  - Retreive DB content
  - Retreive records
  - Prepare query
  - Execute query

Interfata ca metoda de abstractizare Exemplu software (4)

- Java a creat interfata: "**JDBC(TM) Database Access**" (http://docs.oracle.com/javase/tutorial/jdbc/index.html)
- Eu, ca programator, pot sa scriu cod folosind operatiile specificate acolo.
  - getConnection()
  - prepareStatement()
  - executeUpdate()
- Codul meu va functiona cu orice baza de date ce asigura driver pentru **JDBC**

Interfata ca metoda de abstractizare Exemplu software (5)

- Java a creat interfata: "**JDBC(TM) Database Access**" (http://docs.oracle.com/javase/tutorial/jdbc/index.html)
- Eu, ca dezvoltator, pot sa fac un sistem revolutionar de gestiune, care implementeaza ceea ce cere JDBC:
  - getConnection()
  - prepareStatement()
  - executeUpdate()
- Sistemul meu minune va functiona la orice clinet care are softul scris conform **JDBC**

Interfata ca metoda de abstractizare Exemplu software (6)

- **Package java.sql** (http://docs.oracle.com/javase/7/docs/api/)
- Interfaces:
  - Connection
  - Driver
  - PreparedStatement
  - ResultSet

```
1   /*
25
26  package java.sql;
27
28  import java.util.Properties;
29
30  /**
83  public interface Connection  extends Wrapper {
84
85      /**
103     Statement createStatement() throws SQLException;
104
105     /**
137     PreparedStatement prepareStatement(String sql)
138         throws SQLException;
139
140     /**
170     CallableStatement prepareCall(String sql) throws SQLException;
171
172     /**
184     String nativeSQL(String sql) throws SQLException;
185
186     /**
221     void setAutoCommit(boolean autoCommit) throws SQLException;
222
223     /**
233     boolean getAutoCommit() throws SQLException;
234
235     /**
248     void commit() throws SQLException;
249
250     /**
262     void rollback() throws SQLException;
263
264     /**
265      * Releases this <code>Connection</code> object's database and JDBC resources
266      * immediately instead of waiting for them to be automatically released.
267      * <P>
268      * Calling the method <code>close</code> on a <code>Connection</code>
269      * object that is already closed is a no-op.
```

Sumar interfete

- **Interfata** (`interface`) este un **contract** standardizat intre 2 sau mai multe parti.
- Contractul obliga asigurarea unei anumite functionalitati
- Cel ce implementeaza (`implements`) interfata se obliga sa implementeze TOATE functionalitatile
- Cel ce foloseste interfata (ex: voi, in cod) stiti sigur ca aveti la dispozitie functionalitatile necesare si “Don’t know/Don’t care” despre cum anume sunt implementate (MS SQL Server, MySQL, etc)

- Cum se declara/foloseste, la partea de Java!