Proiectare şi programare orientată pe obiecte

Prof. univ. dr. Tudor Bălănescu

Bibliografie

1. *Catrina, O.; Cojocaru, I.:* ***Turbo C++,*** Teora, 1993
2. *Deitel, H. M.; Deitel, P.J.:* ***C++, How to Program,*** Prentice Hall, 2001,3rd edition
3. *Eckel, B.:* ***Thinking in Java,*** Prentice Hall, 1998.
4. *Gries, D; Gries, P.:* ***Multimedia Introduction to Programming Java,*** Springer, 2005
5. *[Meyer] Meyer, B.:* ***Object Oriented Software Construction***, Prentice Hall, 1997, 2nd edition
6. *Schildt, H.:* ***C++, manual complet***, Teora, 1997.
7. *Tănasă, Ş.; Olaru, C.; Andrei, Ş:* ***Java, de la 0 la expert,*** Polirom, 2003

Istoric şi motivaţie

- Eficienţa activităţii de programare şi calităţile sistemelor de programe depind, printre altele, de:
    1. Nivelul de abstractizare şi de specializare suportat de instrumentele utilizate (limbaje, biblioteci de programe etc.)
    2. Tehnologiile de proiectare a sistemelor software,
    3. Modul de organizare al echipelor (management)

1. Abstractizare. Exemplu.

abstractizare

– Limbaj maşină → limbaj de nivel înalt
(conţine detalii hardware) (independent de

detalii)

```
mov        ax,word ptr [bp-4]            x=y+z;
add        ax,word ptr [bp-6]
mov        word ptr [bp-2],ax
```

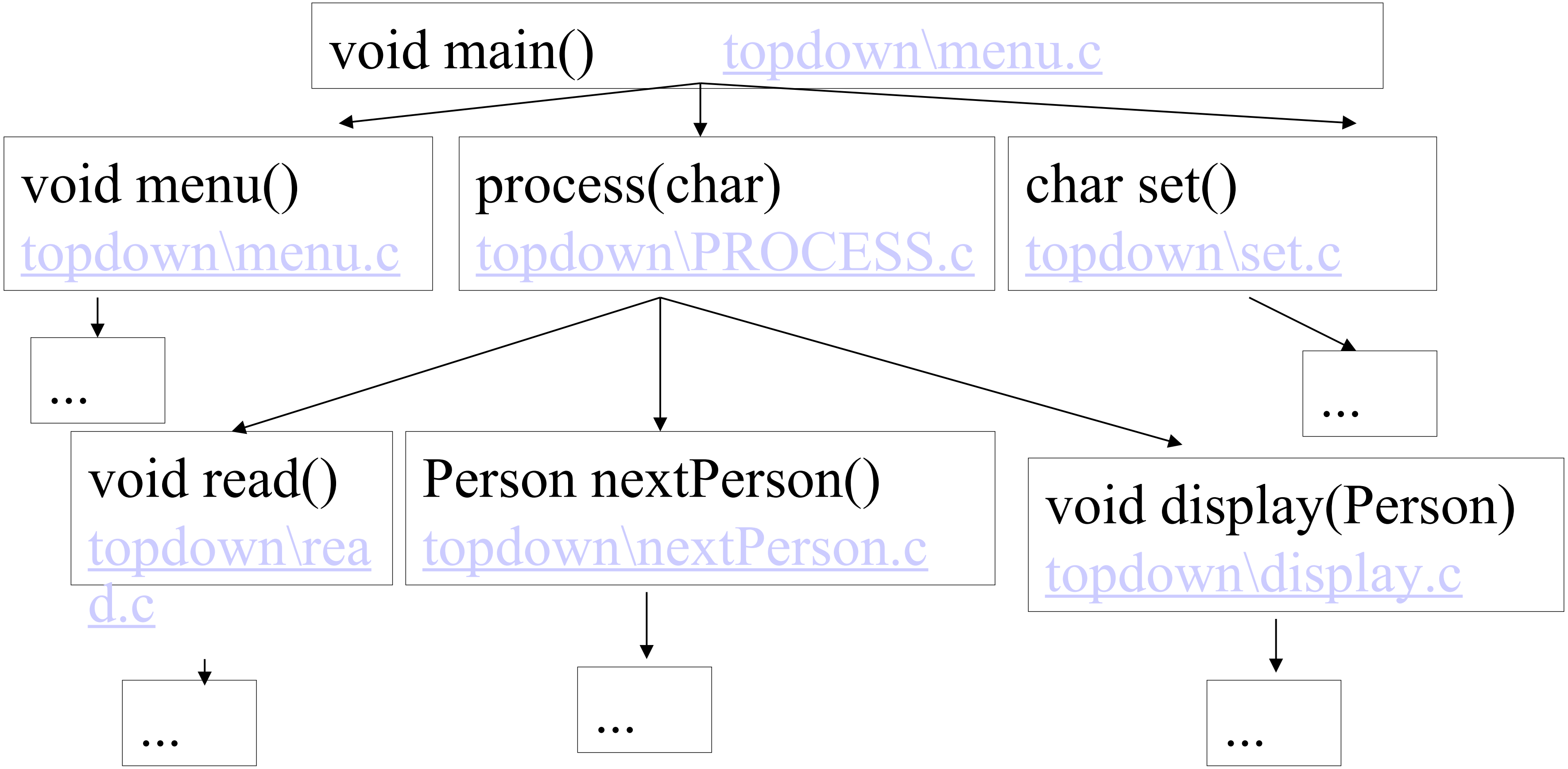
(operaţii: *add, mov* (operaţii: =, +
date: *ax,word ptr [bp-4] etc.*) date: *x,z,y*)

continuare

2. Tehnologie de proiectare. Exemplu. Programare structurată, proiectare top-down.
(Program development by stepwise refinement, Nicklaus Wirth, 1971)
Să se scrie un program cu ajutorul căruia să se introducă şi să se afişeze informaţii despre persoanele dintr-o întreprindere. Fiecare persoană este identificată prin nume şi vârstă. Tipul de operaţie care se execută la un moment dat (introducere sau afişare) este specificat interactiv de utilizator.

```
void main(void){
        char toDo;
        do{
                menu();
                toDo=set(); // setare tip operatie
                process(toDo);
        }while (toDo != QUIT);
}
```

Abstractizare funcţională



void main() topdown\menu.c

void menu() topdown\menu.c

process(char) topdown\PROCESS.c

char set() topdown\set.c

...

...

void read() topdown\read.c

Person nextPerson() topdown\nextPerson.c

void display(Person) topdown\display.c

...

...

...

continuare

- Observaţii
  - Abstractizarea a fost concentrată asupra *algoritmilor de prelucrare* şi a condus la o ierarhie de funcţii. Prin acest proces de abstractizare,
    - a fost stabilită interfaţa de comunicare între funcţii (prin precizarea signaturii lor)
    - au fost ascunse detaliile de implementare
  - Ca o consecinţă a abstractizării funcţionale:
    - metodologia permite distribuirea sarcinilor pe echipe
    - funcţiile pot fi testate separat, utilizând funcţii stub pentru descendenţii direcţi
  - Abstractizarea datelor prelucrate este un *proces secundar,* care însoţeşte procesul de rafinare descendentă a funcţiilor (tipul structură `Person` este definit la nivelul 2, odată cu rafinarea funcţiei `process(char))` topdown\PERSON.H

```
typedef struct{
    char *nume;
    int varsta;
}Person;
```

Deficienţe ale stilului de programare structurată 1/2

1. Distanţă textuală potenţial nelimitată între procedurile de prelucrare a datelor (funcţii) şi descrierea structurii datelor (tipurile de date) (ceea ce impune ”răsfoirea” frecventă a fişierelor)
2. Acces nerestricţionat la câmpurile structurilor (încălcarea principiului Information Hiding), cu consecinţele:
   - setare improprie: `p.varsta= -1;`
   - modificare neatorizată:
     if (p.nume==“Balanescu” p.account+=10000;)
3. Limbajele nu au mecanisme de asociere obligatorie a operaţiei de iniţializare la declararea unei variabile.
   Exemplu.

```
Person  nextPerson(void){
         Person p; // initializare omisa,
         return p;
// cu consecinte negative asupra functiilor care
//utilizeaza rezultatul returnat,
// ex. display(nextPerson()); din process()topdown\PROCESS.c
}topdown\nextPerson.c
```

Deficienţe ale stilului de programare structurată 2/2

4. Adaptare pentru reutilizarea în alte contexte a programelor existente (principiiile Extensibility, Reusability)
   - dificilă, în cazul reutilizării textului sursă (practica Open Source, specifică pentru comunitatea Unix, Lisp etc) :
     - contradicţie cu principiul Information Hiding
     - intervenţiile în textul sursă pot afecta negativ funcţionalitatea de dinainte de intervenţie
   - practic imposibilă în absenţa textului sursă; din raţiuni economice, firmele software nu practică diseminarea textelor sursă; clienţilor le sunt furnizate biblioteci de module obiect (*.obj, *.dll)
5. Diseminare exhaustivă a detaliilor structurale ale variantelor unei anumite entităţi (încălcarea principiului Single Choice: whenever a software system must support a set of alternatives, one and only one module in the system should know their exhaustive list)

(Exemplu. Considerând variantele male-female la tipul Person, atunci aceste alternative trebuie tratate in read(), display() etc.)

Schimbarea listei de alternative atrage modificarea tuturor modulelor care enumeră alternativele.

Concepte de programare orientată pe obiecte

Corectarea deficienţelor semnalate:

1. şi 2. :

- **Încapsulare:** ***includerea*** procedurilor şi datelor într-o singură structură sintactică, numită `clasă` (`class`) şi ***asocierea unor nivele de acces*** (`public, protected, private` etc.)
  topdown\varianta POO\Person.hpp

```
// file Person.hpp
#ifndef PERSON_HPP
#define PERSON_HPP
class Person{
public:  // iterfata
       Person(char *name, int age); //constructori
   Person();
   void read();
   void display();
   void setName(char *newName); // metode de modificare (modificatori, setters)
   void setAge(int newAge);
   char* getName(){return name;}  // metode de interogare (getters)
   int getAge(){return age;}      // indicatie de implementar inline
private: // detalii de implementare, Information hiding
       char * name;
   int age;
};
#endif
```

- Clasa = tip de date, elementele sale se numesc obiecte (`objects`)

```
Person p, *q, v[]={Person(),Person("Andrei",6 )};
q=new Person[4];
q=v;
```

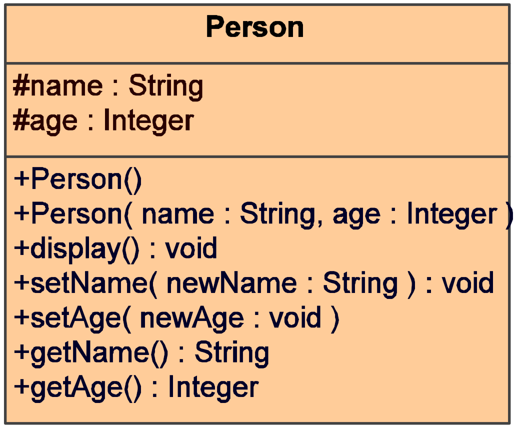
- Specificarea clasei
  (Unified Modeling Language)() stabileşte:
  - Atributele obiectelor din componenţa sa
  - Signatura metodele de prelucrare

  topdown\varianta POO\Person.eps

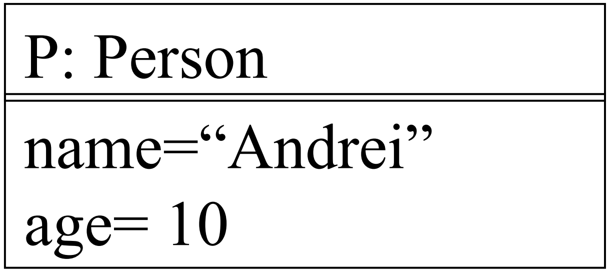
Modelare Unified Modeling Language

- Clasa = tip de date, elementele sale se numesc obiecte (objects)
  Person p, *q, v[]={Person(),Person("Andrei",6)};
  q=new Person[4];
  q=v;
- Specificarea clasei
  (Unified Modeling Language)() stabileşte:
  - Atributele obiectelor din componenţa sa
  - Signatura metodele de prelucrare

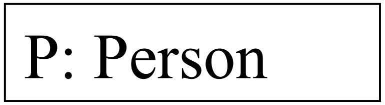
    topdown\varianta POO\Person.eps


Person
#name : String
#age : Integer
+Person()
+Person(name : String, age : Integer)
+display() : void
+setName(newName : String) : void
+setAge(newAge : void)
+getName() : String
+getAge() : Integer

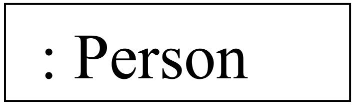
- Specificare obiect: (nume?,clasa, valorile atributelor?)


P: Person
name="Andrei"
age= 10

sau


P: Person

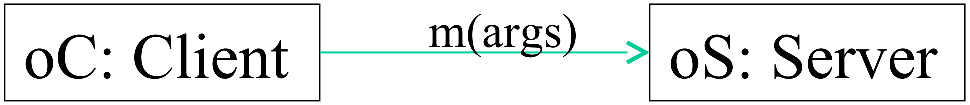
sau


: Person

Terminologie OOP

- Clase, obiecte, atribute, metode
  - Clasa defineşte caracteristicile (atributele şi metodele de prelucrare) obiectelor din componenţa sa
  - Un obiect dintr-o clasă este definit prin valorile atributelor sale

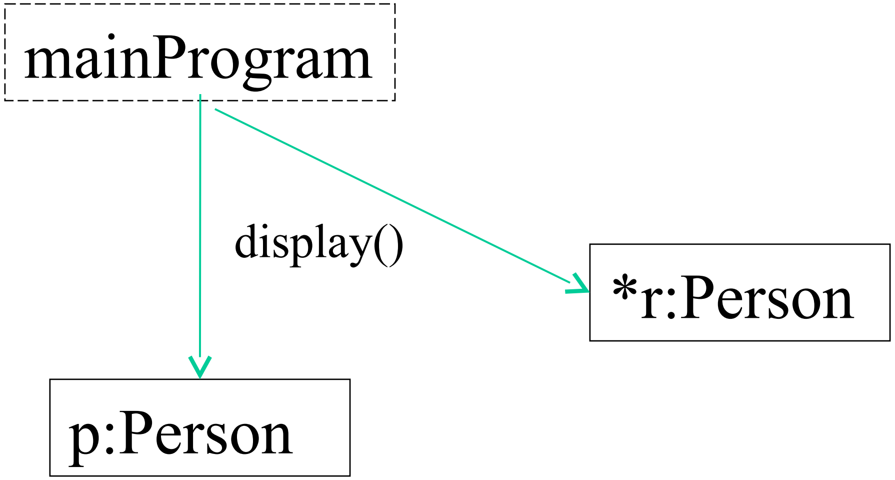
  - O metodă este o funcţie definită în cadrul unei clase
- Interacţiunea dintre obiecte: relaţia Sever/Client
  Într-un sistem software, obiectele interacţionează prin transmiterea de *mesaje.*
  - Un mesaj este un nume de procedură şi o listă de argumente `m(args);` are forma unui apel de funcţie;
  - Interacţiunea presupune existenţa unui obiect *client* `oC` care transmite un mesaj către un alt obiect `oS`, numit obiect *server*.
    - Expresia prin care se transmite mesajul are forma `oS . m(args)`
    - Obiectul server se numeşte *destinaţie* sau *obiect curent* al mesajului
    - activarea metodei mesaj `m` constituie *răspunsul* obiectului server `oS` la solicitarea obiectului client `oC.` În prelucrările pe care le realizează, metoda poate utiliza atât argumentele din lista `args` cât şi atributele obiectului curent `oS`



Exemplu

Program de verificare sintactică a compatibilităţii dintre interfeţe (test driver): topdown\varianta POO\TESTDRIV.CPP

```
#include "Person.hpp"
#include <iostream.h>
Person p,q("Tudor",58), *r;
void main(){
    p.display();
    cout<<endl;
    p.setName(q.getName());
    p.setAge(q.getAge());
    p.display();
    cout<<endl;
    r=&p;
    r->display();
    cout<<endl;
    r->read();
    r->display();
}
```



Poate fi (doar) compilat: dacă nu are erori, interfeţele sunt compatibile.

- Diagrama interacţiunii dintre obiecte:
  - Datorită caracterului hibrid al limbajului C++, unul din obiectele din diagramă nu este creat explicit de programator (este desemnat prin numele `mainProgram`)

Implementare

- topdown\varianta POO\Person.cpp

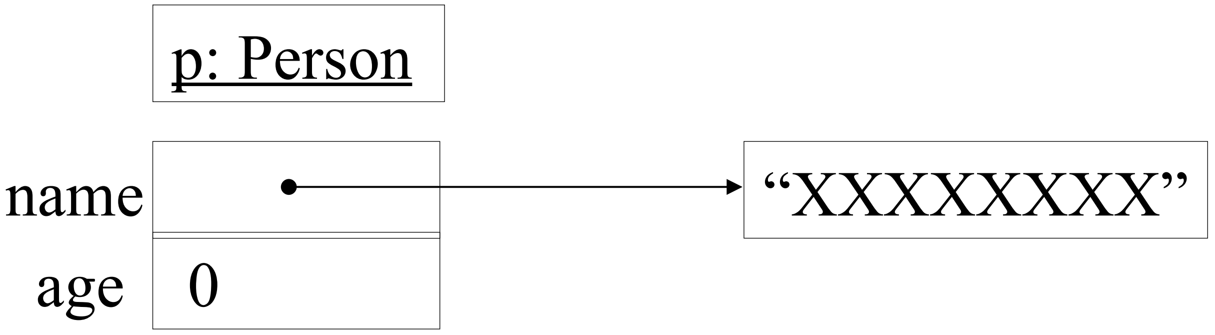
```
//file Person.cpp
#include "Person.hpp"
#include <iostream.h>
Person::Person(char *name, int age){
    this->name=name;  // semantica prin referinta
   this->age=age;
}
Person::Person(){this->name="XXXXXXXX"; this->age=0;}
void Person::read(){
    cout<<"Name of the person, please: "<<endl;       cin>>name;
    cout<<"Age: "<<endl;          cin>>age;
}
void Person::display(){
    cout<<"Name: "<<name<<" Age: "<<age<<endl;
}
void Person::setName(char *newName){
    name=newName; // semantica prin referinta
}
void Person::setAge(int newAge){
    // validare argument
   if(newAge>=0 && newAge<=200) age=newAge;
}
```

Constructori

- Metodele speciale `Person` au rol de a construi obiecte (prin alocarea memoriei necesare pentru atribute) şi (eventual) de a le iniţializa
- Există deci un mecanism de asociere obligatorie a operaţiei de iniţializare cu declararea unei variabile (corectare deficienţa 3).
- Declararea

  ```
  Person p;
  ```

  construieşte (pe stiva de executare) şi **iniţializează** obiectul din diagramă. Obiectul este eliminat din stiva (distrus) la ieşirea din blocul în care a fost declarat (memoria alocată este eliberată)



Gestionarea memoriei

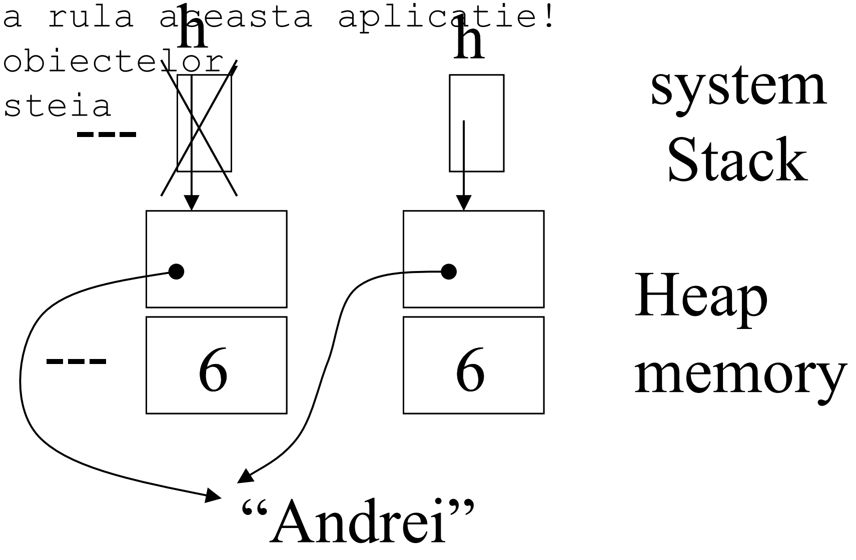
- Declararea
  char *n="Andrei";

  Person *h=new Person(n,6);
  construieşte (şi iniţializează) un obiect în zona de memorie heap.
- Urmatorul program epuizează memoria heap şi sistemul se va bloca într-un tîrziu (criză de memorie, “*memory leak*”).
- *Memory leak* este o eroare tipică limbajelor de programare fără gestionare automată a memoriei (*garbage collector*), precum C, C++ etc.

opdown\varianta POO\Destructor\ANDUR2.CPP

```
//file andur2.cpp
//Person FARA destructor, CRIZA DE MEMORIE
#include "Person.hpp"
// salvati toate fisierele inainte de a rula aceasta aplicatie!
// se aloca memorie pentru atributele obiectelor,
// in zona heap, pana la epuizarea acesteia
char *a="Andrei";
void f(){
   Person *h=new Person (a, 6);
   h->display();
   //delete h;
}
void main(){
    while(1) f();

}
```

Acumulare de obiecte

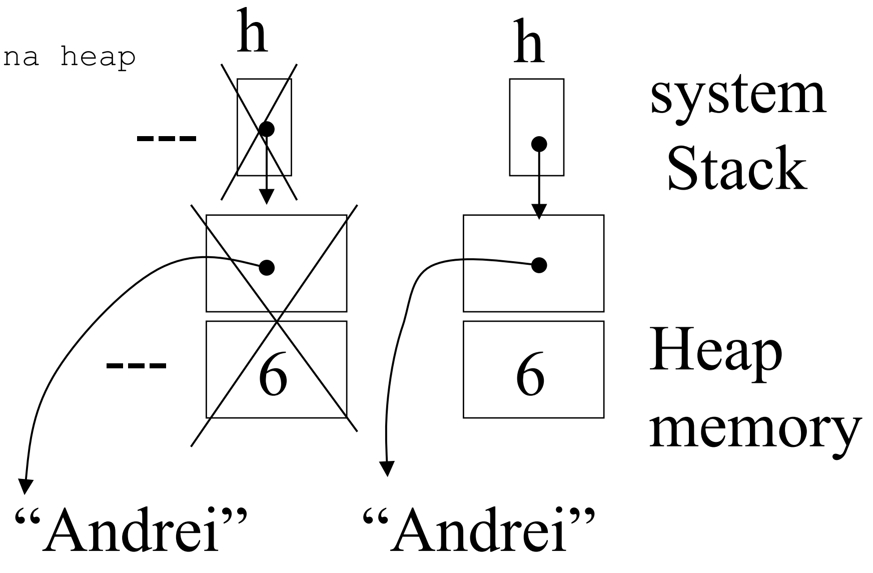


Gestionarea explicită, prin program, a memoriei (delete)

- Criza de memorie din exemplul anterior poate fi evitată prin ştergerea explicită a obiectelor alocate în zona heap (delete).
- Metoda nu este suficientă, după cum se vede în exemplul următor. Deşi memoria alocată pentru atribute este gestionată corect (delete), se acumulează in heap memoria pentru **resursele** obiectelor (valori ale atributelor).

topdown\varianta POO\Destructor\ANDUR4.CPP

```
//file andur4.cpp
//Person FARA destructor,cu DELETE, CRIZA DE MEMORIE
// salvati toate fisierele inainte de a rula aceasta aplicatie!
// se aloca memorie pentru RESURSELE obiectelor.
// in zona heap, pana la epuizarea acesteia
#include "Person.hpp"
#include <string.h>
char *a="Andrei";
void g(){
    // valoare pentru atributul name, in zona heap
    char *n=new char[strlen(a)+1] ;
    strcpy(n,a);
    Person *h=new Person (n, 6);
    h->display();
    delete h;
}
void main(){
    while(1) g();

}
```



(Acumulare de resurse)---

delete[] vs. delete

```
char* p1 = new char[100];
char* p2 = malloc(100);
// p1 and p2 both point to 100 charactors of allocated memory. Those calls were functionally idenctical in that sense.
delete [] p1;
free(p2);
// Both of those calls deleted the 100 chars of ram allocated; again they are functionally identical.
```

Here's where they'**re different:**

```
class AThingy
{
public:
AThingy() { printf("Constructor\n"); }
~AThingy() { printf("Destructor\n"); }
};
AThingy *t1 = new AThingy[100];
AThingy *t2 = (AThingy*)malloc( sizeof(AThingy) * 100 );
// now t1 and t2 both point to 100 thingys. However, the new [] call called the constructor of each of the 100 thingys, but malloc did not!
delete [] t1;
free(t2);
// the delete [] called the destructor for all 100 items, the free did not!
// so malloc-->new, free-->delete.
```

Gestionare memoriei prin metode “destructor”

- C++ oferă mecanisme de eliberare automată a resurselor, prin metode ***destructor.***
- Dacă în clasa Person se adaugă metoda destructor următoare:

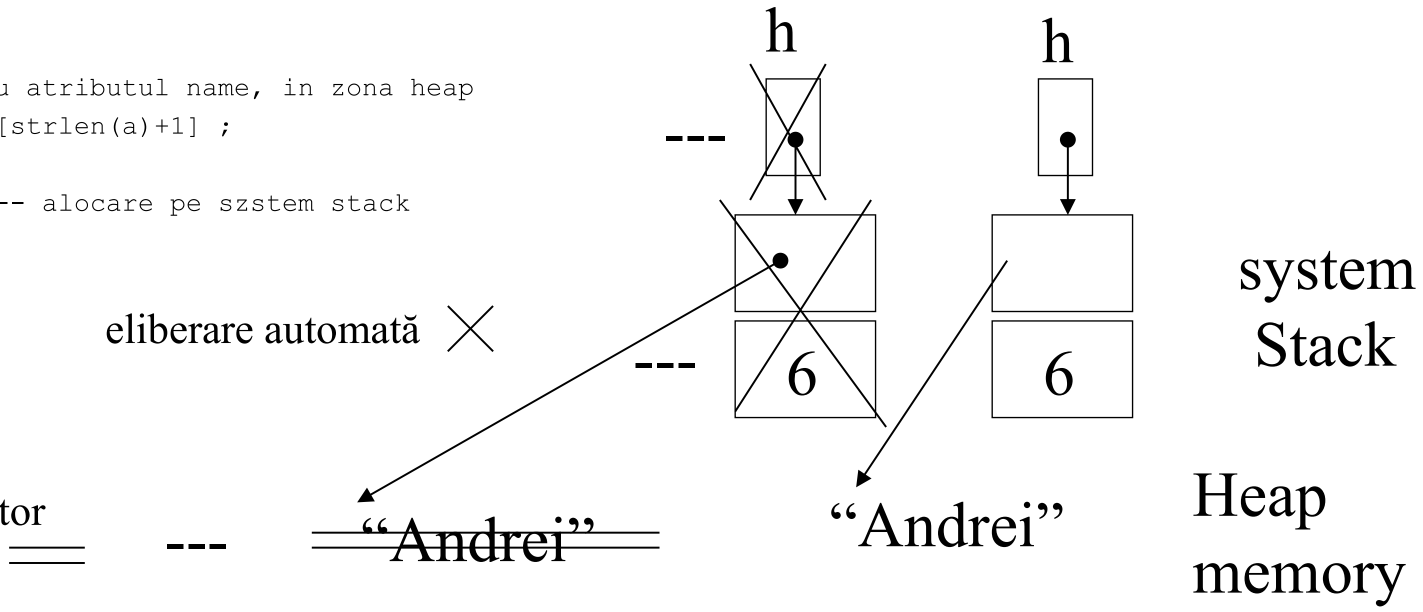
```
Person:: ~Person(){delete[] name;}
```

aceasta va fi activată de fiecare dată, înainte de eliberarea memoriei alocate pentru atributele obiectelor Person în zona stack (adică la ieşirea din blocul în care obiectul a fost creat)

- Sistemul de gestiune automată a memoriei poate fi prin urmare completat cu metode de eliberare a resurselor alocate obiectelor.

topdown\varianta POO\Destructor\ANDUR4.CPP

```
//file andur4.cpp si clasa Person are destructor
#include "Person.hpp"
#include <string.h>
char *a="Andrei";
void g(){
     // valoare pentru atributul name, in zona heap
     char *n=new char[strlen(a)+1] ;
     strcpy(n,a);
     Person h(n, 6); -- alocare pe szstem stack
     h.display();
}
void main(){
    while(1) g();
}
```



Extensibilitate, reutilizabilitate (modularitate)

- Extendibility: the easy of adapting software products to changes of specification [Meyer]
- Reusability: the ability of softwatre elements to serve for the construction of many different applications
- Modularity: the term covers extendibility and reusability.
- Exemplul 3. (enunţul următor este similar celui din Exemplul 2)

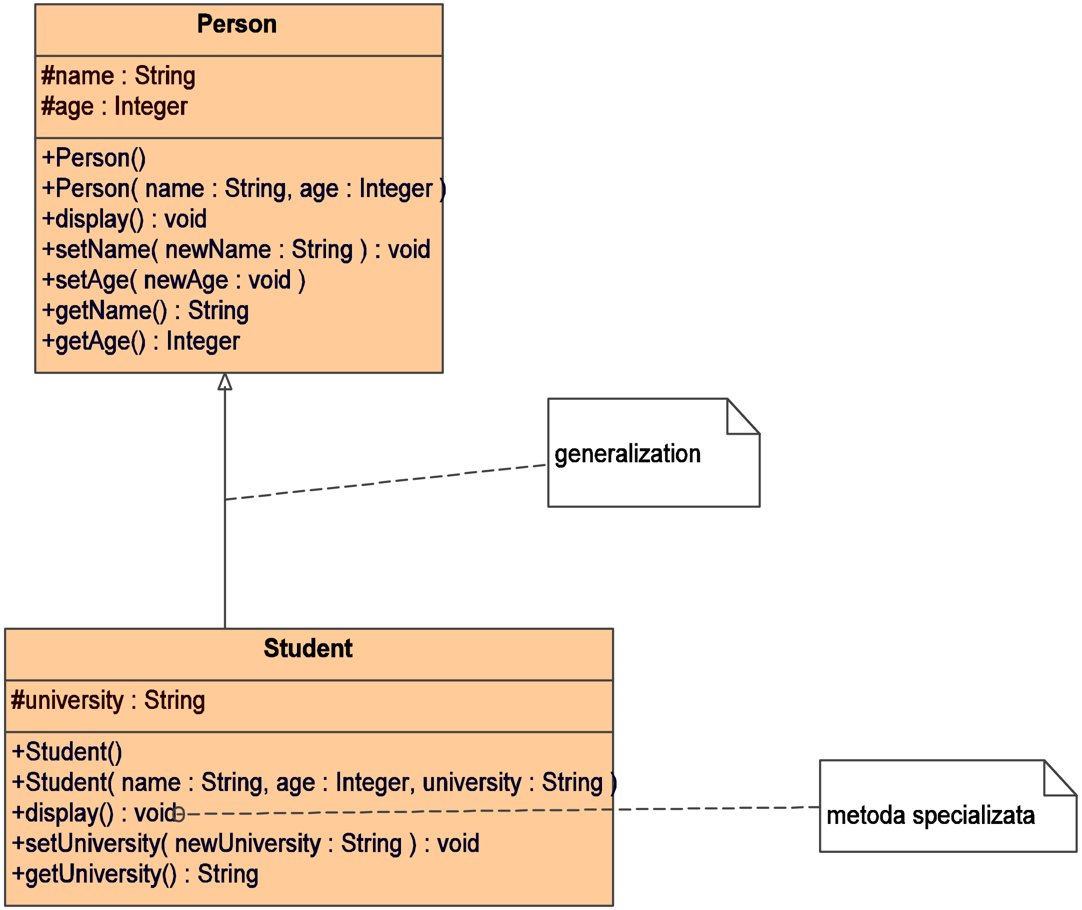
  *Să se scrie un program cu ajutorul căruia să se introducă şi să se afişeze informaţii despre **studenţii** dintr-o **universitate.** Fiecare **student** este identificat prin nume, vârstă şi **universitatea** la care învaţă. Tipul de operaţie care se execută la un moment dat (introducere sau afişare) este specificat interactiv de utilizator.*

- Similitudinea celor două enunţuri conduce la ideea de extensibilitate şi reutilizabilitate.

- Soluţia oferită de OOP este proiectarea unei alte clase care:
  - să moştenească toate caracteristicile clasei Person,
  - să adauge caracteristici noi şi
  - să specializeze unele din caracteristicile moştenite, pentru a corespunde noului context de utilizare.

Relaţia de moştenire (de generalizare; de specializare)

- Relaţia de moştenire(de generalizare; de specializare), caz special de relaţie client-server
- Clasa client Student moşteneşte toate caracteristicile clasei server Person (după cum se observă, nu mai sunt repetate în specificare)
- Adaugă noi caracteristici (sunt specificate explicit)
- Specializează metoda moştenită `void display()` (signatura metodei din specificarea clasei Person este *copiată* în specificarea clasei Student)



Person

#name : String
#age : Integer

+Person()
+Person(name : String, age : Integer)
+display() : void
+setName(newName : String) : void
+setAge(newAge : void)
+getName() : String
+getAge() : Integer

generalization

Student

#university : String

+Student()
+Student(name : String, age : Integer, university : String)
+display() : void
+setUniversity(newUniversity : String) : void
+getUniversity() : String

metoda specializata

Student, o specializare a clasei Person

topdown\varianta POO\STUDENT.HPP

```
   //file Student.hpp
#ifndef STUDENT_HPP
#define STUDENT_HPP
#include "Person.hpp"
#include <iostream.h>
class Student: public Person{ // specializeaza clasa Person
public: //interfata
   Student();
   Student(char *name, int age, char *university);
   void display(); // specializare a metodei Person::display()
   // caracteristici adaugate:
   char* getUniversity();
   void setUniversity();
protected:
   char *university;
};
#endif
```

topdown\varianta POO\STUDENT.CPP

```
//file Student.cpp
#include "Person.hpp"
#include "Student.hpp"
#include <iostream.h>

   Student::Student()
            :Person(){  // lista de initializare
            university="Universitatea din Pitesti";
   }
   Student::Student(char *name, int age, char *university)
            :Person(name, age){// lista de initializare
            this->university= university;
   }
   void Student::display(){
            Person::display();
            cout<<endl<< university <<endl;
   }
   char* Student::getUniversity(){return university;}
   void  Student::setUniversity(){
            this->university=university;
   }
```

Namespaces (C++)

http://msdn.microsoft.com/en-us/library/5cb46ksf.aspx

- The C++ language provides a single global namespace. This can cause problems with global name clashes.
  For instance, two header files, each defining a class with the same name.
- A namespace is a declarative region that attaches an additional identifier to any names declared inside it.

```
namespace one {
   class String { ... };
}
namespace two {
   class String { ... };
}
```

- The using directive allows the names in a **namespace** to be used without the *name*
  using namespace <namespace-name>

`std` namespace, using directive vs. using declaration

- The ***`std`*** namespace
  The ANSI/ISO C++ standard requires you to explicitly declare the namespace in the standard library.

when using iostream, you must specify the namespace of cout in one of the following ways:

```
#include <iostream>
using std::cout; // using declaration
void f(){ cout<<endl;}//error, endl undefined

void main() {
        std::cout << "Hello "; //explicit qualification
        using namespace std; //using directive
        cout << "World." << endl;
} // the efect of  using namespace std; ends here!
int g(){
        cout<<endl;}//error, endl undefined
        //using std::cout declaration still visible!
```

Note the difference between the using **directive** and the using **declaration**;
•the using directive allows **all the names** in a namespace to be used without qualification.
•the using declaration allows an **individual name** to be used without qualification,

Namespace variables vs. Global or Local variable

```
namespace one{
   int k=1,j=1;
}
using namespace one;
int k=2;
int main(){
   int j=2;
   //k=1; // error, ambiguous symbol
   j=3; // local variable
   one::j=4; // namespace variable
}
```

It is an **error** to have a **namespace variable** with the same name as a **global variable**.

If a **local variable** has the same name as a namespace variable, the **namespace variable is hidden**

More on **using declaration(1/)**

- When used to declare a member, a using declaration must refer to a member of a base class.

```
class C {
 public: int   g();
};
```

```
class D2 : public B {
 public: using B::f; //  ok: B is a base of D2
 // using C::g; // error: C isn't a base of D2 };
```

- Members declared with a using declaration can be referenced using explicit qualification.
- The :: prefix refers to the global namespace.

```
void f() { printf_s("In f\n"); }
namespace A {
 void g() {
   printf_s("In A::g\n");
 }
}
namespace X {
  using ::f; // global f using
  A::g; // A's g
}
```

```
void h() {
  printf_s("In h\n");
  X::f(); // calls ::f
  X::g(); // calls A::g
}
int main() { h(); }
```

Results:
In h
In f
In A::g

More on **using declaration(2/)**

- When a using declaration is made, the synonym created by the declaration refers only to definitions that are valid at the point of the using declaration. Definitions added to a namespace after the using declaration are not valid synonyms.
- A name defined by a using declaration is an alias for its original name. It does not affect the type, linkage or other attributes of the original declaration.

```
namespace A {
  void f(int) {}
}
using A::f;
// f is a synonym for
// A::f(int) only
namespace A {
 void f(char) {}
}
```

```
void f() {
 f('a'); // refers to A::f(int),
 //even though A::f(char) exists
}
void b() {
  using A::f;
 // refers to A::f(int) AND  A::f(char)
  f('a'); // calls A::f(char);
}
```

More on **using declaration(3/)**

- With respect to functions in namespaces, if a set of local declarations and using declarations for a single name are given in a declarative region, they must all refer to the same entity, or they must all refer to functions.

```
namespace B {
 int i;
 void f(int);
 void f(double);
}
```

```
void g() {
  int i;
  using B::i; // error: i declared twice
  void f(char);
  using B::f; // ok: each f is a function
}
```

Relaţii speciale de tip client-server

Mostenirea** şi **agregarea

Observaţie:

- *Relaţia "obiectul D **este un** B" se implementează prin **moştenire;***

```
class D:public B{...}
```

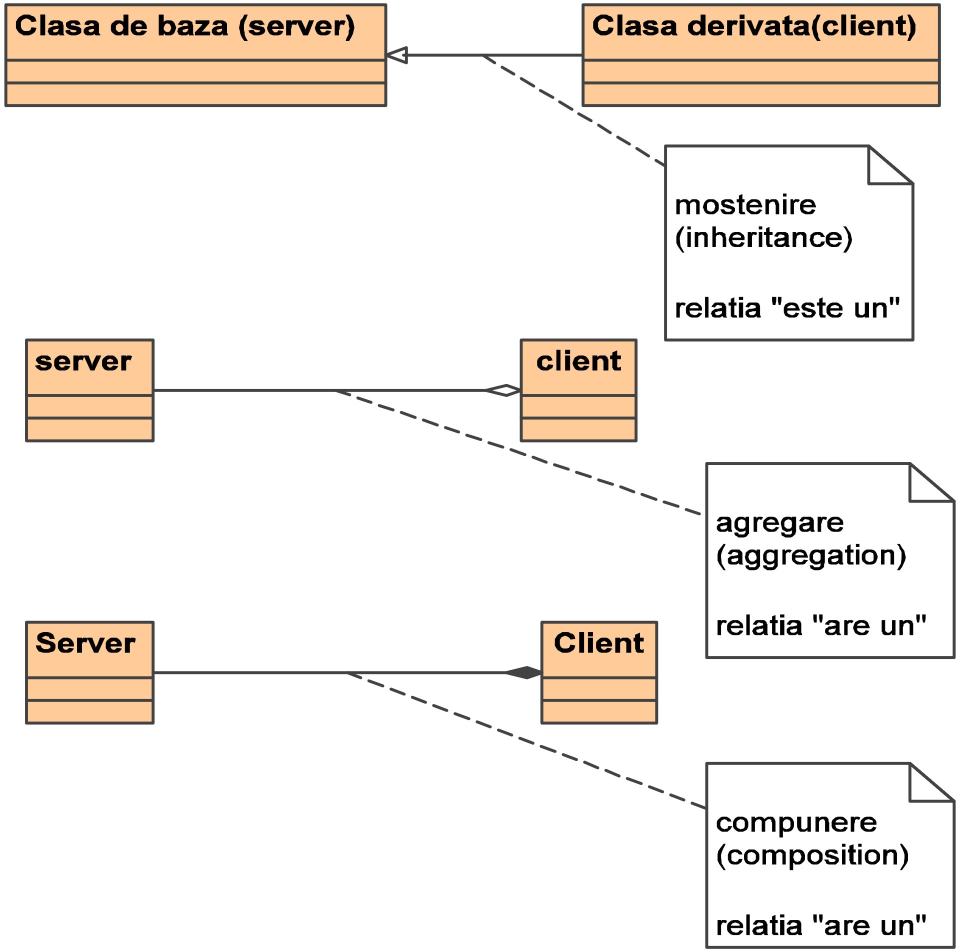
- *relaţia "obiectul D **are un** B" se implementează prin*
  - *agregare*

```
class D{    B *server;
                 server=new
        B();
…
};
```

  - sau ***compunere***

```
class D{    B server;
…
};
```

- *dacă există dubii, este preferată relaţia de moştenire*



Agregare-Compunere: reutilizare prin cumpărare de servicii

- Reutilizarea de către un client C a serviciilor unui server S poate fi făcută prin:
  - **distribuirea** serviciilor cumpărate
    - cost de proiectare redus pentru C, care oferă acces direct propriilor clienţi la serviciile lui S
    - incomodă pentru clienţii lui C, care trebuie să cunoască atât serviciile lui C cât şi pe cele ale lui S
  - **asumarea** serviciilor cumpărate
    - cost de proiectare mai mare pentru C, care
      - blochează accesul direct propriilor clienţi la serviciile lui S
      - furnizează servicii similare, exploatând serverul S
    - utilă pentru clienţii lui C, care ignoră detaliile lui C (serverul S)

Exemplu: Circle, Sphere

- Serverul Circle
  Circle\CIRCLE.HPP

```
// file Circle.hpp
#define PI 3.14
class Circle{
public:
   Circle(float radius=1.0):radius(radius){}
   virtual float area(){return PI*radius*radius;}
   virtual float length(){return 2*PI*radius;}
   virtual float getRadius(){return radius;}
   virtual void  setRadius(float radius){this->radius=radius;}
protected:
   float radius;
};
```

- Circle\CIRCDRV.CPP

```
// file Circdrv.cpp, test driver pentru serverul Circle
#include "Circle.hpp"
#include <iostream.h>
void main(){
   Circle c(1.0);
   cout<< c. getRadius() <<endl << c. length() <<endl c. area()   <<endl;
   c.setRadius(2* c.getRadius());
   cout<< c. getRadius() <<endl;  // de doua ori mai mare
   cout<< c. length() <<endl;  // de doua ori mai mare
   cout<< c. area()   <<endl;  // de 4 ori mai mare
}
```

Relaţia “ is a (kind of)” : sfera e (un fel de) cerc

- Sphere moşteneşte şi specializează caracteristicile serverului Circle
  - metoda `area()` este specializată
  - Unele din metodele moştenite `length()` nu mai au sens: lungimea sferei?

```
Circle\SPHEREIS.HPP
// file Sphereis.hpp, specializeaza Circle
// Sphere is a "kind " of Circle
#include "Circle.hpp"
class Sphere:public Circle{
public:
   Sphere(float radius):Circle(radius){}
   // specializare
   virtual float area(){return 4*PI*radius*radius;}

    // metoda proprie
   virtual float volume(){return
   (4/3.0)*PI*radius*radius*radius;}
};
```

Test driver pentru Sphere

```
Circle\SPHISDRV.CPP
// file Sphisdrv.coo, test driver pentru Sphere
#include "Sphereis.hpp"
#include <iostream.h>
void main(){
   Sphere s(1.0);
   cout<< s. getRadius() <<endl; // metoda mostenita, cu sens
   cout<< s. length() <<endl;         // metoda mostenita, fara sens:
                                      // lungimea sferei?
   cout<< s. area()   <<endl;    // aria sferei, metoda specialiyata

   // urmeaza aria cercului mare
   // sfera e un "fel" de cerc, deci conversie
   cout<<((Circle)s).area()   <<endl;
   // sau
   cout<<s.Circle:: area()   <<endl;
}
```

Ascunderea serviciilor distribuite care isi pierd sensul

- Cazul length(), pentru Sphere Circle\ascund\SPHEREIS.HPP

```
// file Sphereis.hpp, specializeaza Circle
// cu ascunderea unor servicii care isi pierd sensul
// Sphere is a "kind " of Circle
#include "k:\POO_CU~1\Circle\Circle.hpp"
class Sphere:protected Circle{ // nu public!
public:
    Sphere(float radius):Circle(radius){}
    virtual float area(){return 4*PI*radius*radius;}// specializare
        // metoda proprie
    virtual float volume(){return (4/3.0)*PI*radius*radius*radius;}
    virtual float circleLength(){return Circle::length();}
    virtual float circleArea(){return Circle::area();}
public:
    Circle::getRadius;
    Circle::setRadius;
};
Circle\ascund\SPHISDRV.CPP
```

```
// file Sphisdrv.coo, test driver pentru Sphere
#include "Sphereis.hpp"
#include <iostream.h>
void main(){
    Sphere s(1.0);
    cout<< s. getRadius() <<endl; // metoda mostenita, cu sens
    //cout<< s. length() <<endl;   //metoda mostenita, fara sens, acum inaccesibila: lungimea sferei?
    cout<< s. area()   <<endl;    // aria sferei

    // urmeaza aria cercului mare
    cout<<s.circleArea()   <<endl;
    //lungimea cercului mare
    cout<<s.circleLength()<<endl;

}
```

Relaţia “has a”: Sphere “are un” Circle cu distribuirea serviviilor

- Distribuirea serviciilor cumpărate (varianta cu serverul Circle agregat)

Circle\distrib\SPHEREDS.HPP

```
// file Sphereds.hpp, agregare Circle si distribuire servicii
// Sphere has a Circle, ale careui servicii le face publice.
#include "k:\POO_CU~1\Circle\Circle.hpp"
class Sphere{
public:
   Sphere(float radius){pc= new Circle(radius);}
   // area() este metoda proprie
   //(nu este specializare a metodei din Circle!)
   virtual float area(){
   // atentie, pc->radius inaccesibil!
   return 4*PI*pc->getRadius()*pc->getRadius();
   }
   // metoda proprie volume()
   virtual float volume(){
   return (4/3.0)*PI*pc->getRadius()*pc->getRadius()*pc->getRadius();
   }
public:
   Circle *pc;// cerc mare al sferei, agregat in Sphere

};
```

Test driver, distribuire, agregare

- Circle\distrib\SPHDSDRV.CPP

```
// file Sphisdrv.coo, test driver pentru Sphere
#include "Sphereds.hpp"
#include <iostream.h>
void main(){
    Sphere s(1.0);
    // serviciul getRadius() distribuit
    // atentie: s.getRadius(): error, not a member of Sphere
    cout<< s. pc->getRadius() <<endl;
    //cout<< s.length() <<endl;   //error, not a member

    cout<< s. area()   <<endl;       // aria sferei,

    // urmeaza aria cercului mare
    cout<< s. pc->area() <<endl; // serviciu distribuit

    cout<< s. pc->length() <<endl; // are sens, este lungimea cercului mare!
/*
    // sfera NU mai e un "fel" de cerc, deci
    cout<<((Circle)s).area()   <<endl;// could not find a match for
    'Circle::Circle(Sphere)'
    // sau
    cout<<s.Circle:: area()   <<endl; // 'Circle' is not a public base class of
    'Sphere'
*/
}
```

Relaţia “has a”: Sphere “are un” Circle cu asumarea serviviilor

- Asumarea serviciilor cumpărate (si ilustrare relaţia de compunere)

Circle\asumare\SPHEREAS.HPP

```
/ file Sphereas.hpp,  asumare servicii si relatia de compunere
// Sphere has a Circle (composition), ale carui servicii le asuma.
#include "k:\POO_CU~1\Circle\Circle.hpp"
class Sphere{
public:
     Sphere(float radius){c=Circle(radius);}
     //servicii asumate
     virtual float getRadius(){return c.getRadius();}
     virtual void setRadius(float radius){c.setRadius(radius);}
     virtual float circleLength(){return c.length();}
     virtual float circleArea(){return c.area();}
     // area() este metoda proprie, aria sferei
     //(nu este specializare a metodei din Circle!)
     virtual float area(){
                 // atentie, c.radius inaccesibil!
                 return 4*PI*c.getRadius()*c.getRadius();
     }
     // metoda proprie volume()
     virtual float volume(){
                 return (4/3.0)*PI*c.getRadius()*c.getRadius()*c.getRadius();
     }
private:// serverul este inaccesibil clientilor Sphere
     Circle c;// cerc mare al sferei,  in Sphere (composition)

};
```

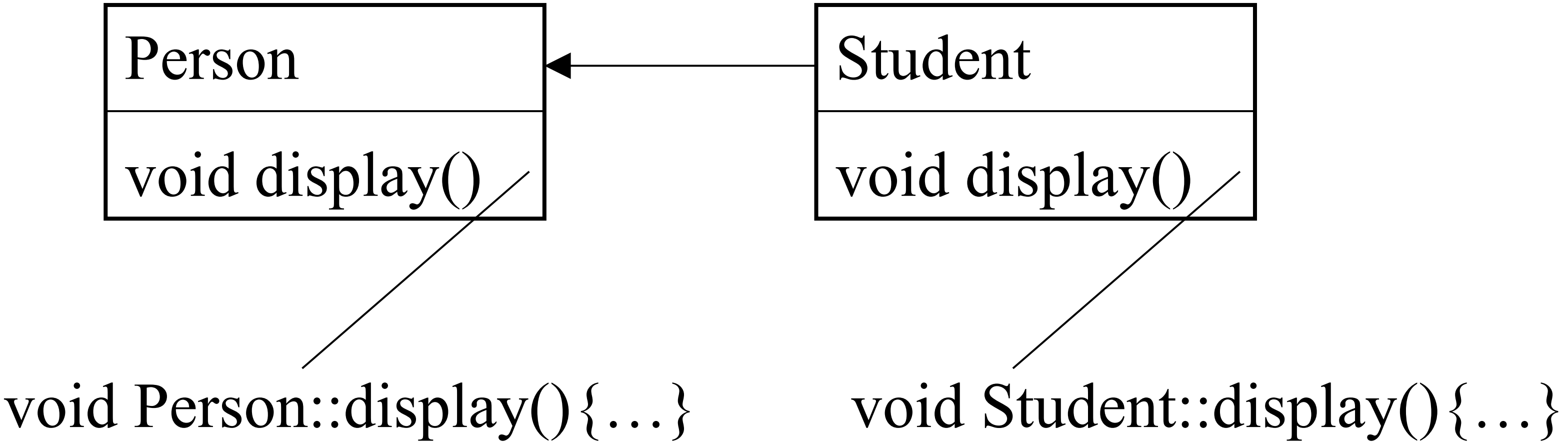
Test driver, asumare, compunere

- Circle\asumare\SPHASDRV.CPP

```
// file Sphasdrv.cpp, test driver pentru Sphere
#include "Sphereas.hpp"
#include <iostream.h>
void main(){
   Sphere s(1.0);
   // serviciul getRadius(), asumat
   cout<< s.getRadius() <<endl;
   s.setRadius(2.0);
   //cout<< s.length() <<endl;   //error, not a member
   cout<< s.circleLength()<<endl; // lungimea cercului
   mare
   cout<< s.circleArea()<<endl; // aria cercului mare
   cout<< s. area()   <<endl;       // aria sferei,
}
```

Programe cu caracter general; polimorfism

- Metodologia OOP permite scrierea unor programe cu caracter general: create pentru a prelucra un anumit tip de date T, acestea pot prelucra date din orice subtip S al lui T.
- Noţiuni fundamentale:
  - legare dinamică (late binding) a metodelor
  - expresii (semantic) polimorfice



Legare statică, legare dinamică

```
char c;
Person *p;
cin>>c;
if (c==‘P’) p= new Person(“Tudor”, 58);
else p=new Student(“Stefan”, 23, “UPIT”);

p->display();               expresie de transmitere mesaj
```

Răspunsul la mesaj depinde de modul de legare a metodei – mesaj la una din implementările din ierarhie:

- legare statică (implicită în C++): `Person::display()`, indiferent de valoarea lui c.
  Implementarea este determinată de tipul declarat al ***expresiei de destinaţie*** (p în cazul considerat, al carei tip declarat este `Person`)
- legare dinamică (specificată în C++ prin cuvântul cheie `virtual`):
  - `Person::display()`, dacă `c==‘P’`
  - `Student::display()`, în caz contrar.
    Implementarea este determinată de tipul ***obiectului de destinaţie*** (acesta poate fi de tip `Person`, `Student` sau orice alt subtip al acestora)
- Expresia `p->display()` are mai multe semnificaţii în cazul legării dinamice (expresie semantic polimorfă). Fenomenul se numeşte ***polimorfism*** indus de relaţia de de subtip.

Test driver pentru polimorfism

Observaţie. Pentru ca expresia `p->display();` din exemplul de mai jos să fie polimorfă, modificaţi fişierele Person.hpp şi Student.hpp, astfel ca în specificarea celor două clase metoda void display() să fie legată dinamic (late binding). Prin urmare, în loc de `void display()` va apărea:

```
virtual void display()
```

topdown\varianta POO\POLIMORF.CPP

```
//file polimorf.cpp
//test driver
#include "Person.hpp"
#include "Student.hpp"
#include <iostream.h>
void main(){
   Person *p;
   char c;
   cin>>c;
   if (c=='P') p=new Person("Tudor",58);
   else p= new Student("Stefan",23, "Universitatea din Pitesti");
   p->display();
}
```

`Rezultate (presupunem ca se introduce caracterul X):`

- `Legare statica:`
  ```
  Name: Stefan Age: 23
  ```
- `Legare dinamica:`
  ```
  Name: Stefan Age: 23
  Universitatea din Pitesti
  ```

Exemplu: funcţie generală

- Exemplu. Funcţie de afişare, în format special, a datelor despre obiecte Person; exploatând fenomenul de polimorfism semantic, poate fi folosită pentru orice subtip al clasei Person (precum Student sau orice altă clasă derivată din Person)
  topdown\varianta POO\general.hpp

```
//file general.hpp
#include  "Person.hpp"
void display(const Person *p); // functie, nu metoda; C++, limbaj "hibrid"
topdown\varianta POO\generala.cpp
//file generala.cpp
#include <iostream.h>
#include  "Person.hpp"
void display(const Person *p){ // functie, nu metoda; C++, limbaj "hibrid"
     cout<<endl<<"______________"<<endl;
     p->display(); // expresie polimorfa, presupunem virtual void display()
     cout<<endl<<"______________"<<endl;
}
```

Test driver pentru funcţia generală

`topdown\varianta POO\testgen.cpp`

```
//file testgen.cpp
#include "general.hpp"
#include "Person.hpp"
#include "Student.hpp"
void main(){
   Person *p;
   p=new Person("Tudor", 58);
   display(p);  // functie generala
   p=new Student("Stefan",23, "Universitatea din Pitesti");
   display(p);
}
```

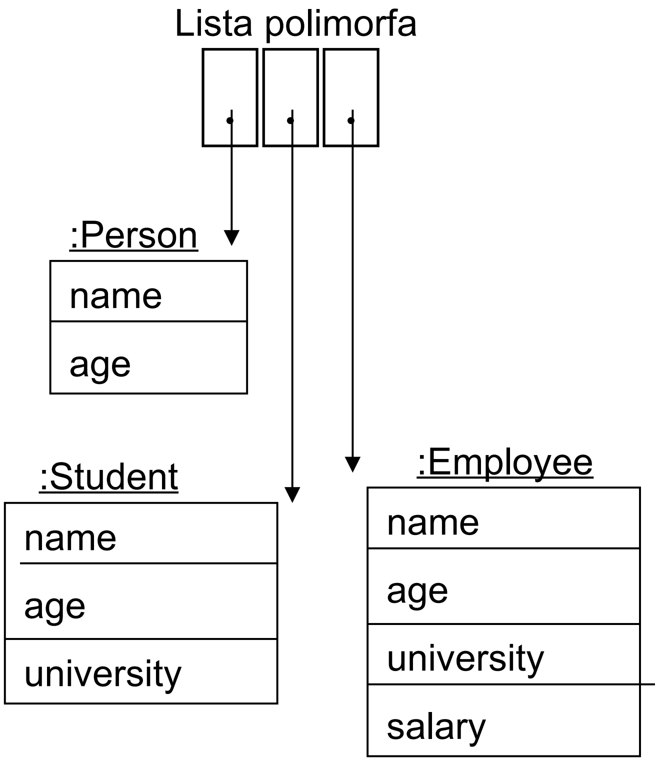
- Observaţie. Funcţia `void display(const Person *p)` poate fi utilizată pentru orice subtip al tipului Person.
- Această facilitate are aplicabilitate la scrierea unor programe ce ***pot prelucra tipuri de date care sunt definite ulterior*** momentului în care a fost scris programul.

Liste polimorfe

- Vector polimorf: elementele din componenţa sa pot avea tipuri diferite

Lista polimorfa\POLIMORF.CPP.

```
// file polimorf.cpp

#include "Person.hpp‟ Lista polimorfa\PERSON.HPP
#include "Student.hpp‟ Lista polimorfa\STUDENT.HPP
#include "Employee.hpp‟ Lista polimorfa\EMPLOYEE.HPP
#include <iostream.h>
#define N 3

Person* listaPolimorfa[N]; //lista polimorfa
void main(){
    //citire lista polimorfa
    int i;
    for( i=0;i<N;i++){
                char c;
                cout<<endl<<"Elementul listei? (P, S sau E):"; cin>>c;
                switch(c){
                    case 'P':listaPolimorfa[i]=new Person();
                                break;
                    case 'S':listaPolimorfa[i]=new Student();
                            break;
                    case 'E':listaPolimorfa[i]=new Employee();
                            break;
                }

                listaPolimorfa[i]->read(); // expresie polimorfa!
    }
    // afisare lista polimorfa
    for( i=0;i<N;i++){
                listaPolimorfa[i]->display(); // expresie polimorfa!
    }
}
```



Lista polimorfa

:Person

name

age

:Student

name

age

university

:Employee

name

age

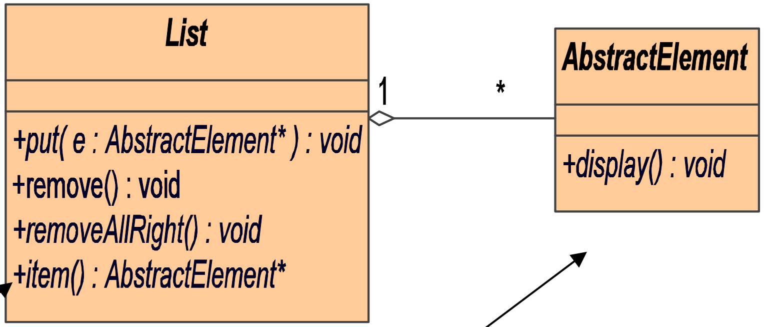
university

salary

Lista polimorfă (cu cursor)

Lista polimorfa\Cursor\LIST.HPP

```
//file List.hpp
#ifndef LIST_HPP
#define LIST_HPP
#define Boolean int
class AbstractElement{
public:
     virtual void display()=0;
};

class List{
public:
     virtual void put(AbstractElement*)=0;
     virtual Boolean Empty()=0;
     virtual Boolean Full()=0;
     virtual Boolean isFirst()=0;
     virtual Boolean isLast()=0;
     virtual Boolean before()=0;
     virtual Boolean after()=0;
     virtual AbstractElement* item()=0;
     virtual void back()=0;
     virtual void forth()=0;
     virtual void removeAllRight()=0;
     virual void remove(){};
};
#endif
```



List
1
*
AbstractElement
+put(e : AbstractElement) : void*
+remove() : void
+removeAllRight() : void
*+item() : AbstractElement**
+display() : void

metodă concretă

metode abstracte

clase abstracte

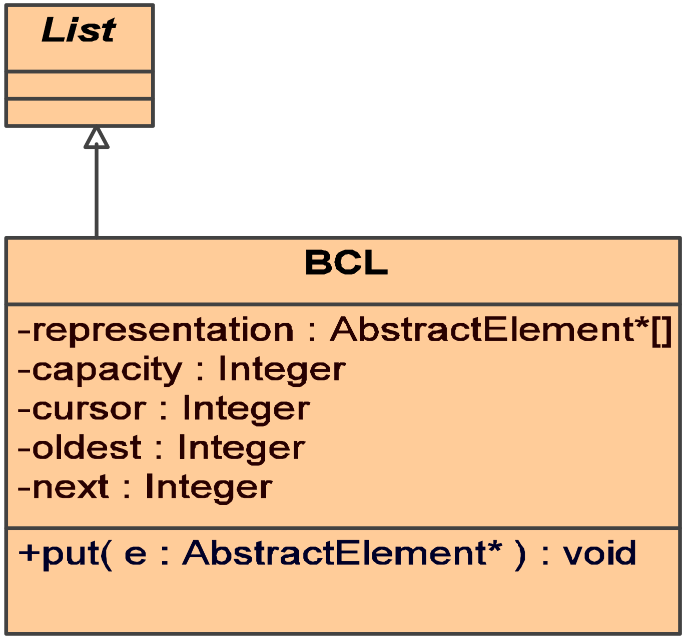
- **metodă concretă în clasă abstractă;**
- este implementată printr-o acţiune de efect nul
- în subclasele concrete nu este obligatorie specializarea metodei remove();
- justificare: implementarea acestei metode se face de regulă ineficient (în cazul reprezentării prin vectori)
- dacă într-o aplicaţie este acceptabilă o operaţie de ştergere care de fapt nu are nici un efect, atunci aceasta corespunde implementării din clasa abstractă.

Lista polimorfa, pentru istoric de acţiuni (specificare)

Lista polimorfa\Cursor\ISTORIC.HPP

```
// file istoric.hpp
#ifndef ISTORIC_CPP
#define ISTORIC_CPP
#include "List.hpp"
class BCL:public List{
public:
     BCL(int capacity=1);
     virtual void put(AbstractElement*);
     virtual Boolean Empty();
     virtual Boolean Full();
     virtual Boolean isFirst();
     virtual Boolean isLast();
     virtual Boolean before();
     virtual Boolean after();
     virtual AbstractElement* item();
     virtual void back();
     virtual void forth();
     virtual void removeAllRight();
private:
     int capacity;
        int remembered; //capacity-1
     int cursor;
     int oldest;
     int next;
     AbstractElement **representation;
};
#endif
```

`remove()`,
metodă moştenită din List



List

BCL

-representation : AbstractElement*[]
-capacity : Integer
-cursor : Integer
-oldest : Integer
-next : Integer

+put(e : AbstractElement*) : void

Lista polimorfa, pentru istoric de acţiuni (implementare)

Lista polimorfa\Cursor\ISTORIC.CPP

```
// file istoric.cpp
#include "istoric.hpp"
BCL::BCL(int capacity):capacity(capacity){
                        representation=new AbstractElement*[capacity];
                        remembered=capacity-1;
                        oldest=0;
                        next=0; //oldest=next <=> empty
                        cursor=capacity-1;

}

void BCL::put(AbstractElement* e){
        representation[next]=e;
        cursor=next;
        next=(next+1)%capacity;
        // inainte de primul element, totdeauna un loc gol!
        // deci next==oldest numai cand este emptyu
        if (next==oldest) oldest=(oldest+1)%capacity;
}
```

Continuare implementare

```
Boolean BCL::Empty(){return(oldest==next);
}
Boolean BCL::Full(){
    return((next+1)%capacity==oldest);
}
Boolean BCL::isFirst(){return (cursor==oldest);}
Boolean BCL::isLast(){//next imediat dupa cursor
    return ((cursor+1)%capacity== next);
    // pentru  cazul cursor=capacity-1 & next=0
}
Boolean BCL::before(){// oldest imediat dupa cursor
    return ((cursor+1)%capacity==oldest);
    // pentru  cazul cursor=capacity-1 & oldest=0
}

Boolean BCL::after(){// cursor dupa ultimul, adica egal cu next
    return (cursor==next);
}
AbstractElement* BCL::item(){
    return representation[cursor];
}
void BCL::back(){ cursor=(cursor-1 + capacity)%capacity; }
void BCL::forth(){cursor=(cursor+1)%capacity;}
void BCL::removeAllRight(){next= (cursor+1)%capacity;}
```

Polimorfism parametric (generic) (1/5)

- Funcţii generice
  - Există cazuri de funcţii ce au signaturi şi implementări care diferă doar prin tipul datelor prelucrate. Generic\SWAP.CPP

```
typedef struct{
    float x,y;
}Complex;
void swap(int &x, int &y) {
    int t=x;
    x=y; y=t;
}
void swap(Complex &x, Complex &y) {
    Complex t=x;
    x=y; y=t;
}
void main(){
    int x=0, y=1;
    float f=0, g=1;
    Complex c={1,2}, d={10,20};
    swap(x,y); //interschimba x cu y
    swap(c,d); //interschimba c cu d
    swap(f,g); // NU interschibma!
}
```

De ce?

Polimorfism parametric (generic) (2/5)

- În astfel de cazuri se pot construi macroinstrucţiuni în care tipul de date apare ca parametru.
  - Ele servesc drept şabloane (templates) pentru generarea unor definiţii de funcţii, prin precizarea tipului de date
  - Generarea este o fază ce precede compilarea

Generic\SWAP.TPL   Generic\SWAPDRV.CPP

```
// file swap.tpl
template <class Type>
void swap(Type &x, Type &y) {
          Type t=x;
          x=y;
          y=t;
}
```

macroinstructiunea

```
template <class T>
void swap(Type, Type )
```

este "instantiata" (expandata) in 3 variante

```
// file instswap.cpp
#include "swap.tpl"
typedef struct{
          float x;
          float y;
}Complex;
void main(){
          int x=0, y=1;
          float f=0, g=1;
          Complex c={1,2}, d={10,20};
          swap(x,y); //interschimba x cu y
          swap(c,d); //interschimba c cu d
          swap(f,g); // acum  interschibma!
}
```

Polimorfism parametric (generic) (3/5)
Clase parametrizate (generice)

- Reluăm exemplul listei cu cursor, implementată printr-un vector circular. Tipul elementelor din lista este un parametru al clasei BCL (prescurtat BCL). Generic\Cursor\ISTORIC.TPL

```
// file istoric.tpl
#ifndef ISTORIC_TPL
#define ISTORIC_TPL
//specificare
// Atentie, specificarea si
// implemenatrea in acelasi fisier!
#define Boolean int
template <class TypeOfElement>
class BCL{
public:
BCL(int capacity=1);
          virtual void put(TypeOfElement);
          virtual Boolean Empty();
          virtual Boolean Full();
          virtual Boolean isFirst();
          virtual Boolean isLast();
          virtual Boolean before();
          virtual Boolean after();
          virtual TypeOfElement item();
          virtual void back();
          virtual void forth();
          virtual void removeAllRight();
```

```
private:
          int capacity;
        int remembered; //capacity-1
          int cursor;
          int oldest;
          int next;
          TypeOfElement *representation;
};
```

Vector de elemente, nu de referinţe, pentru a facilita legarea statică, atunci când se doreşte.
Totuşi, poate fi vector de referinţe, dacă Type Of Element este un tip de referinţă, precum T*

Polimorfism parametric (generic) (4/5)
Clase parametrizate (generice), implementare

```
 // implementare, schiţă
template<class TypeOfElement>
BCL<TypeOfElement>::BCL(int capacity):capacity(capacity){
                    // aceeasi implementare din cayul listelor eterogene
}
template<class TypeOfElement>
void BCL<TypeOfElement>::put(TypeOfElement e){
                    // aceeasi implementare din cayul listelor eterogene
}
// and so on…

#endif
```

Metodele sunt parametrizate

Numele clasei arata ca un nume expandat

Polimorfism parametric (generic) (5/5) Aplicaţie

- Clasa BCL este utilizată pentru a crea liste de mai multe tipuri.

Generic\Cursor\DRIVER.CPP

```
//  file driver.cpp
#include <iostream.h>
#include "istoric.tpl"
#include "person.hpp"
#include "student.hpp"
void main(){
    BCL<Person*> *listHeterogen=
              new BoundedCircularList<Person*>(3);
    BCL<Person> *listOmogen=
              new BoundedCircularList<Person>(3);
    listHeterogen->put(new Person("Tudor",58));
    listHeterogen->put(new Student("Stefan",23, "UPit"));
    listOmogen->put(*(new Person("Tudor",58)));
    listOmogen->put(*(new Student("Stefan",23,"UPit")));
    listHeterogen->put(new Person("Tudor",5));
    listHeterogen->put(new Student("Stefan",23,"UPit"));
    listOmogen->item().display();
    listOmogen->back();
    listOmogen->item().display();
    listHeterogen->item()->display();
    listHeterogen->back();
    listHeterogen->item()->display();
    //and so on
}
```

Liste BCL; tipurile elementelor sunt diferite

Lista **eterogenă** cu elemente de *tip referinţă* `Person*`

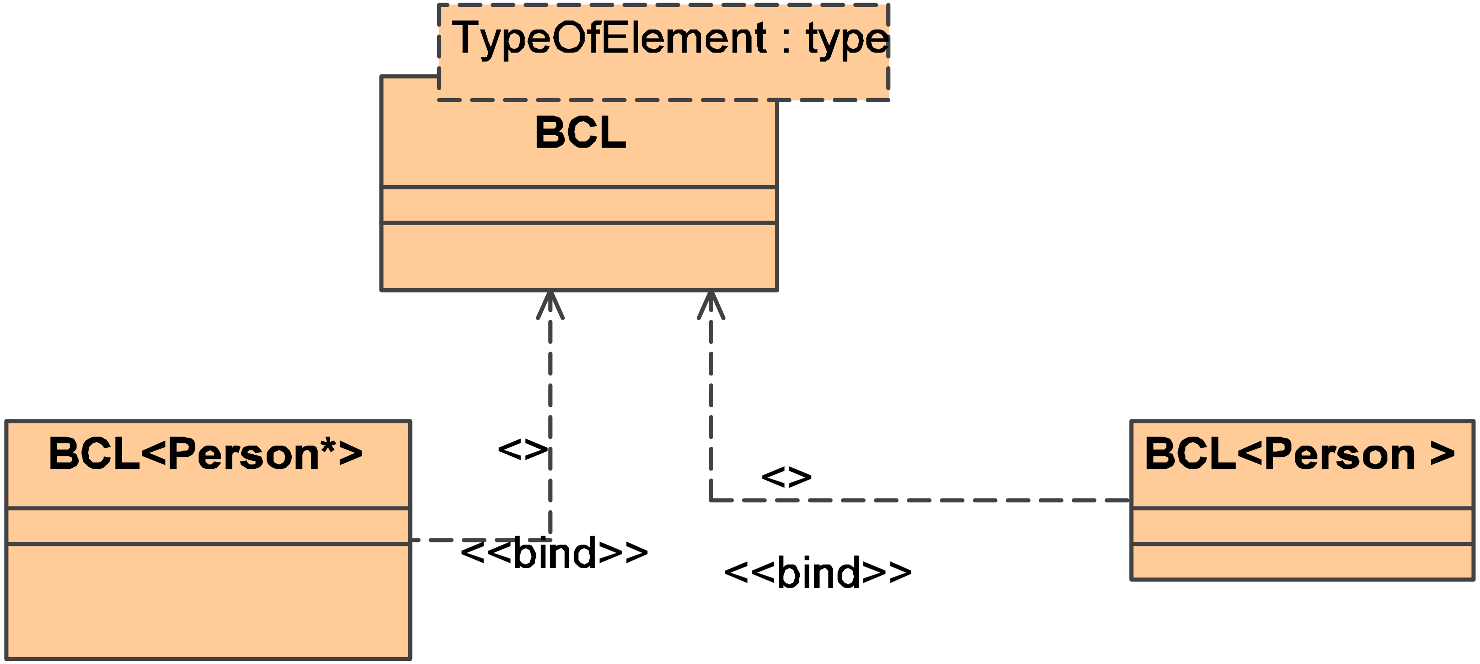
Lista **omogenă** cu elemente de tip `Person`

Conversie la Person, deci listă omogenă

Legare statică,
pentru eficienţă
Se ştie ca toate elementele sunt de tip Person

Legare dinamică

•**Observaţie.** Programele care implementeză polimorfismull parametric sunt mai eficiente decât cele care implementează polimorfismul de subtip, dar mai puţin generale.

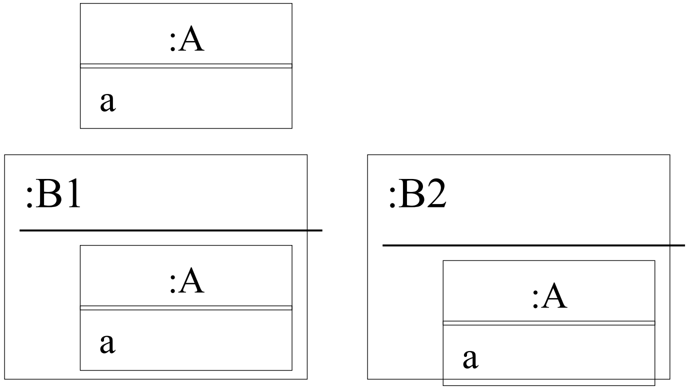

TypeOfElement : type
BCL
BCL<Person*>
<>
<<bind>>
<>
<<bind>>
BCL<Person >

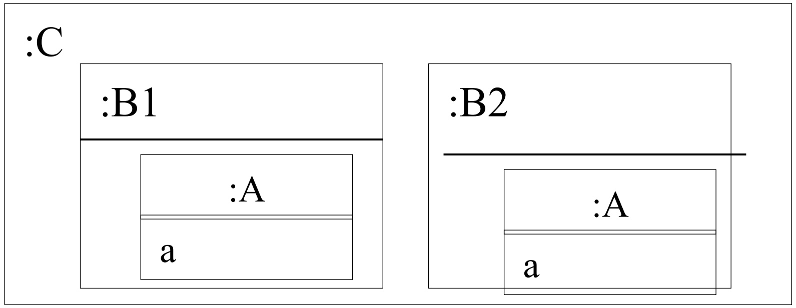
Polimorfism parametric (generic) (4/4)

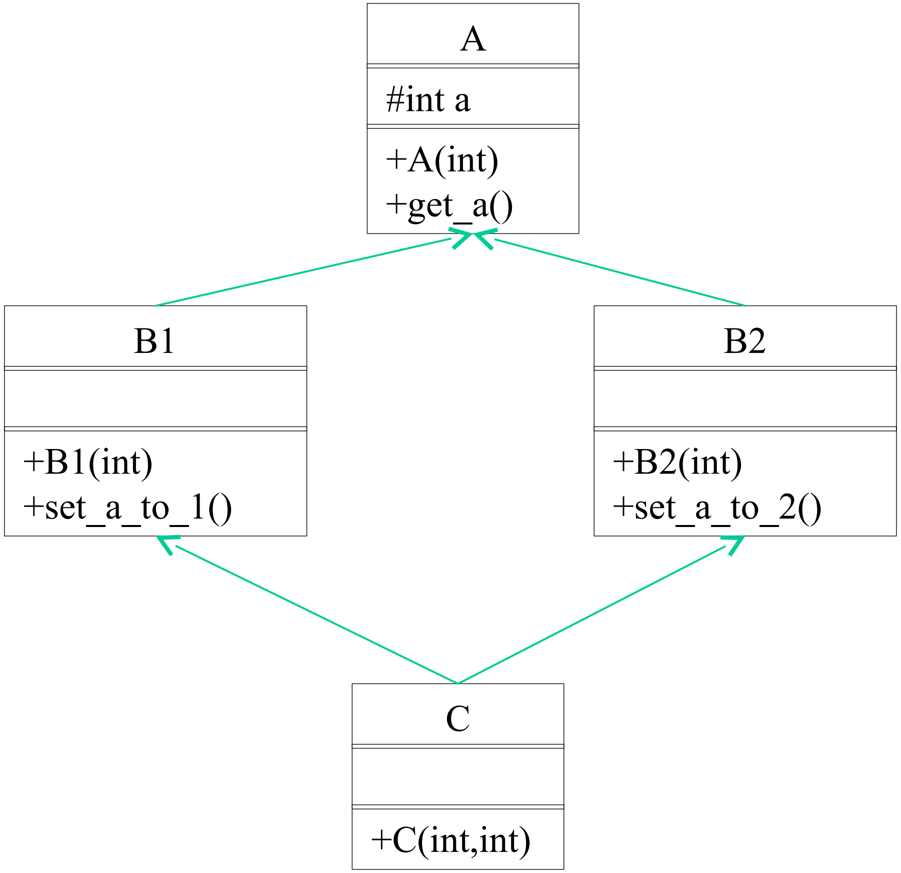
Moştenire multiplă in C++

- Ambiguitate, rezolvata prin:
  - specializare (overriding)
  - nume calificate

Atribute în copie







Multiple inheritance, cont

```
class A{
public:
    A(int i){a=i;}
    int get_a(){return a;}
protected:
    int a;
};
class B1: public A{
public:
    B1(int i):A(i){}
    void set_a_to_1(){a=1;}
};
class B2: public A{
public:
    B2(int i):A(i){}
    void set_a_to_2(){a=2;}
};
class C: public B1,public B2{
public:
    C(int i=0, int k=0):
              B1(i), B2(k){}
};
```

```
//file
int _tmain(int argc, _TCHAR* argv[])
{
        C c(0,0);
        c.set_a_to_1();
        // cout<< c.get_a()<<endl;
        //ambiguous access
        cout<< c.B1::get_a()<<endl;
        cout<< c.B2::get_a()<<endl;
        c.set_a_to_2();
        // cout<< c.get_a()<<endl;
        //ambiguous access
        cout<< c.B1::get_a()<<endl;
        cout<< c.B2::get_a()<<endl;
        _getch();
        return 0;
}
// results
1
0
1
2
```

Obiecte fantoma; constructori cu semantică prin referinţă sau prin valoare

Functions and operators overloading

- Why overloading?

```
void swap(float, float){…}
int k, j; swap(k,j); // does not work. Why?
```

- Solutions:
  - “overnaming” ? Cumbersome!

    ```
    void swap_floats(float, float)
    void swap_ints(int,int)
    ```

    etc.

  - Overloading: Nice!

    ```
    void swap(float,float);
    void swap(int,int);
    ```

    etc.

- You have already used lots of “overloading”!

```
cout<<1<<“a”<<1.2<<endl;
```

Finding the Address of an Overloaded Function

(Herbert Schildt)

- In general, when you assign the address of an overloaded function to a function pointer, it is the declaration of the pointer that determines which function's address is obtained.
- Further, the declaration of the function pointer must exactly match one and only one of the overloaded function's declarations.

```
int myfunc(int a);
int myfunc(int a, int b);
int (*fp)(int a); // pointer to int f(int)


fp = myfunc; // points to myfunc(int)
cout << fp(5); //OK
```

Errors on overloading

- Overloaded finctions must be different:
  - either in the type of parameters or in
  - number of parameters
- Errors:

```
float  func();
int func(); // error, return type is not sufficient

void f(int *p);
void f(int p[]); // error, int[] int * are the same

void f(int x);
void f(int &x);// error,
             //passing mechanism is not sufficient
```

Ambiguities on overloading

- Automatic conversion:

```
void f(double x);
void f(float x);
f(10.1); // no error, f(double); automatic conversion
   to double
f(10); // ambiguity!

void g(char x);
void g(unsigned x);
g(‘c’); // no error, g(char);
g(10); // ambiguity! 10 converted to char otr to
   unsigned char
```

- Implicit arguments:

```
   void f(int x);
   void f(int x, int y=0);
   f(10); // ambiguity!
```

Default Arguments vs. Overloading

(Herbert Schildt)

- In some situations, default arguments can be used as a shorthand form of function overloading.
- Example. Two customized versions of the standard strcat() function.
  The first version will operate like strcat() and concatenate the entire contents of one string to the end of another.
  The second version takes a third argument that specifies the number of characters to concatenate.

  Using overloading:

  ```
  void mystrcat(char *s1, char *s2, int len);
  void mystrcat(char *s1, char *s2);
  ```

  Using default arguments:

  ```
  void mystrcat(char *s1, char *s2, int len = -1);
  ```

Operator overloading

- Operator overloading using member methods
- Operator overloading using friend functions
- Overloading assignment operator
  - In C++, if the = is not overloaded, a default assignment operation is created automatically for any class you define. The default assignment is simply a member-by-member, bitwise copy.
  - By overloading the =, you can define explicitly what the assignment does relative to a class. Notice that the **operator=() function returns *this**
  - Use only method to overload
  - It is not inherited by subclasses
- Except for the = operator, operator functions are inherited by any derived class. However, a derived class is free to overload any operator (including those overloaded by the base class).
- These operators cannot be overloaded:
  . : : .* ?

- Creating Prefix Forms of the Increment and Decrement Operators (++x, --x)

  ```
  C operator++();
  ```

- Creating Postfix Forms of the Increment and Decrement Operators (x++, x--)

  ```
  C operator++(int x);
  ```

- Overloading shorthand operators (+=, *= etc.)

Single Choice Principle

Java

Some history (1/3)

- **Java** is a programming language originally developed by James Gosling (O.C., Ph.D.) at Sun Microsystems (later, a subsidiary of Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform.
- James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box (receive the additional analog cable TV channels and convert them to frequencies that could be seen on a regular TV.) projects.
- The language, initially called *Oak* after an oak tree that stood outside Gosling's office, also went by the name *Green* and ended up later renamed as *Java*, from a list of random words.

Some history (2/3)

- Java is an object oriented language whose main purpose was to be used with embedded systems such as cell phones (set-top boxes contains also such systems).
- But later it gained more importance to be used with Web pages that were dynamic in nature. Java applets and servlets are the important mechanisms for implementing this.
- Another advantage of using Java is the concept of JavaBeans, which is a software component model for Java that allows the rapid development of an application by using a visual buider

Some history (3/3)

- Java syntax is much derived from C and C++, but:
  - has a simpler object model and
  - fewer low-level facilities.
- Java applications are typically compiled to *byte code (class file)* that can run on any *Java Virtual Machine (JVM)* regardless of computer architecture.
- Java is considered by many as one of the most influential programming languages of the 20th century, and is widely used from application software to web applications.

Java goals and benefits

Bruce Eckel:

- Java goal: reducing complexity *for the programmer.* In the early days, this goal resulted in code that didn't run very fast (although this has improved over time), but it has indeed produced amazing reductions in development time—half or less of the time that it takes to create an equivalent C++ program.
- It goes on to wrap many of the complex tasks that have become important, such as **multithreading** and **network programming**, in language features or libraries that can at times make those tasks easy.
- In all ways—creating the programs, working in teams, building user interfaces to communicate with the user, running the programs on different types of machines, and easily writing programs that communicate across the Internet—Java increases the communication bandwidth *between people.*

Java, compile and running

- JDK, Java Developer’s Kit from Sun MicroSystems
  - javac, Java compiler
  - virtual code (byte code)
- JRE, Java Runtime Environment
  - java, virtual machine
  - interpreter

Comparing C++ and Java (Bruce Eckel, Thinking in java, 1998)

Comparing C++ and Java (1.1/20) (Bruce Eckel, Thinking in java, 1998)

- Interpreted Java runs slower than C++
  - But nothing prevents java from being compiled and there are just-in-time compilers
- Java has both kinds of comments like C++ does
  - /*--*/ traditional C-style multiline comment
  - // C++ line comment
- Java has comment documentation
  - Comment documentation appears between /** and */
  - The tool to extract the comments is called *Javadoc*, and it is part of the JDK installation.
  - The output of Javadoc is an HTML file that you can view with your Web browser

Comparing C++ and Java (1.2/20) (Bruce Eckel, Thinking in java, 1998)

- Java has built-in support for commenting documentation (for documentation maintenance)

```
package javaapplication5;
/**
 * @author Tudor
 */
/**
A simple class: objects have no fields
*/
public class Main {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }
}
```

>javadoc Main.java
to produce Main.html and many others

Displayed by a browser, Main.html

javaapplication5
Class Main

java.lang.Object
    |-----javaapplication5.Main

public class **Main**
extends java.lang.Object

A simple class: objects have no fields
Constructor Summary
Main()

Method Summary
static void **main**(java.lang.String[] args)

Method Detail
 main
public static void **main**(java.lang.String[] args)
Parameters:
        args - the command line arguments

Comparing C++ and Java (2/20) (Bruce Eckel, Thinking in java, 1998)

- Everything must be in a class
  - There are no global functions or data
  - If you want the equivalent, make ***static*** methods or data.

```
public class Main {
    public static void main(String[] args) {
                System.out.println(""+C.addUnu(1));
    }
    double f=java.lang.Math.PI;
}

class C{
  public static double e=2.71;
  public static int addUnu(int i){return i+1;}
  double p= java.lang.Math.abs(-1.34);
}
/* Not
int addUnu(int i){return i+1;}
*/
```

- All method definitions are in the body of the class (but they are not inlined)
- Class definition: roughly the same form as in C++, but no closing semicolon.
- There are no scope resolution operator ::. Java uses “.”(dot) for everything.
  - ClassName.methodName() (calling a ***static*** method)
  - import java.awt.*; (package names are established using the dot)

- Java has primitive types for efficient access.
  - ***boolean, char, byte, short, int, long, float, double***
  - all primitive types have *specified sizes that are machine independent for portability*
  - Type checking and type requirements are much tighter in Java:
    - Conditional expression *must be only boolean,* not integral
    - The result of an expression must be used; you can’t just say x+y for side efect.

Comparing C++ and Java (3/20) (Bruce Eckel, Thinking in java, 1998)

Java, *"is more of a pure OO language"*

Tipuri primitive	Lunigime codificare	Valoare minimă	Valoare maximă	Tip înfăşurător *(wrapper type)* subtip Object
Boolean {true, false}	unspecified	-	-	**Boolean**
char	16 biţi	Unicode 0	Unicode 2^{16}-1	**Character**
byte	8 biţi	-128	+127	**Byte**
short	16 biţi	-2^{15}	2^{15}-1	**Short**
int	32 biţi	-2^{31}	2^{31}-1	**Integer**
long	64 biţi	-2^{63}	2^{63}-1	**Long**
float	32 biţi	*	*	**Float**
double	64 biţi	*	*	**Double**
void	-	-	-	**Void**

nu se schimbă de la
o arhitectură la alta,
este unul din motivele de **portabilitate**
ale programelor Java

Operatorul < nu este aplicabil

tipului boolean!

Fără bit de semn,
unsigned

y in your code. If not, you can use a "Unicode escape" such as "\u0108" (capital C with circumflex), or "S\u00ED se\u00F1or" (Sí Señor in Spanish). Always use 'single quotes' for char lit

The integral types (byte, short, int, and long) can be expressed using decimal, octal, or hexadecimal number systems.

Literals

- *float* and *double,* should never be used for precise values, such as currency.

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int decVal = 26; int octVal = 032; int hexVal = 0x1a
double d1 = 123.4;
double d2 = 1.234e2; d3=1.234E2; // same value as d1, but in scientific notation
float f1 = 123.4f,  f2=123.4F;
```

The integral types (byte, short, int, and long) can be expressed using decimal, octal, or hexadecimal number systems.

char and String

•Literals of types char and String may contain any Unicode (UTF-16) characters.

you can use a "Unicode escape" :

- •'\u0108‘ (capital C with circumflex), use always ‘ for char literals
- • "S\u00ED se\u00F1or" (Sí Señor in Spanish), use always " for String
- •int se\u00F1or; (Unicode escape sequences may be used elsewhere in a program)

• Always and "double quotes" for String literals.

•special escape sequences for char and String literals:

\b (backspace), \t (tab), \n (line feed), \f (form feed), \r (carriage return), \" (double quote), \' (single quote), and \\ (backslash).

null literal can be used as a value for any reference type. null may be assigned to any variable, except variables of primitive types. (null is often used in programs as a marker to indicate that some object is unavailable.)

there's also a special kind of literal called a *class literal*, formed by taking a type name and appending ".class"; for example, String.class. This refers to the object (of type Class) that represents the type itself.

```
System.out.println (String.class.getClass());         //class java.lang.Class

System.out.println (Integer.class.getClass());        //class java.lang.Class

System.out.println (String.class);                    //class java.lang.String

System.out.println (String.class.toString());         //class java.lang.String
```

Comparing C++ and Java (4/20) (Bruce Eckel, Thinking in java)

char type uses the international 16-bit Unicode character set, so it can represent most national characters.

```
char Gamma='Γ';
int i=Gamma;
System.out.println(i); // prints 915
```

- Static quoted strings are automatically converted into String objects. There is no independent static character array string like in C or C++

```
System.out.println("a string".getClass().getName()); // prints: java.lang.String
```

- Java adds triple shift >>> to act as a logical right shift by inserting zeroes at the top end (>> inserts the bit sign)

```
System.out.println(-1>>>1);       // prints 2147483647 (=2^31 -1)
System.out.println(-1>>1);        // prints -1
```

Comparing C++ and Java (5/20) (Bruce Eckel, Thinking in java, 1998)

- Arrays have different structure and behavior in Java than they do in C++ (although they look similar)
  - ***length*** member
  - Run time checking of bounds
  - All arrays are created on the heap
  - array identifier is a first-class object, with all of the methods commonly available to all other objects.
  - You can assign one array to another (the array handle is simply copied)

```
 int[] a=new int[2];// or int a[]
int b[] ={1,2,3};   // aggregate initialization
char[] c;  c= new char[3];
a=b;
    System.out.println(a.getClass().getName()); // prints [I
     System.out.println(b.getClass().getName()); // prints [I
    System.out.println(c.getClass().getName()); // prints [C, not String
    System.out.println(a.length); // prints 3
```

Comparing C++ and Java (6/20) (Bruce Eckel, Thinking in java, 1998)

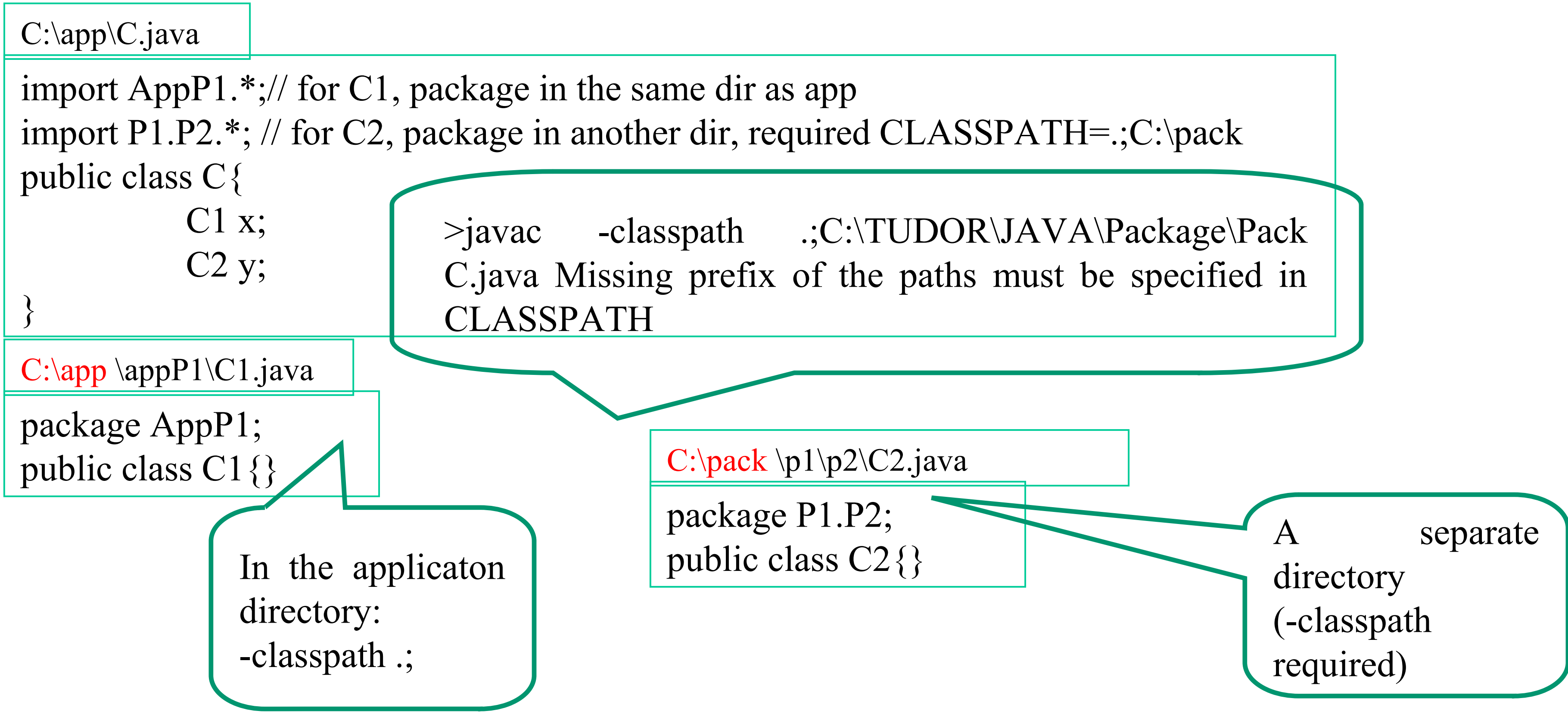
- All objects of non-primitive types can be created only via ***new*** operator, on heap.
  - There is no equivalent to creating non-primitive objects on the stack.
- All primitive types can be created only on the stack, without new.
  - There are wrapper classes for all primitive types, so you can create heap-base objects.

```
                Integer I= new Integer(1); // on heap
        int i=1; //on stack
        I=i; // boxing i into an object on heap
        i=I; // unboxing an object to a primitive type
```

- Arrays of primitives are of special case: can be allocated via aggregate initialization (like in C++) or by using new.

Comparing C++ and Java (7/20) (Bruce Eckel, Thinking in java)

- No forward declarations are necessary in Java
- Java has no preprocessor and preprocessor-like macros
  - If you want to use classes from another library, use **import**.
- Java uses ***packages*** in place of ***namespaces***
  direct relationship between packages and folder structure(. stands for \)



C:\app\C.java

```
import AppP1.*;// for C1, package in the same dir as app
import P1.P2.*; // for C2, package in another dir, required CLASSPATH=.;C:\pack
public class C{
        C1 x;
        C2 y;
}
```

>javac -classpath .;C:\TUDOR\JAVA\Package\Pack C.java Missing prefix of the paths must be specified in CLASSPATH

C:\app \appP1\C1.java

```
package AppP1;
public class C1{}
```

In the applicaton directory:
-classpath .;

C:\pack \p1\p2\C2.java

```
package P1.P2;
public class C2{}
```

A separate directory (-classpath required)

Comparing C++ and Java (8/20) (Bruce Eckel, Thinking in java, 1998)

- Initialization:
  - Objects handles defined as class members are automatically initialized to ***null.***
    Object o; // o is a handle to an Object
  - Initialization of primitive class data members is guaranteed (to zero or equivalent)
  - Data member can be initialized explicitelly,
    - either when you define them (*not in C++ !),* class C{int i=1;…}
    - or in constructor.
  - Initialization is consistent for ***static*** and non-***static*** members alike.

    ```
    class C{
              int i=1;
              static int s=1;
    }
    ```

    - You do not need to define storage for static members, like in C++

      ```
      class C{static int s;}
      int C::s=1;
      ```

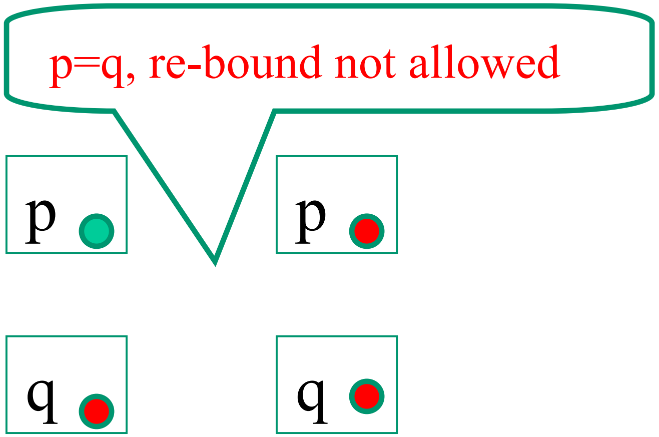
Comparing C++ and Java (9/20) (Bruce Eckel, Thinking in java, 1998)

- There are no Java pointers in the sense of C or C++
  - new operator creates a "*reference*" *(a kind of "restricted " pointer)*
    - Don't have to be bound at the point of creation (unlike C++)
    - Can be re-bound, (unlike C++) which eliminates the need for pointers
  - In any event, there is no pointer arithmetic.

How to make something like C swap(int *, int*) ?

C++

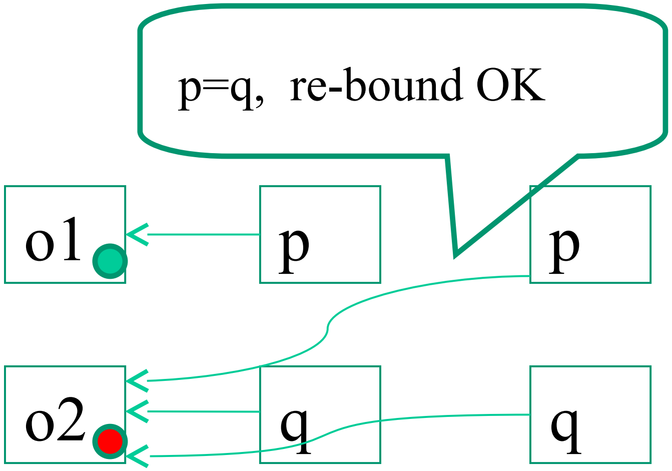
Person p, q;//initialization done!



p=q, re-bound not allowed

Java

Person p,q; // no initialization, **new** required later



p=q, re-bound OK

Comparing C++ and Java (10.1/20) (Bruce Eckel, Thinking in java)

- Passing arguments to methods: two points of view on the same mechanism.
  - ***Java passes everything by value.***

```
     int a=1; int b=2;
     C.swap(a,b); 1 2, no swap, Java passes everything by value.
      Integer aI=1, bI=2;
     C.swap(aI,bI); );//1 2,  no swap, Java passes everything by value
class C{
   public static void swap (int i, int j){int t=i; i=j; j=t;}
   public static void swap (Integer i, Integer j){Integer t=i; i=j; j=t;}
}
```

How to swap?

  - ***Java passes primitives by value , but objects are passed by reference.***

"It depends on how you think of a handle,"
- passing a handle allows the caller's object to be changed unexpectedly.

Comparing C++ and Java (10.2/20) (Bruce Eckel, Thinking in java)

- It looks as objects are passed by reference.
  - Hence, there are no copy constructor
- Java has constructors that are similar to C++ constructors
  - You get a default constructor if you don’t define one, and if you define a non-default constructor, there’s no automatic default constructor defined for you, just like in C++
  - there are no copy constructor (no need, it looks as objects are passed by reference)
- There are **no destructors** in Java
  - The lifetime of an object is determined by the *garbage collector*
  - Garbage collection makes *memory leaks* much harder, but not impossibly
  - There is a **finalize( )** method that’s a member of each class, something like a C++ destructor, but **finalize( )** is called by the garbage collector (when20) and is supposed to be responsible only for releasing "resources" (such as open files, sockets, ports, URLs, etc).
  - If you need something done at a specific point, you must create a special *clean-up* method and call it, not rely upon finalize( ).
    Note: In the clean-up method, explicitly call all the clean-ups for the base class and member objects .

Comparing C++ and Java (11/20) (Bruce Eckel, Thinking in java, 1998)

- Java has method overloadig that works virtually identically to C++ function overloading
- Java does not support default arguments
- There is no ***goto*** in Java
  - Use ***break*** [<iteration label>] or ***continue*** [<iteration label>] to jump out of the middle of multiply nested loops.
- Java use a single-rooted hierarchy (root class java.lang.***Object***)
  - All classes of an application are in a single tree
  - **C++ appers to be the single OO language** that does not impose a single-rooted hierarchy. It has a “forest” of trees.

Comparing C++ and Java (12/20) (Bruce Eckel, Thinking in java)

- Initially, Java has no parameterized types (or templates)
  - The Java collections: Vector, Stack, Hashtable hold Object references
  - They are not designed for efficiency, like C++ Standard Template Library (STL)
- Since Java SE5, Java supports *parameterized types (by generics)*
- Generics also work with interfaces (no details here)

```
//: generics/Holder3.java , from Bruce Eckel
public class Holder3<T> {  // compare with C++ template
        private T a;
        public Holder3(T a) { this.a = a; }
        public void set(T a) { this.a = a; }
        public T get() { return a; }
        public static void main(String[] args) {
          Holder3<Automobile> h3 =
                new Holder3<Automobile>(new Automobile());
          Automobile a = h3.get(); // No cast needed
          // h3.set("Not an Automobile"); // Error
          // h3.set(1); // Error
        }
} ///:~
```

Comparing C++ and Java (13/20) (Bruce Eckel, Thinking in java, 1998)

- You can also parameterize methods within a class. The class itself may or may not be generic—this is independent of whether you have a generic method.

```
//: generics/GenericMethods.java (from Bruce Eckel)
public class GenericMethods {
    public <T> void f(T x) {
    System.out.println(x.getClass().getName());
    }
    public static void main(String[] args) {
    GenericMethods gm = new GenericMethods();
    gm.f("");
    gm.f(1);
    gm.f(1.0);
    gm.f(1.0F);
    gm.f('c');
    gm.f(gm);
    }
}
```

```
/* Output:
java.lang.String
java.lang.Integer
java.lang.Double
java.lang.Float
java.lang.Character
GenericMethods
```

Comparing C++ and Java (14/20) (Bruce Eckel, Thinking in java, 1998)

- Access specifiers are placed on each definition (instead of controlling blocks, like in C++)
  - Explicit access specifiers:
    - ***public, private,***
    - ***protected (***accessible to inheritors *and to other in the same package**)***
      There is no equivalent of C++ ***protected***
  - Implicit: “friend” only to other classes in the same package

Comparing C++ and Java (15/20) (Bruce Eckel, Thinking in java, 1998)

- Java **does not provide multiple inheritance (**but provides **interfaces)**
- Inheritance has the same effect as in C++, but syntax is different
  - ***extends***

super keyword to specify methods to be called in the base class (**but only one level up in the hierarchy„ in the parent class**)

```
class B{
                B(int i){}
               void m(){}
}
class D extends B{ // only a single base class alowed
               D(int i,int j){super(i);}
               @Override
               public void m(){super.m();}
}
```

Comparing C++ and Java (16/20) (Bruce Eckel, Thinking in java)

- Inheritance in Java does not change the protection level of the inherited members
  - (you can not specify public, protected or private as you can in C++, class D:protected B{…})
- Overridden methods in a derived class can not assign an weaker access privilege
- There is **no virtual** keyword, all non-static methods are late bound
  - ***final*** keyword provides some latitude for efficiency tuning (the method can't be overridden, it may be **early bound**) ; final methods can be overriden.
- Java provides **interfaces** (see later)
  - Creates the equivalent of an abstract class with no data members.
  - Classes ***implements*** as many interfaces as necessary
- Java has method overloading, but no operator overloading.
  - String class does use + and += to concatenate strings, but that is a special built-in case

- Exception specifications are vastly superior to those in C++ (see later)
- Java has built-in multithreading support (see later)

Comparing C++ and Java (17/20) (Bruce Eckel, Thinking in java, 1998)

- Run-time type identification functionality is quite similar to that in C++. To get information about handle X, you can use, for example
  X.getClass().getName();
- To perform a type-safe downcast :
  derived d = (derived)base;
  just like an old-style C cast.
- The **const** issues in C++ are avoided in Java by convention. To create a compile-time constant value:

  ```
  class C{
          static final int SIZE = 255;
          static final int BSIZE = 8 * SIZE;
  }
  ```

  To use: C.SIZE a.s.o.

Comparing C++ and Java (18/20) (Bruce Eckel, Thinking in java, 1998)

- Java contains standard libraries for solving specific tasks (C++ relies on non-standard third-party libraries) .
  These tasks include:
  - Networking
  - Database Connection (via JDBC)
  - Multithreading
  - Distributed Objects (via RMI and CORBA)
  - Compression
  - Commerce

  The availability and standard nature of these libraries allow for more rapid application development.

Comparing C++ and Java (19/20) (Bruce Eckel, Thinking in java, 1998)

- Since Java can be too restrictive in some cases, you could be prevented from doing important tasks such as directly accessing hardware. Java solves this with *native methods* that allow you to call a function written in another language (currently only C and C++ are supported). Thus, you can always solve a platform-specific problem (in a relatively non-portable fashion, but then that code is isolated). Applets cannot call native methods, only applications.

Comparing C++ and Java (20/20) (Bruce Eckel, Thinking in java)

- Generally, Java is more robust than C++, via:
  - **Object handles initialized to null (a keyword)**
  - **Handles are allways checked and exception are thrown for failures**
  - **All arrays are checked for bound violations**
  - **Automatic garbage collection prevents memory leaks**
  - **Clean, relatively fool-prof exception handling**
  - **Simple language support for multithreading**
  - **Bytecode verification of network applets**

Program “standalone”; regular console application

- Un program “standalone” conţine o clasă ce are o metodă statică
  `public void static main(String[] args)`
- Executarea (interpretativă) este iniţiată prin metoda statică
- Exemplu Java\Main\Standalone.java

Obligatoriu
numele fisierului este
numele clasei public
Iar extensia `.java`

```
// file Standalone.java
public class Standalone {
    public static void main(String[] args) {
        System.out.println("Hello, Program standalone cu "
                  + args.length
                  + " argumente in linia de comanda");
        for(int i=0; i<args.length;i++)
          System.out.println(args[i]);
    }
}
```

atribut al oricărui
obiect vectori

Lansare in executare prin linia de comandă
>java Standalone sir_1 sir_2… sir_n
unde $n \geq 0$ iar şirurile sunt separtae
prin unul sau mai multe spaţii

Program “standalone”; regular GUI application (windowed app)

```
import java.awt.*;// Abstract Window Toolkit
public class GUI extends Frame{
    public static void main(String[] args) {
         GUI f = new GUI();
        //f.resize(300,200); deprecated
        //f.show();deprecated;
         f.setSize(300,200);
         f.setTitle("My Graphical User Interface");
         f.setVisible(true);
    }
}
```

Does not close on clicking x

GUIs with Swing vs. awt

```
import javax.swing.*;
public class JGUI extends JFrame{
    public static void main(String[] args) {
            JGUI f = new JGUI();
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            f.setTitle("Hello Swing");
            f.setSize(300, 200);
            f.setVisible(true);
    }
}
```

closes on clicking x

Java client-side programming

- The Web's initial server-browser design provided for interactive content, but the interactivity was ***completely provided by the server***. The server produced static pages for the client browser, which would simply interpret and display them (*HyperText Markup Language (HTML) contains simple mechanisms for data gathering)*
- Submiting data back to the server passes through the *Common Gateway Interface (CGI)* provided on all Web servers.
- The response of a CGI program depends on how much data must be sent, as well as the load on both the server and the Internet.
- Solution: **client-side programming;** a Web browser is like a limited operating system and it is harnessed to do whatever work it can
- Client-side programming solutions:
  - Plug-ins
  - Scripting languages (JavaScript, VBScript)
  - Java allows client-side programming via the *applet and with Java Web Start.*
    Java Runtime Environment (JRE) have to be installed. Note that Microsoft chose not to include the JRE with Internet Explorer

Applet

- An applet is a mini-program that will run only under a Web browser.
- The applet is downloaded automatically as part of a Web page (just as, for example, a graphic is automatically downloaded).
- When the applet is activated, it executes a program.
- There is a a single program (applet, on the server) and that program automatically works with all computers that have browsers with built-in Java interpreters.
- it provides you with a way to automatically distribute the client software from the server at the time the user needs the client software, and no sooner
- Because they must be safe, applets are limited in what they can accomplish.
  - *An applet can't touch the local disk. (writing or reading) a.s.o.*
- Many applet restrictions are relaxed for trusted applets (those signed by a trusted source) in newer browsers.

Example: an applet

```
// from Bruce Eckel, Thinking in Java, page 477
import java.awt.*;
import java.applet.*;
public class Applet3 extends Applet {
        String s; int inits = 0, starts = 0, stops = 0,  paints =0;
        public void init() { inits++; }
        public void start() { starts++; }
        public void stop() { stops++; }
        public void paint(Graphics g) {
                paints++;
                s = "inits: " + inits +
                        ", starts: " + starts +
                        ", paints: " + paints +
                        ", stops: " + stops;
                g.drawString(s, 10, 10);
        }
}
```

two paints for each browser window

minimization

//Se lansează prin **încărcarea într-un browser** a următorului **fişier HTML**

```
<APPLET code=“Applet3.class" width="500" height="100">
</APPLET>
```

The singly rooted hierarchy

- In Java (as with virtually all other OOP languages *except for C++)* all classes are inherited from a single base class, *java.lang.Object*.
- Benefits:
  - All objects have an interface in common, so they are all ultimately the same fundamental type.
  - All objects can be guaranteed to have certain functionality. You know you can perform certain basic operations on every object in your system.
    - boolean equals(Object);
    - int hashCode();
    - Class<?>getClass();
    - String toString()
    - a.s.o
  - All objects can easily be created on the heap, and argument passing is greatly simplified.
  - makes it much easier to implement a garbage collector (one of the fundamental improvements of Java over C++)
  - information about the type of an object is guaranteed to be in all objects, you'll never end up with an object whose type you cannot determine;
    - this is especially important with system-level operations, such as exception handling
    - allow greater flexibility in programming.

Clasa radacină `Object` (`java.lang.Object`)

- Java\Object\Main.java

```
// file Main.java
public class Main {
    public static void main(String[] args) {
        Object o=new Object();
        Object o1=o;

        System.out.println(o.toString());
        System.out.println(o);
        System.out.println(
          o.getClass().getName() + '@' +
   Integer.toHexString(o.hashCode()));


        System.out.println(o.equals(o1)); //true
        o=new Object();
        System.out.println(o.equals(o1)); // false !

        System.out.println(o.hashCode()); //  18464898,
   System.out.println(o1.hashCode()); // 17332331,
   System.out.println(o.hashCode());  // 18464898
    }
}
```

toate cele 3 instr afiseaza java.lang.Object@108786b

dar la urmatoarea executare posibil alta valoare !

equals implementează cea mai tare condiţie de egalitate
De obicei, este specializată (overriding)

Everything is an Object

- System.out.println((Object)true); //true
- System.out.println((Object)1); // 1
- System.out.println((Object)1.2); //1.2
- System.out.println(((Object)1).toString()); //1

Explicit *upcasting*
treating a derived type as though it
were its base type

Specializarea metodei `equals` *(overriding)*

```
class C{
    public C(int i){x=i;}
    public boolean equals(C c) { return x==c.x;}
    private int x=0;
}
C c1,c2;
c1=new C(1);
c2=new C(1);

System.out.println(   c1==c2                    );              // false!
System.out.println(   ((Object) c1).equals(c2)  );    // false

System.out.println(   c1.equals(c2)             );    // true
System.out.println(   (Object) c1.equals(c2)    );              //true
```

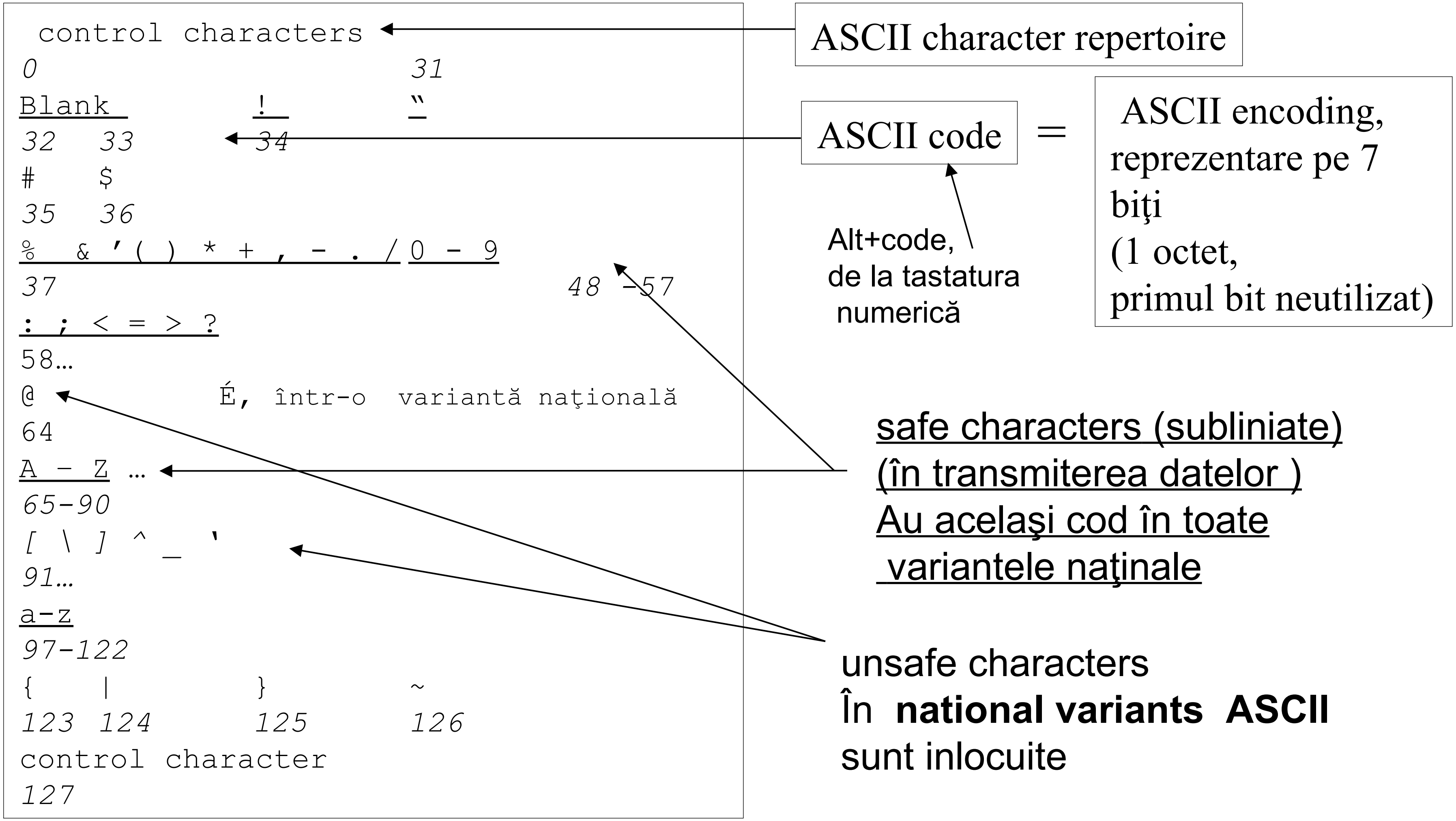
Character repertoire, code, encoding

- Character repertoire: a set of (distinct) characters
- Character code: a one-to-one correspondence between repertoire and a set of nonnegative integers
- Character encoding: a one-to-one correspondence between the set of nonnegative integers and sequences of octets.

Exemplu:

Character repertoire	Glyph (forma vizuală)	Character Code	Character encoding
NUMBER SIGN	#	35	0x23
Upper case A	A	65	0x41
Cyrillic “ia”	я	36817	0x8FD1
etc.			

Example. Character Encoding pe 7 biţi:
American Standard Code for Information Interchange (ASCII)
US-ASCII şi variantele naţionale



control characters

0 31

Blank ! ‟

32 33 34

$

35 36

% & ’ ( ) * + , - . / 0 - 9

37 48 - 57

: ; < = > ?

58…

@ É, într-o variantă naţională

64

A - Z …

65-90

[\ ] ^ _ '

91…

a-z

97-122

{ | } ~

123 124 125 126

control character

127

ASCII character repertoire

ASCII code = ASCII encoding, reprezentare pe 7 biţi (1 octet, primul bit neutilizat)

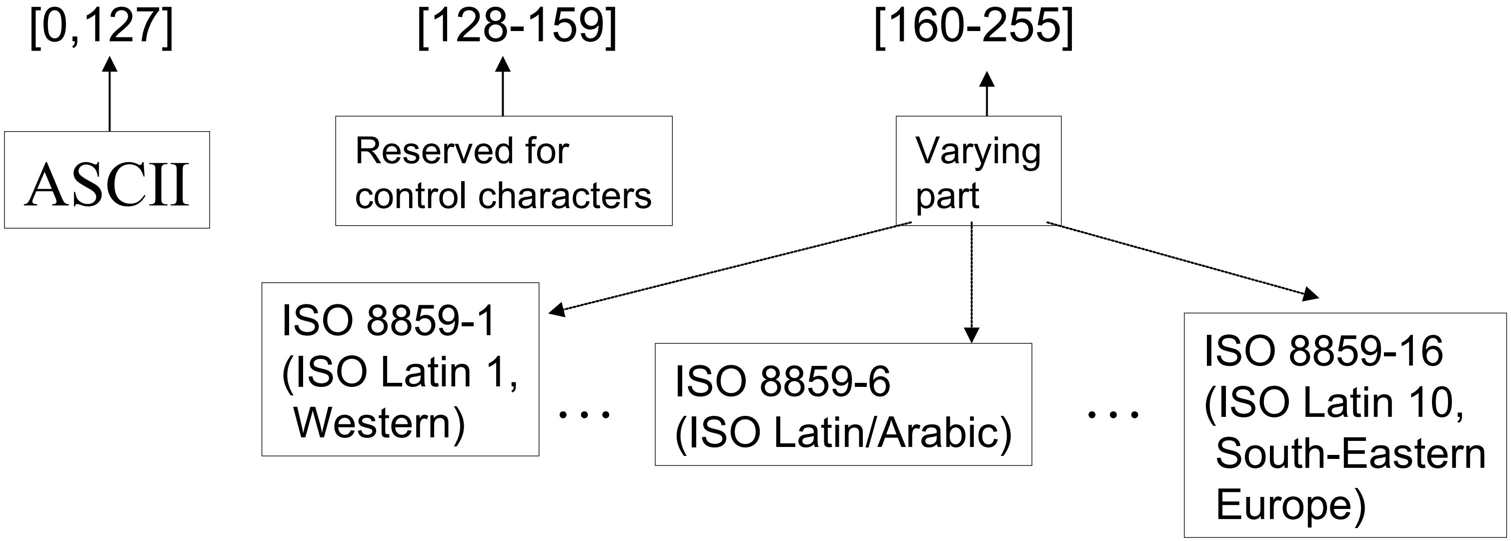
Alt+code, de la tastatura numerică

safe characters (subliniate) (în transmiterea datelor) Au acelaşi cod în toate variantele naţinale

unsafe characters
În **national variants ASCII**
sunt inlocuite

Character encoding: 8 biţi (un octet) Familia ISO 8859 (Latin)

- Familia de seturi de caractere (repertoire) ISO 8859(International Standard Organisation, Latin)
  - Orice set din această familie are 256 de caractere, numerotate de la 0 la 255
  - US- ASCII este un subset al oricărui set
  - Codurile :



[0,127]

[128-159]

[160-255]

ASCII

Reserved for control characters

Varying part

ISO 8859-1 (ISO Latin 1, Western)

…

ISO 8859-6 (ISO Latin/Arabic)

…

ISO 8859-16 (ISO Latin 10, South-Eastern Europe)

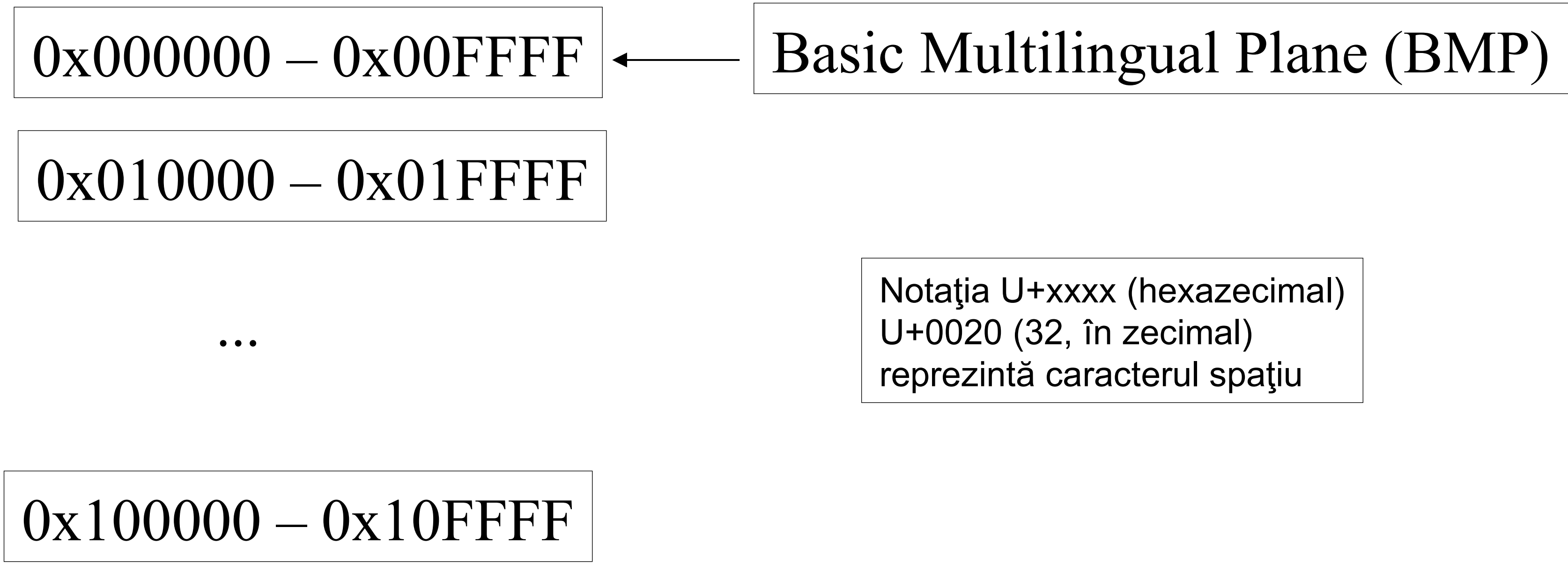
  - Character encoding: 8 biţi (un octet)

Familia Windows Latin (code pages)

- Familia este similară cu ISO Latin-1, dar o parte din codurile 128-159 sunt asociate unor caractere tipăribile, precum (smart quotes, em dash, en dash, trademark symbol)
- Seturile se numesc Code pages (Cp):
  - WinLatin 1 (Cp 1252) (similar cu ISO Latin 1)
  - WinGreek (Cp 1253)
  - Cp1250, pentru România

Universal Character Set (UCS), Unicode

- UCS este un *character repertoire* şi un *character code* de dimensiuni mari, definit prin standardul ISO 10646
- Unicode este un *character repertoire, character code* şi *character encoding* compatibil cu standardul ISO 10646
- Iniţial a fost proiectat pentru coduri pe 16 biţi (65536 caractere), ulterior a fost extins pentru a putea include coduri (poziţii în repertoire) din intervalul 0x0 – 0x10FFFF, adică 1.114.111 caractere, repartizate în 17 planuri de 16 biţi.



0x000000 – 0x00FFFF

Basic Multilingual Plane (BMP)

0x010000 – 0x01FFFF

…

0x100000 – 0x10FFFF

Notaţia U+xxxx (hexazecimal)
U+0020 (32, în zecimal)
reprezintă caracterul spaţiu

Encodings for Unicode

- UCS-2 (the native Unicode encoding), 2 octeţi consecutivi, utilizat înainte de extinderea Unicode
- UTF-32, codificare directă pe 4 octeţi
- UTF-16:
  - caracterele din BMP, pe 2 octeţi
  - Pentru alte plane, *surrogate pairs*
- UTF-7/ UTF-8 (next slide)
  - Fiecare caracter este reprezentat prin unul sau mai mulţi octeţi cu valori în intervalul [0-127]. **Majoritatea caracterelor ASCII** sunt reprezentate eficient, printr-un singur octet.
- **Observaţie.** Editorul de texte Microsoft@Word furnizeză, la salvarea unui text în formatul Plain text, opţiuni pentru mai multe codificări: UTF-8, UTF-7, US-ASCII, Windows etc.

Encodings for Unicode: UTF-8

- ***UTF-8 (UCS Transformation Format 8)***
  - Caracterele [0-127], un octet: prin urmare, **setul ASCII este reprezentat eficient**
  - În celelalte cazuri, un cod de caracter poate fi reprezentat printr-o secventa de mai multi octeti(conform unui anumit algoritm), fiecare cu valori în intervalul [128-255] (adică cel mai semnificativ bit este totdeauna 1)
  - Pentru caracterele din BMP maximum 3 octeti
    - From Unicode UCS-4 to UTF-8:
      Start with the Unicode number expressed as a decimal number ud.
    - If ud <128 (7F hex) then UTF-8 is 1 byte long, the value of ud.
    - If ud >=128 and <=2047 (7FF hex) then UTF-8 is 2 bytes long.
      byte 1 = 192 + (ud div 64)
      byte 2 = 128 + (ud mod 64)
    - If ud >=2048 and <=65535 (FFFF hex) then UTF-8 is 3 bytes long.
      byte 1 = 224 + (ud div 4096)
      byte 2 = 128 + ((ud div 64) mod 64)
      byte 3 = 128 + (ud mod 64)
  - Pentru caracterele 65536 (10000 hex) <= ud <=2097151 (1FFFFF hex), encoding UTF-8 are 4 octeti.

Example (UTF-8 encoding)

ch	dec	hx	U-hex	U-dec	UTF-dec	UTF-hx	lit	Unicode name	PostScript name
Ş	170	AA	2122	8482	226.132.162	E284A2	â„˘	TRADE MARK	SIGN trademark

Java Encoding Schemes

Character-encoding schemes that are supported by the Java platform.

- **US-ASCII**
- **ISO-8859-1**
- **UTF-8**
- **UTF-16**
    - It uses 16 bits for most characters but includes 32-bit characters for ideogram-based languages such as Chinese.
    - A Western European-language document that uses UTF-16 will be twice as large as the same document encoded using UTF-8.
    - But documents written in far Eastern languages will be far smaller using UTF-16.
    - Note: UTF-16 depends on the system's byte-ordering conventions.
        - Intel processors (those used in PC's) use "Little Endian" byte order: (The little end comes first in a 16-bit or 32-bit word)
        - some systems use the reverse order.
        - Character A is
            - 41 00 in Little Endian
            - 00 41 in Big Endian
    - Interchanging UTF-16 documents between such systems:
        - Should be done by a conversion.

Programe cu nume de variabile în alte limbi (standalone)

- Java\Unicode\Hallo.java

// editor MSWORD: **se salveaza ca Plain text (*.txt), encoding Unicode**

// se compileaza cu
javac -encoding UTF-16 Hallo.java

without –encoding:
error message,
illegal character

- la executare, consola nu suporta afisarea caracterelor.
- vezi varianta applet a aceluiasi program

```
public class Hallo {
  public static void main(String[] args) {
    String α = "Greek: Γρεεκ";
    System.out.println(α);
    String ш = "Russian (Cyrillic): йцнгшщзфывпиь ";
    System.out.println(α);
    String șirÎnRomână = "ăîșț A";
    System.out.println(șirÎnRomână);
  }
}
```

Şir în limba rusă

Variabilă în limba greacă

Hex:
5f 01 69 00 72 00 ce 00 6e 00 52 00 6f 00 6d 00 e2 00 6e 00 03 01 (encoding **Unicode, Little Endian,** 22 bytes)
c5 9f 69 72 c3 8e 6e 52 6f 6d c3 a2 6e c4 83 (encoding **UTF-8,** 15 bytes)

Programe cu nume de variabile în alte limbi (varianta applet)

se salveaza ca Unicode

//Se compileaza cu
javac -encoding UTF-16 HelloApplet.java

- Java\Unicode\HelloApplet.java

```
import java.applet.*;
import java.awt.*;
public class HelloApplet extends Applet{
   public void paint(Graphics g){
           g.drawString("Greek: Γρεεκ", 10,10);
           g.drawString("Russian (Cyrillic): йцунгшщзфывапить", 10,30);

           g.drawString("Romanian: ăîșț", 10,60);
   }
}
```

//Se lansează prin **încărcarea într-un browser** a următorului **fişier HTML**

- Java\Unicode\applet.html

```
<APPLET code="HelloApplet.class" width="500" height="100">
</APPLET>
```

Java şi Character encodigs

- Intern, platforma java (Java SDK şi Java RE) prelucrează caractere Unicode, pe 16 biti.
- Implicit, compilatorul Java prelucrează numai fişiere ISO8859-1(Latin 1); dacă fişierul este scris cu altă codare, aceasta va fi specificataă în opţiunea -encoding
- Sistemul de operare pe care este instalată platforma utilizează o anumită codare (character encoding), numită *codare implicită*.
- Codarea implicită este selectată automat de platforma Java.. Mediul Java realizează conversia Unicode ↔ codare implicită
- Codarea implicită este determinată prin expresia `System.getProperty("file.encoding");` pe calculatorul meu este `Cp1253`, adică `windows-1253`
- Codările suportate de o maşină virtuală Java se pot determina prin servicii ale clasei java.nio.charset.Charset, ca în exemplul următor:
- Pe calculatorul meu, sunt suportate zeci de codări, dar nu ISO 8859-10 (Latin 6, Nordic) şi ISO 8859-16 (Latin 10, South-Eastern Europe) ; codare UTF-16 este suportată şi a fost utilizată în exemplele anterioare pentru conversia de la Unicode la ISO8859-1(Latin 1)

Java\Unicode\Charset.java

```
import java.util.*;
public class Charset{
   public static void main(String[] args){
    System.out.println("Codare implicita: "+ System.getProperty("file.encoding"));
    System.out.println("Alte codari suportate:");
    SortedMap sm=java.nio.charset.Charset.availableCharsets();
    Set c=sm.keySet();
    Object[] ob=c.toArray();
    for(int i=0; i<ob.length; i++)System.out.println(ob[i]);
   }
}
```

Observaţi că au fost utilizate două clase `Charset`, distinse prin pachetele din care fac parte

Unicode

- Tipul char este mulţimea de caractere `international16-bit Unicode character set`.Java\CaractereUnicode.java

```
import java.io.*;
import java.util.*;
public class CaractereUnicode{
  public static void main(String[] args){
        InputStreamReader isr= new InputStreamReader (System.in);
        System.out.println(isr.getEncoding());
        char x='\u0fff';
        System.out.println(x);
        Locale loc= Locale.getDefault();
        System.out.println("Localizare implicita: "+
loc.toString());
        System.out.println(loc.getCountry());
  }
}
```

- Va afişa:
  Cp1252
  ?
  Localizare implicita: en_US
  US

•Numele canonic al setului de caractere Windows `Latin-1`
compilatorul Java prelucrează doar fişiere `Latin-1`
•`Latin-1` este utilizat şi pentru comunicare intre JVM (Unicode)
şi sistemul de operare pe care aceasta este instalată.
• Conversia Latin-1 ↔ altă codare se poate face cu utiliitarul *native2ascii*

Obiectul `loc` conţine informaţii despre limba şi regiunea
la care este setat calculatorul (English US în cazul dat).
Dacă se schimbă setarea
(*Start/Control Panel/Regional and Language Options/Regional Options*
to `Romanian`) se afişează
CP1250,
ro_RO, RO

Specificarea explicită a codării utilizate în operaţiile de intrare-ieşire(1/2)

- Obiectele claselor `InputStream` şi `OutputStream` procesează şiruri de `bytes`.
- Pentru a sigura conversia `codarea unui fisier ↔ Unicode` în operaţiile de intrare ieşire, se utilizează obiecte ale claselor `Reader` sau `Writer`.
- Codarea fişierului, dacă nu este cea implicită, trebuie specificată explicit.

Exemplu. Java\Unicode\SpecifyEncoding.java

```
import java.io.*;
public class SpecifyEncoding {
    public static void main(String[] args) throws Exception{
BufferedReader rdrImplicitEncoding =
    new BufferedReader(
       new InputStreamReader(new FileInputStream("infile.txt")));
     String line = rdrImplicitEncoding.readLine();
     System.out.println(line);
BufferedReader rdrSpecifiedEncoding =
     new BufferedReader(
       new InputStreamReader(new FileInputStream("infile.txt"),
                         "UTF-16"));
       line = rdrSpecifiedEncoding.readLine();
     System.out.println(line);

    }
  }
```

Codarea implicită (CP1253)

Codare specificată (UTF-16)

Fişierul infile.txt conţine textul
`Tudor Bălănescu`
şi a fost salvat Unicode. Citirea utilizând codarea Cp1253 va înscrie în line aproximativ de două ori mai multe caractere decât sunt în fişier, provenind din interpretarea octetului suplimentar utilizat in Unicode

Basic Input/Output

- **I/O Streams**
  - An *I/O Stream* represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.
  - Streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects.
  - Some streams simply pass on data; others manipulate and transform the data in useful ways.
- java.io package contains a fairly large number of classes that deal with Java input and output. Most of the classes consist of:
  - Byte streams that are subclasses of **InputStream** or **OutputStream**
    - There are many byte stream classes. FileInputStream, FileOutputStream etc.
  - Character streams are subclasses of **Reader** and **Writer**
  - Object streams **(ObjectInputStream** and **ObjectOutputStream**) transmit entire objects.
- InputStream and OutputStream reads and writes 8-bit bytes.
- The Reader and Writer classes read and write 16-bit Unicode characters
- ObjectInputStream reads objects; ObjectOutputStream writes objects.

java.io class hierarchy (1/3)(selection)

- **File** (implements java.lang.Comparable, java.io.Serializable)

  File(String pathname)

- ***InputStream*** *(abstract class:superclass of all classes representing an input stream of bytes.)*

  public abstract int read()

  - //Reads the next byte of data from the input stream.
  - The value byte is returned as an int in the range 0 to 255.
  - If no byte is available because the end of the stream has been reached, the value -1 is returned.

  The standard input stream **System.in** is an InputStream

  - **ByteArrayInputStream**
  - **FileInputStream**

    FileInputStream(String), FileInputStream(File), etc.
  - **FilterInputStream**
    - **BufferedInputStream**
    - **DataInputStream** (implements java.io.DataInput):

      read primitive Java data types from an underlying input stream in a machine-independent way

      DataInputStream(InputStream in): creates a DataInputStream that uses the specified underlying InputStream.

      byte readByte(), char readChar(), double readDouble(), float readFloat(), int readInt(), boolean readBoolean() etc.
  - **ObjectInputStream** (implements java.io.ObjectInput, java.io.ObjectStreamConstants)
  - **PipedInputStream**
  - **AudioInputStream**

java.io class hierarchy (2/3)(selection)

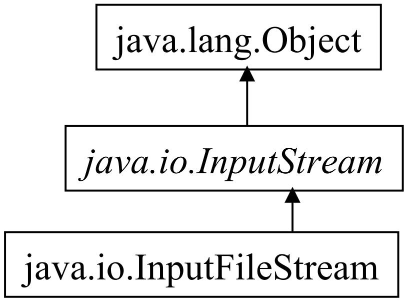
- ***OutputStream*** *(abstract class)*
  public abstract void write(int b): Writes the specified byte (the low 8 bits) to this output stream
  - **ByteArrayOutputStream**
  - **FileOutputStream**
  - **FilterOutputStream**
    - **BufferedOutputStream**
    - **DataOutputStream (**implements java.io.DataOutput)
      DataOutStream(OutStream);
      void writeInt(int); writes an int to the underlying output stream as four bytes, high byte first.
      void writeChar(int):writes a char to the underlying output stream as a 2-byte value, high byte first.
      writeUTF(String):writes a string to the underlying output stream using Java modified UTF-8 encoding in a machine-independent manner
      writeFloat(float): converts to int and writes as 4 bytes
    - **PrintStream (**adds functionality to another output stream, namely the ability to print representations of various data values conveniently)
      (The **PrintWriter** class should be used in situations that require writing characters rather than bytes)
      PrintStream(OutputStream);
      PrintStream(OutputStream out, boolean autoFlush, String encoding)
      print(int), println(int) …println(string)

      System.err and System.out are PrintStreams
  - **ObjectOutputStream** (implements java.io.ObjectOutput, java.io.ObjectStreamConstants)
  - **PipedOutputStream**
- **RandomAccessFile** (implements java.io.DataInput, java.io.DataOutput)

java.io class hierarchy (2/2) (selection)

- ***Reader*** *(abstract class for reading character streams)*
  abstract int read(char[] cbuf, int offset, int len); *Read characters into a portion of an array, starting from offset.*
  - **BufferedReader** (for line oriented operations)
    BufferedReader(Reader in)
    - **LineNumberReader**
  - **CharArrayReader**
  - **FilterReader**
    - **PushbackReader**
  - **InputStreamReader**
    InputStreamReader(InputStream in, Charset cs); **InputStreamReader**(InputStream in); (default charset)
    - **FileReader**
  - **PipedReader**
  - **StringReader**
- ***Writer***
  - **BufferedWriter**
  - **CharArrayWriter**
  - **FilterWriter**
  - **OutputStreamWriter**
    OutputStreamWriter(OutputStream out, CharsetEncoder enc) ; **OutputStreamWriter(OutputStream out)** ;
    - **FileWriter**
  - **PipedWriter**
  - **PrintWriter** (for line oriented operations)
  - **StringWriter**
- **StreamTokenizer**
  StreamTokenizer(Reader r);int nextToken();
  Fields: **double nval; String sval; int ttype** (type of token just read)
  static int TT_EOF, TT_EOL, TT_NUMBER, TT_WORD

Byte Streams



- Programs use *byte streams* to perform input and output of 8-bit bytes.
- All byte stream classes are descended from InputStream and OutputStream.
- If you use binary data, such as integers or doubles, then use the InputStream and OutputStream classes.

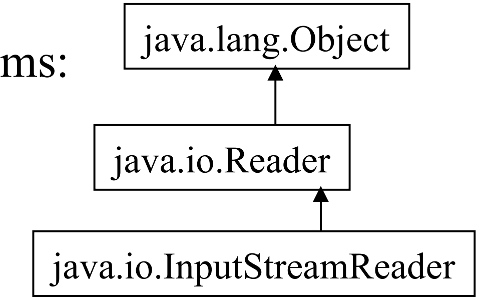
```
FileInputStream in = null;
FileOutputStream out = null;
in = new FileInputStream("xanadu.txt");
out = new FileOutputStream("outagain.txt");
int c;
while ((c = in.read()) != -1) out.write(c);
if (in != null) in.close();
if (out != null) out.close();
```

•Notice that read() returns an int value.
•Using a int as a return type allows read() to use -1 to indicate that it has reached the end of the stream.
    •Hex FFFFFFFF, eof
    •Hex 000000FF, char US-ASCII 127

Always Close Streams

Character Streams that Use Byte Streams

- There are two general-purpose byte-to-character "bridge" streams: InputStreamReader and OutputStreamWriter.
- Use them to create character streams when there are no prepackaged character stream classes that meet your needs.



- Some methods of InputStreamReader
  - Constructor:
    public **InputStreamReader**(InputStream in, String charsetName) throws UnsupportedEncodingException
    - Creates an InputStreamReader that uses the named charset.
      - charsetName - The name of a supported charset
  - public int **read**() throws IOException
    - » Reads a single character.
    - » **Returns:**
      The character read, or -1 if the end of the stream has been reached

Specificarea explicită a codării utilizate în operaţiile de intrare-ieşire cu caractere

```
import java.io.*;
public class SpecifyEncoding {
    public static void main(String[] args) throws Exception{
     FileInputStream byteIS=
                  new FileInputStream(args[0]+".in");
     InputStreamReader charIS=
                  new InputStreamReader(byteIS,"UTF-8");

     int i; int j=0;
     char[] arrch=new char[100];

     // reads chars in (UTF-8 encoding) in an array
     while ((i = charIS.read()) != -1) arrch[j++]=(char)i;
     if (charIS != null)charIS.close();

     java.awt.Frame f=new java.awt.Frame(String.copyValueOf(arrch));
     f.setVisible(true);
    }
  }
```

an input byte stream

*.in file is saved as UTF-8 encoding

an input char stream, with UTF-8 encoding

The title (containing Romanian diacritics) is properly displayed, only with charIS encoding UTF-8 specified

More encoding examples

```
BufferedReader in =
                            new BufferedReader(new InputStreamReader(System.in));
  System.out.print("Enter File name : ");
  String str = in.readLine();
  File file = new File(str);
    BufferedWriter out = new BufferedWriter(new OutputStreamWriter
                          (new FileOutputStream(file),"UTF8"));
  out.write("WelCome to RoseIndia.Net");
    out.close();
```

Writing in UTF-8 encoding

```
BufferedReader i =

    new BufferedReader(new InputStreamReader(new FileInputStream(file),"8859_1"));
        String str1 = i.readLine();
```

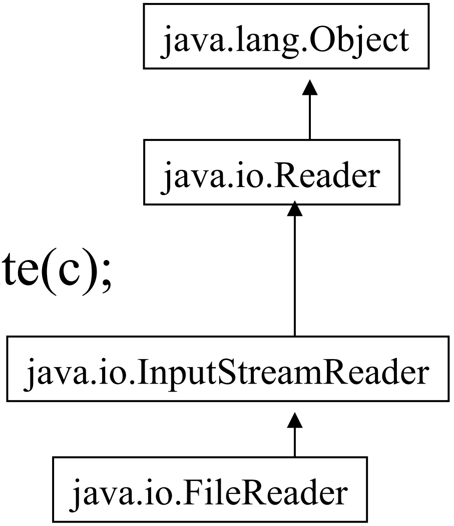
Reading ISO Latin-1

Encoded Data

Character Streams

- Character stream I/O automatically translates Unicode internal format to and from the local character set.
- In Western locales, the local character set is usually an 8-bit superset of ASCII.
- All character stream classes are descended from Reader and Writer

```
FileReader inputStream = null;
FileWriter outputStream = null;
inputStream = new FileReader("xanadu.txt");
outputStream = new FileWriter("characteroutput.txt");
int c;
while ((c = inputStream.read()) != -1) outputStream.write(c);
if (inputStream != null)   inputStream.close();
if (outputStream != null)  outputStream.close();
```



java.lang.Object

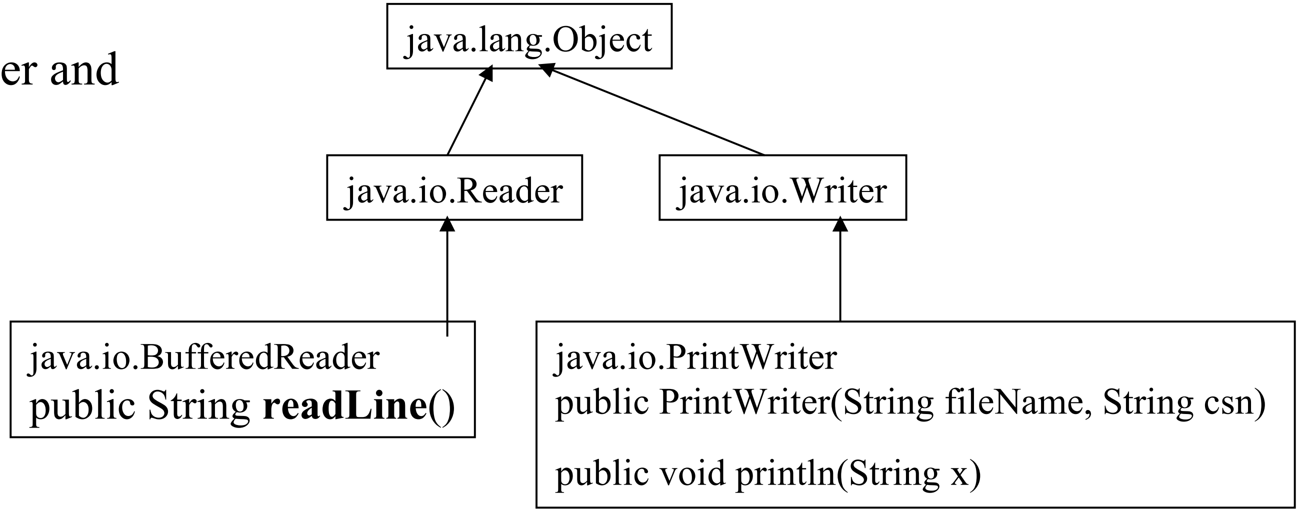
java.io.Reader

java.io.InputStreamReader

java.io.FileReader

Line-Oriented I/O

- Character I/O usually occurs in bigger units than single characters. One common unit is the line: a string of characters with a line terminator at the end.
- A line terminator can be a carriage-return/line-feed sequence ("\r\n"), a single carriage-return ("\r"), or a single line-feed ("\n").
- Supporting all possible line terminators allows programs to read text files created on any of the widely used operating systems.
- Classes:
  - BufferedReader and
  - PrintWriter


java.lang.Object

java.io.Reader

java.io.Writer

java.io.BufferedReader
public String **readLine**()

java.io.PrintWriter
public PrintWriter(String fileName, String csn)

public void println(String x)

Line Oriented IO, example

```
BufferedReader inputStream = null;
PrintWriter outputStream = null;
inputStream =
            new BufferedReader(new FileReader("xanadu.txt"));
outputStream =
            new PrintWriter(new FileWriter("characteroutput.txt"));
String s;
while ((l = inputStream.readLine()) != null)outputStream.println(l);
if (inputStream != null) inputStream.close();
if (outputStream != null) outputStream.close();
```

- appends the line terminator for the current operating system.
- This might not be the same line terminator that was used in the input file.

Scanning: Breaking Input into Tokens (1/2)

```
import java.io.*;
public class Tokenizer{
    static InputStreamReader instr; static BufferedReader br;
    static StreamTokenizer strTok;
    static InputStream istream;
    public static void main (String args[])throws IOException{
            istream=System.in; instr= new InputStreamReader(istream);
            br= new BufferedReader(instr); strTok= new StreamTokenizer(br);
            strTok.eolIsSignificant(false);
            System.out.println("Enter lines with numbers and identifiers:");
            System.out.println("Numbers colected, identifiers ignored.");
            while (strTok.nextToken()!= StreamTokenizer.TT_EOF){
              switch(strTok.ttype){
              case StreamTokenizer.TT_NUMBER:
                        System.out.print( strTok.nval +" ");break;
              case StreamTokenizer.TT_EOL:
                        System.out.println( "TT_EOL "); break;
              default: break;
              }
            }
            System.out.println("TT_EOF");// Ctrl-Z
    }
}
```

false: EOL treted as whitespace
true: EOL is a token (TT_EOL)

a double value always

Never executed for
strTok.eolIsSignificant(false);

Scanning: Breaking Input into Tokens (2/2)

- Objects of type **java.util.Scanner** are useful for breaking down formatted input into tokens and translating individual tokens according to their data type.
- By default, a scanner uses white space to separate tokens. (White space characters include blanks, tabs, and line terminators.

```
java.util.Scanner s = null;
double sum = 0;
try {s = new Scanner( new BufferedReader(new FileReader("usnumbers.txt")));
s.useLocale(Locale.US);
while (s.hasNext()) {
        if (s.hasNextDouble()) {sum += s.nextDouble();}
           else  s.next();
}
s.close();
System.out.println(sum);
```

Returns true if the next token in this scanner's input can be interpreted as a long value in the default radix using the nextLong() method.

Finds and returns the next complete token from this scanner.

Lesson: Basic I/O ????

Operatorul >>>

- Java are un operator nou: >>> (deplasare logică la dreapta):

```
int i=8
System.out.println(i>>>1);// 4
System.out.println(i>>>3);// 1
System.out.println(i>>>4);// 0
```

- O instrucţiune trebuie să aibă o acţiune explicită, nu poate fi doar o expresie aritmetică ce nu provoacă schimbări în context:

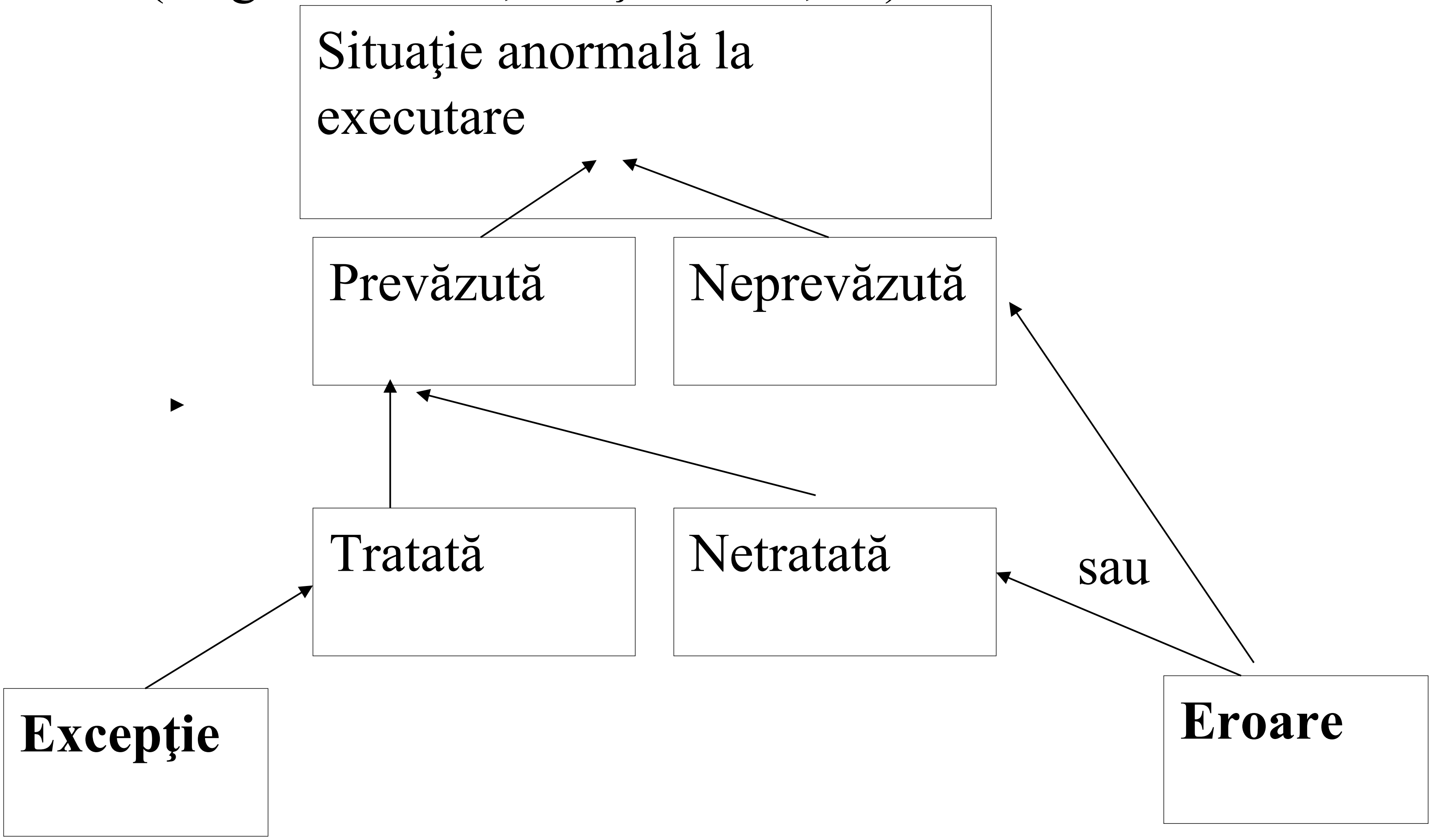
```
i=2*i;
i++;
++i;
2*i;// not a statement
i>>>1;// not a statement (??dar face o modificare!)
```

- ?? Condiţiile pot fi doar expresii de tip `boolean`, nu `int`

`if(i) i=0;// a mers sub` jdk1.5.0_06

Tratarea excepţiilor 1/4

Clasificarea situaţiilor anormale din timpul executării (diagrama UML, relaţia “is a”, →)



Tratarea excepţiilor 2/4

- Pentru tratarea excepţiilor, sunt necesare mecanisme de:
  - Definire excepţie (în Java, obiect de tip `java.lang.Exception`)
  - Identificare excepţie (prin condiţii logice a căror încălcare provoacă apariţia unei excepţii de un anumit tip)
  - Lansare excepţie (în java, `throw`)
  - Interceptare excepţie; la distanţă (textuală) de locul lansării (în Java, blocurile `try` şi `catch`)
  - Tratare excepţie; posibilitate de tratare diferenţiată atât de tipul excepţiei, cât şi de locul de interceptare (în Java, blocurile `try` şi `catch`)
- **Principiu:** Excepţiile **nu** trebuie tratate în metoda care le identifică şi lansează (server), ci în metoda care a activat metoda server (adică în metoda client, care la rândul său poate juca rolul de server)

  Limbajul Java: clauza `throws`:

```
server() throws Exception{
...
        if(identificare excepţie)throw new Exception();
...
}
client(){
        try{  server()
        }catch(Exception ex){
                tratare excepţie
        }
}
```

Tratarea excepţiilor 3/4

Exemplu. Clasa Stack Java\ExceptionStiva\stack.java

```
class Overflow extends Exception{// definire exceptie
	public Overflow(Stack o){source=o;}
	public Stack getSource(){return source;}
	private Stack source;

}
class Underflow extends Exception{
	public Underflow(Stack o){source=o;}
	public Stack getSource(){return source;}
	private Stack source;
}
public class Stack{
	public Stack(int n, String s){
			dim=n; name=s; top=-1;
			v= new Object[dim];
	}
```

```
	public void insert(Object o) throws Overflow{
	//serverul nu trateaza
	//exceptiile pe care le lanseaza!
			if (top < dim-1) v[++top]=o;
			else throw new Overflow(this);
			// identificare si lansare
	}
	public void delete() throws Underflow{
			if(top >=0) top--;
			else throw new Underflow(this);
	}
	public Object item() throws Underflow{
			if(top >=0) return v[top];
			else throw new Underflow(this);
	}
	public String getName(){return name;}


	private Object v[];
	private int dim;
	private int top;
	private String name;
}
```

Tratarea excepţiilor 4/4

- StackDriver Java\ExceptionStiva\StackDriver.java

```
public class StackDriver{
   public static void main(String
   args[]){
     Stack s1,s2;
     s1=new Stack(2, "stiva Tudor");
     s2=new Stack(2, "stiva Andrei");
     try{
           s1.insert("aa");
           s1.delete();
           s1.item();//Underflow
     }catch(Overflow e){//interceptare
           //tratare
           System.out.println("Overflow
   la "               +
              e.getSource().getName());
     } catch (Underflow e){

   System.out.println("Underflow la "
                      +
              e.getSource().getName());
     }
```

```
    try{
            s1.insert("aa");
            s1.delete();
            s2.insert("aa");
            s2.insert( new
    Integer(10));
            s2.insert("aa");
    //Overflow
    } catch (Overflow e){
            System.out.println("Overflow
    la "              +
                 e.getSource().getName());
    } catch (Underflow e){
            System.out.println("Underflow
    la "              +
                 e.getSource().getName());
    }

   }

}
```

Moştenire simplă
Interfaţă (`interface`) 1/4

- În plus faţă de carcteristicile moştenite de la clasa Object, în Java, o subclasă mai poate moşteni caracteristici **de la cel mult o** altă superclasa.
- Eventualele dificultăţi de proiectare provocate de renunţarea la moştenirea multiplă sunt depăşite prin utilizarea conceptului *interface* (interfaţă).
- Cuvîntul cheie `interface` introduce o construcţie sintactică prin care se **specifică** signatura unor metode **publice şi abstracte** ce urmează a fi **implementate** în alte clase.

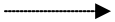
```
public interface InputOutput{
          public abstract String read();
          public abstract void print(String s);
}
//specificarea public abstract  e singura posibila
//si se poate omite
```

- Conceptul se deosebeşte de clasa abstractă deorece:
  - Nu poate conţine implementări de metode
  - Nu conţine atribute (deci nu se rezervă memorie); dar poate conţine definiţii de constante: `...static final...`
  - Moştenirea multiplă este aplicabilă în cazul interfeţelor (remarcaţi că absenţa atributelor face ca problema atributelor în copie să nu mai apară!)
- Pot fi declarate referinţe de tip `interface`:

```
InputOutput consola;
```

- Pentru a fi utilă, o interfaţă trebuie implementată de una sau mai multe clase. Cuvantul cheie este `implements` iar în diagrame UML relaţia este   O clasă poate implementa mai multe interfeţe (dar specializează cel mult o clasă!)

```
class C extends OSinguraClasa, implements InputOutput, AltaInterfata{
          public String read(){…aici se implementeaza read…}
          public print( String s){…aici se implementeaza print…}

          //aici se implementeaza obligatoriu toate metodele din AltaInterfata
}
```



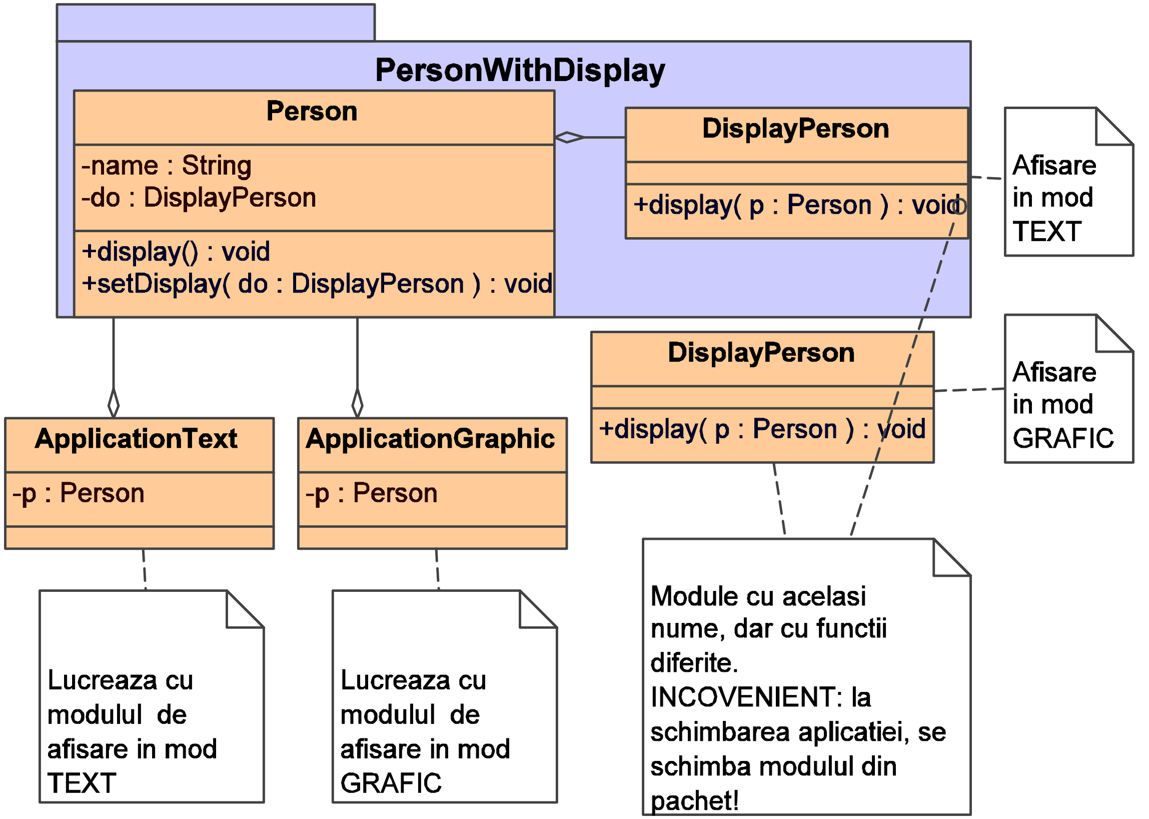
- O referinţă de tip interface paote referi orice obiect dintr-o clasa ce implementează interfaţa:

```
consola=new C();
```

Moştenire simplă
Interfaţă (`interface`) 2/4

- Exemplu. Utilizarea interfeţelor.
  - Proiectul clasei `Person,` include în interfaţa clasei `metoda void display().` Implementarea acestei metode este însă dependentă de contextul în care utilizăm clasa `Person.` De pildă, în cazul în care dorim ca afişarea să se facă într-un fişier în loc de monitor, va trebui să modificăm implementarea acestei metode.
  - Deoarece motivul modificării nu este unul intrinsec obiectelor "persoană" ci mai degrabă unul colateral, proiectul clasei este considerat că încalcă principiul *"Single Resposability Principle (SRP)": a class should have only a single reason to change.*
    Cu alte cuvinte, clasa Person are în sarcină responsabilităţi colaterale, (afişarea).
  - Pentru a reduce responsabilităţile clasei , `Person` proiectăm o clasă specială `DisplayPerson` ale cărei obiecte sunt utilizate pentru afişarea datelor despre persoane. Pentru aceasta, orice obiect `Person` este agregat cu un obiect `DisplayPerson.` Modificarea contextului de afişare va afecta de acum încolo doar clasa `DisplayPerson.` Proiectul este prezentat în diagrama următoare.

Diagrama claselor Person şi DisplayPerson (fară interfaţă)



Module cu acelasi nume, dar cu functii diferite.
INCOVENIENT: la schimbarea aplicatiei, se schimba modulul din pachet!

Person şi DisplayPerson, agregate direct (afişare în mod text)

Java\Person\SRPNONOCP\Person.java

```
public class Person{
   public Person(String s){name=s;}
   public void display(){display.display(this);}
   public String getName(){return name;}
   public void setDisplay(DisplayPerson dp){display=dp;}
   protected String name;
   protected DisplayPerson display=new DisplayPerson();
}
class DisplayPerson{
   public void display(Person p){ System.out.println(p.getName());}
}
Java\Person\SRPNONOCP\TestDriver.java
public class TestDriver{
   public static void main(String[] args){
   Person p=new Person("Tudor");
   p.setDisplay(new DisplayPerson());
   p.display();
   }
}
```

Person şi DisplayPerson, agregate direct
(afişare în mod grafic)

Java\Person\SRPNONOCP\Another DisplayPerson\DisplayPerson.java

```
import java.awt.*;
public class DisplayPerson extends Frame{
   public DisplayPerson(){
          setTitle("Display a Person");
          tf=new TextField(20);
          add("Center",tf);
          tf.setText("XXXXXXXXXX");
          setSize(100,250);
          //setVisible(true);

   }
   public void display(Person p){
          tf.setText(p.getName());
          setVisible(true);
   }

   TextField tf;
}
```

Codul DisplayPerson.class
va înlocui pe cel care
face afişare în mod text

Principiul Open Closed Principle (OPC)

- Soluţia de agregare a claselor `Person` şi `DisplayPerson` are însă alt neajuns: la modificarea clasei `DisplayPerson`, fişierul `DisplayPerson.class,` din pachetul în care se află clasa Person, trebuie substituit prin noua versiune, care are însă acelaşi nume. Schimbarea tipului de aplicaţie, chiar dacă nu mai necesită schimbarea clasei `Person,` conduce la schimbări în pachetul in care se află această clasă, în loc să conducă la schimbări in pachetul aplicaţiei. În exemplul anterior, serverul `DisplayPerson` a fost substituit cu altul, cu acelaşi nume, care face afişare într-o fereastră Windows.
- Acesta este un simptom al unei legături prea directe între clientul `Person` şi serverul `DisplayPerson`. Este o încălcare a principiului OPC: *open for extension, closed for modification: it should be possible to change the environment without changing the class.*
- O slăbire a conexiunii tari impuse prin agregarea claselor `Person` şi `DisplayPerson` se poate face prin interpunerea unei interfeţe între cele două clase, ca în diagarma următoare.
- Schimbarea modului de afişare se face acum natural, prin **adăugarea,** în pachetul aplicaţiei TestDriver, a unor clase ce implementează interfaţa DisplayInterface şi modificarea corespunzătoare a aplicaţiei (linia `p.setDisplay(new DisplayPerson());//***`). Noile clase au un alt nume, diferit de DisplayPerson şi ele pot coexista în acelaşi pachet.
- **Observaţie.**

> *Utilizarea unei clase abstracte în locul interfeţei `DisplayInterface` ar restrânge domeniul de aplicabilitate.* Într-o *astfel de situaţie, clasa `DisplayPerson Graphic` nu mai poate fi utilizată deoarece ar trebui să specializeze atât clasa abstractă cât şi clasa `java.awt.Frame` (moştenire multiplă).*

Person şi DisplayPersonText, cu interfaţa DisplayInterface între ele

Java\Person\OCP\Person.java

```
public class Person{
    Person(String s){name=s;}
    public void display(){display.display(this);}
    public String getName(){return name;}
    public void setDisplay(DisplayInterface dp){display=dp;}
    protected String name;
    protected DisplayInterface display;
}
```

Java\Person\OCP\DisplayInterface.java

```
public interface DisplayInterface{
    public void display(Person p);
}
```

Java\Person\OCP\DisplayPerson.java

```
public class DisplayPersonText implements DisplayInterface{
    public void display(Person p){ System.out.println(p.getName());}
}
Java\Person\OCP\TestDriver.java
public class TestDriver{
    public static void main(String[] args){
            Person p=new Person("Tudor");
            p.setDisplay(new DisplayPersonText());//***
            p.display();
    }
}
```

Person şi DisplayPersonGraphic, cu interfaţa DisplayInterface între ele

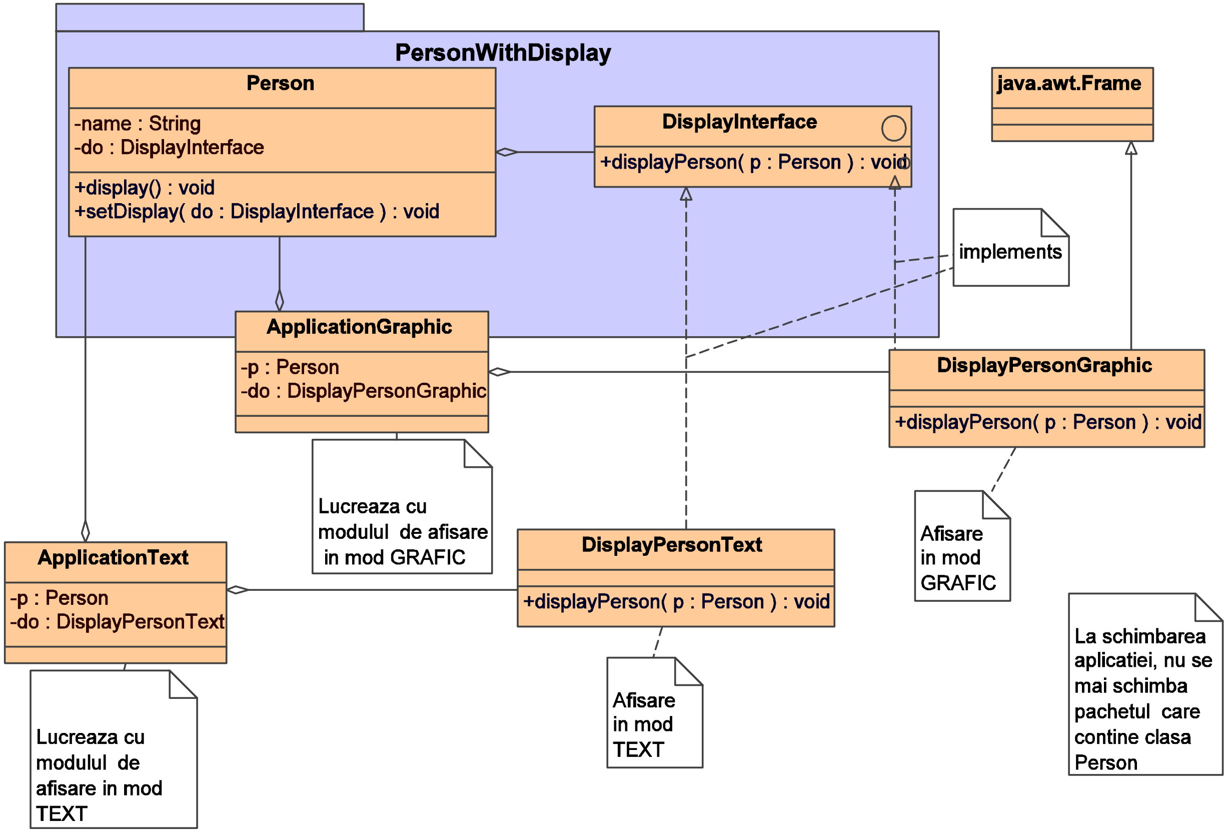
Java\Person\OCP\DisplayPersonGraphic.java

```
import java.awt.*;
public class DisplayPersonGraphic
    extends Frame
implements DisplayInterface{
    public DisplayPersonGraphic(){
    setTitle("Display a Person");
    tf=new TextField(20);
    add("Center",tf);
    tf.setText("XXXXXXXXXX");
    setSize(100,250);
    //setVisible(true);

    }
    public void display(Person p){
    tf.setText(p.getName());
    setVisible(true);
    }

    TextField tf;
}
```

Java\Person\OCP\TestDriverGraphic.java

```
public class TestDriverGraphic{
    public static void main(String[]
    args){
              Person p=new
    Person("Tudor");
              p.setDisplay(new
    DisplayPersonGraphic());
              p.display();
    }
}
```

Interfaţa DisplayInterface: diagrama claselor


PersonWithDisplay
Person
-name : String
-do : DisplayInterface
+display() : void
+setDisplay(do : DisplayInterface) : void
DisplayInterface
+displayPerson(p : Person) : void
java.awt.Frame
implements
ApplicationGraphic
-p : Person
-do : DisplayPersonGraphic
DisplayPersonGraphic
+displayPerson(p : Person) : void
Lucreaza cu modulul de afisare in mod GRAFIC
Afisare in mod GRAFIC
ApplicationText
-p : Person
-do : DisplayPersonText
DisplayPersonText
+displayPerson(p : Person) : void
Afisare in mod TEXT
La schimbarea aplicatiei, nu se mai schimba pachetul care contine clasa Person
Lucreaza cu modulul de afisare in mod TEXT

Clase adaptoare (adapters)

Callback (1/2)

Jerry wrote:

> Can anyone explain to me what is callback method? Any example will be
> highly appreciated.

its when you 'register' a method with some other object, which that object can then call back, when it likes..However, in Java there is no way to 'register a method' (i.e. pass a function pointer like in C/C++).

In Java, you pass one object to the other, then the other object calls the method of the object back.

When designing, its usually a very good idea to use interfaces rather than Classes as this reduces the number of methods that the other class has access to .

```
interface CallBack {
void methodToCallBack();
}
```

Callback (2/2)

```
class CallBackImpl implements CallBack {

    public void methodToCallBack() {
    System.out.println("I've been called back";
    }
    }

    class Caller {

    public register(CallBack callback) {
    callback.methodToCallBack();
    }

    public static void main(String[] args) {

    Caller caller = new Caller();
    CallBack callBack = new CallBackImpl();

    caller.register(callBack);
    }

    }
```

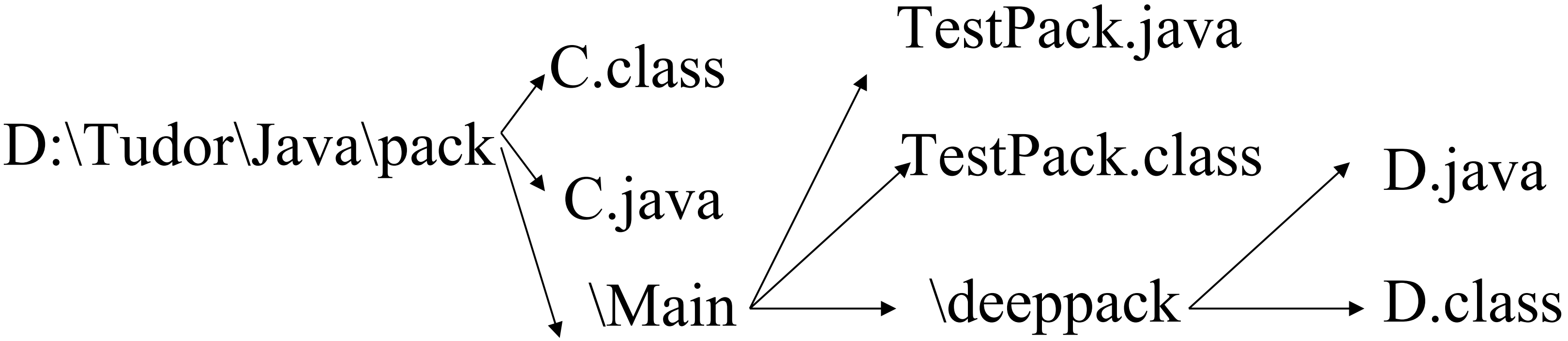
A simple example, but shows the mechanics.

Its a very powerful technique, used by various design patterns.

Google 'Visitor Pattern' for a starter...

Pachete (1/3)

- Clasele pot fi grupate în biblioteci, numite pachete (package).
- Numele unui pachet este un sufix oarecare de drum (path) către un `directory` ce conţine clasele care vor fi incluse în pachet. În nume apare caracterul . în loc de \.
- Includerea claselor unui fişier într-un pachet se face prin directiva package `<nume de pachet>;` la compilare, fişierul trebuie sa fie în directorul `<prefix director>;` `<nume de pachet>;` clasele rezultate for fi în acest mod memorate chiar în directorul care da numele pachetului.
- Utilizarea claselor dintr-un pachet se face prin directiva import `<nume de pachet>.*`
- Se poate utiliza o singură clasă `C`, prin import `<nume de pachet>.C`
- Înainte de compilare, variabila CLASSPATH va trebui să conţină valoarea `<prefix director>;` din CLASSPATH şi `<nume de pachet>;` compilatorul va reconstitui calea catre directorul de clase importate.
- Considerăm următoarea structură arborescentă de fişiere şi repertoare (directories)



Pachete (2/3)

Pentru compilarea şi interpretarea programului din fişierele următoare:

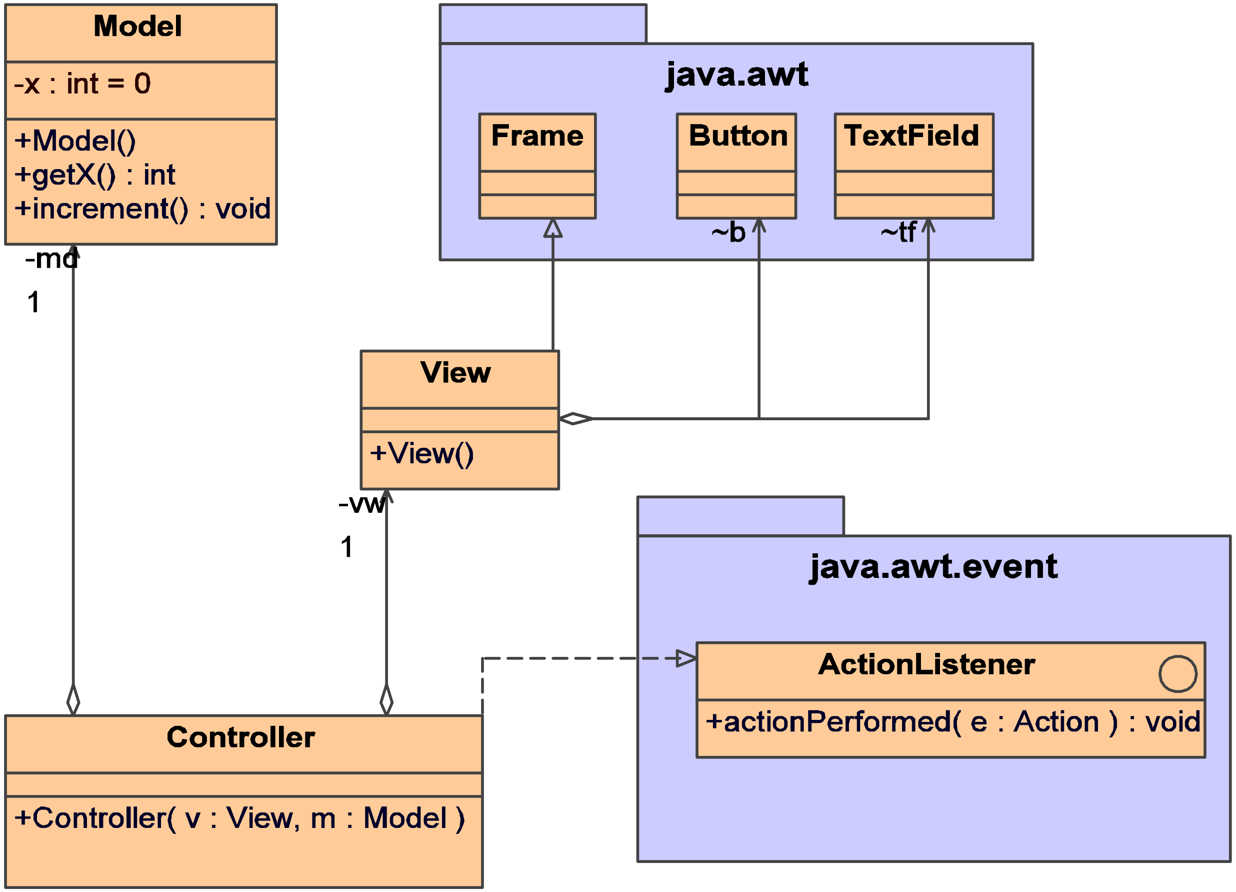
```
//file D:\Tudor\java\pack\C.java
package pack;
public class C{
    public void m(){System.out.println("class C from D:\\Tudor\\JAVA2006\\pack");}
}

// file D:\Tudor\java\pack\Main\deeppack\D.java
package deeppack;
public class D{
    public void m(){System.out.println("class D from D:\\Tudor\\JAVA2006\\Main\\
    Pack");}
}

// file D:\Tudor\java\pack\Main\TestPack.java
import pack.*;
import deeppack.*;
public class TestPack{
    public  static void main(String[] args){
              C c= new C();
              c.m();
              D d= new D();
              d.m();
    }
}
```

Pachete (3/3)

- Se execută următoarele comenzi:

```
>D:
>cd D:\Tudor\java\pack
>javac C.java
>cd D:\Tudor\java\pack\Main\deeppack
>javac D.java
>cd D:\Tudor\java\pack\Main
>rem urmeaza prefixele pachetelor
>set CLASSPATH= %CLASSPATH%; D:\Tudor\java\;
>set CLASSPATH= %CLASSPATH%; D:\Tudor\java\pack\Main\
>javac TestPack.java
>java TestPack
```

Arhitectura Model View Controller (1/3)



Model View Controller (2/3)

```
Java\MVC\Model.java
// file Model.java
public class Model{
          private int x=0;
          public Model(){};
          public void increment(){x++;}
          public int get_x(){return x;}
}
```

```
•Java\MVC\View.java
// file View.java
import java.awt.*;
public class View extends Frame{
          Button b;
        TextField tf;
          public View(){
                    setTitle("Exemplu Model-View-Controller");

                    b= new Button("Actiune");
                    add("North",b);

                    tf=new TextField(10);
                    add("Center",tf);

                    setSize(100,250);
                    setVisible(true);
          }
}
```

Model View Controller (3/3)

Java\MVC\Controller.java

```
// file Controller.java
import java.awt.event.*;
public class Controller implements ActionListener{

          public Controller(View v, Model m){
                    vw=v;md=m;
                    v.b.addActionListener(this);
          }
          public void actionPerformed(ActionEvent e){
                    md.increment();
                vw.tf.setText(String.valueOf(md.get_x()));
          }
          private View vw;
          private Model md;
}
```

```
Java\MVC\MVC.java
// file MVC.java
public class MVC{
          public static void main(String[] args){
                    View v=new View();
                    Model m= new Model();
                    Controller c= new Controller(v,m);
          }
}
```

Observer-Observable

Observer-Observable

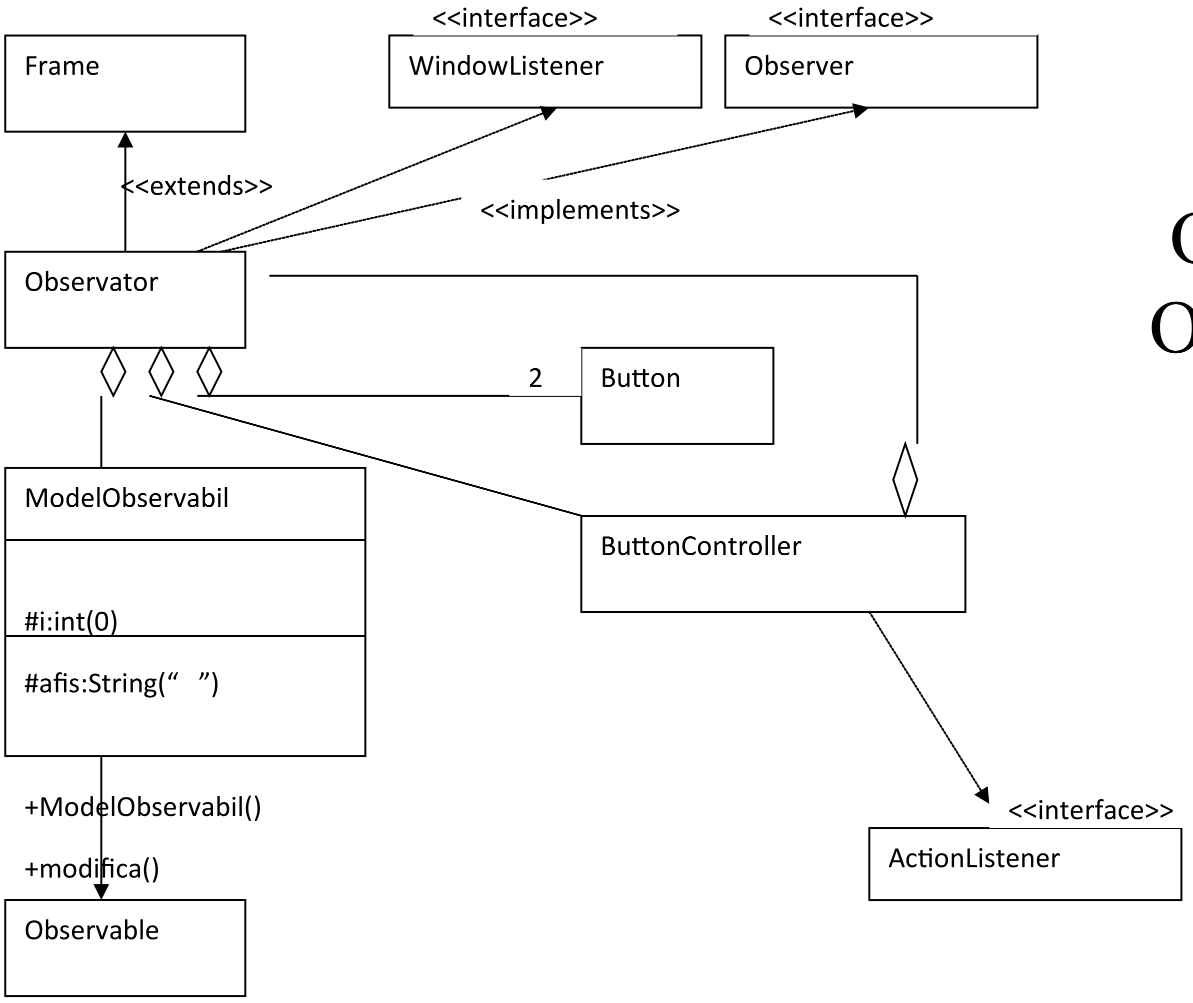
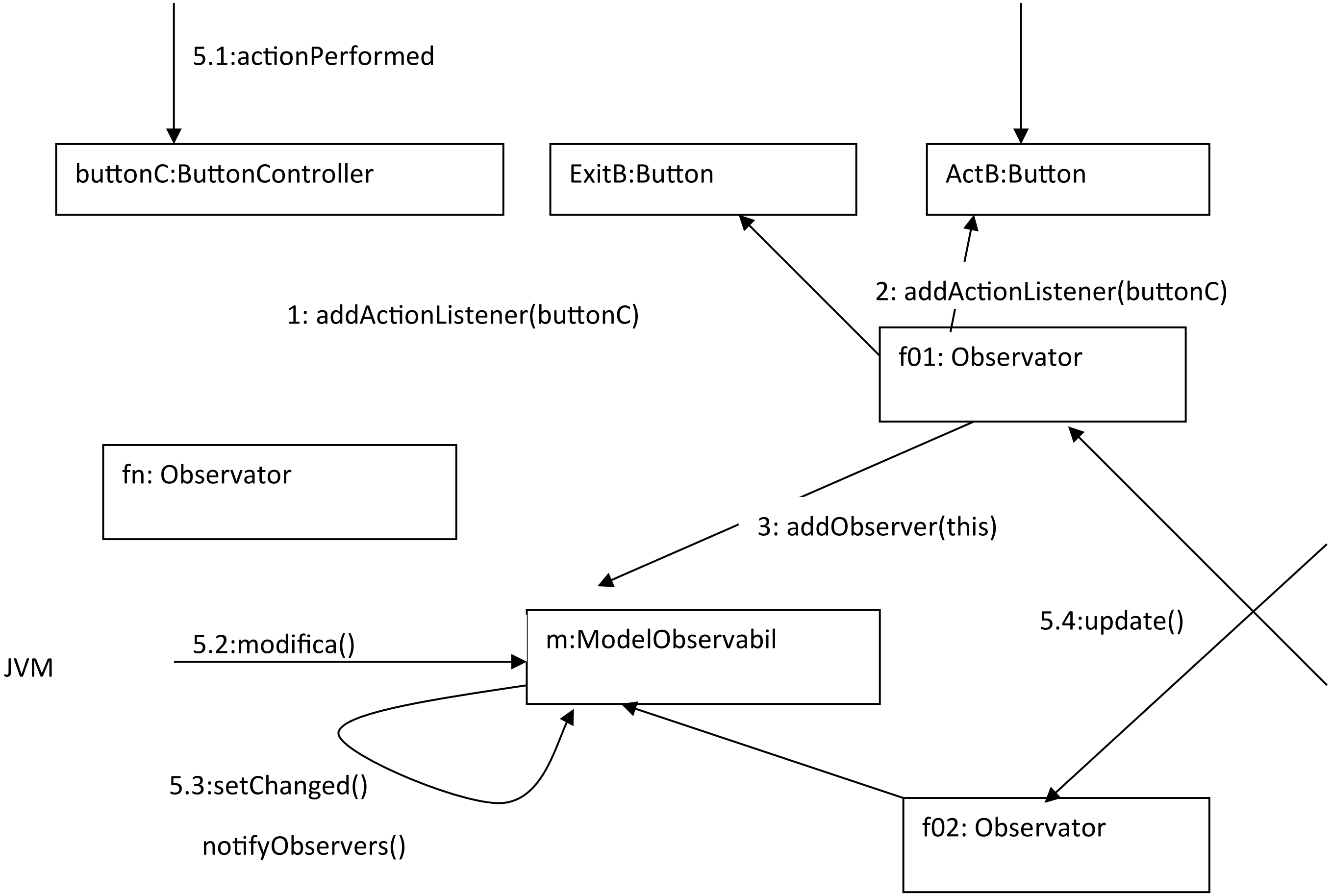


Diagrama de clase

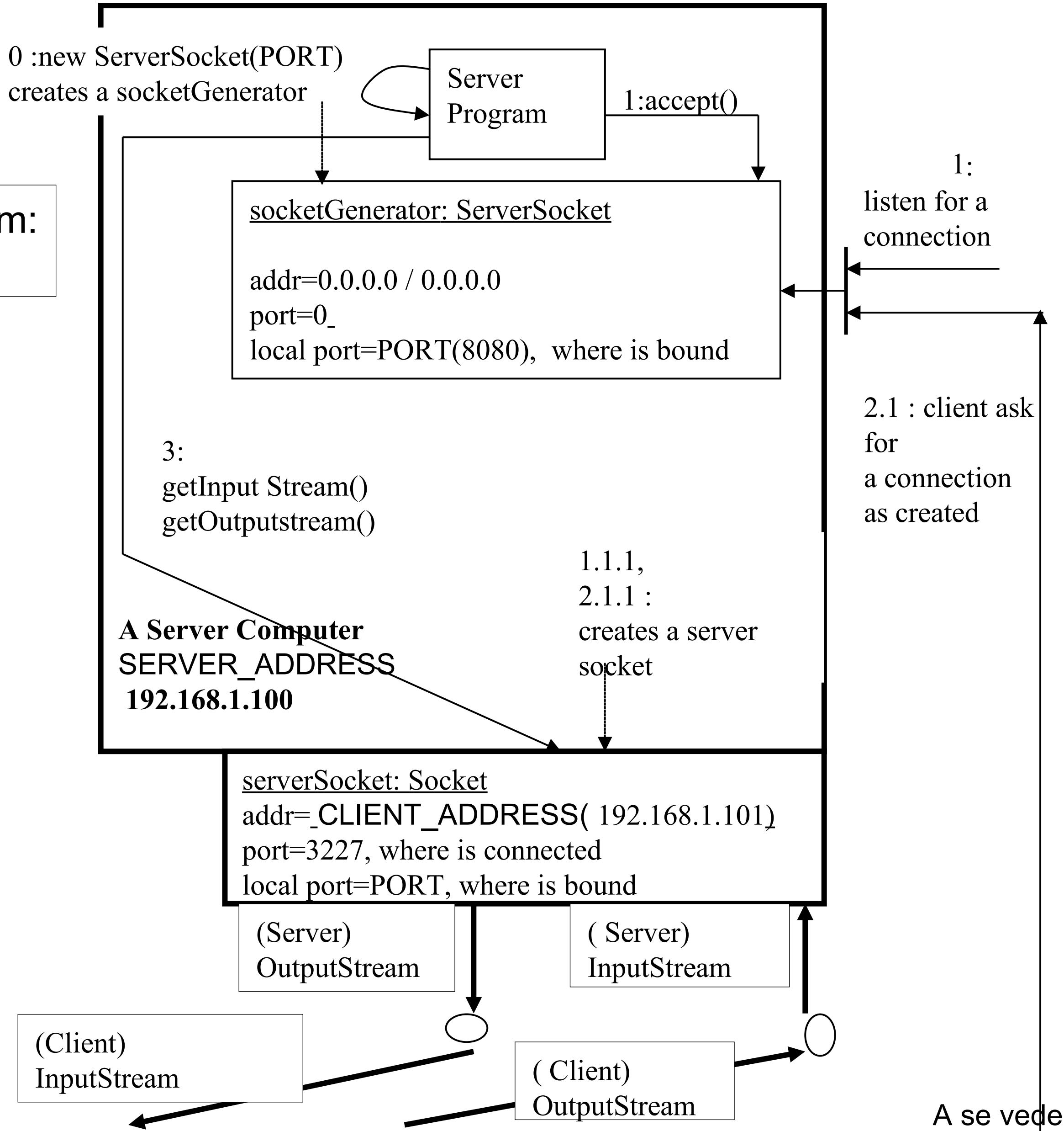


5.1:actionPerformed
buttonC:ButtonController
ExitB:Button
ActB:Button
1: addActionListener(buttonC)
2: addActionListener(buttonC)
f01: Observator
fn: Observator
3: addObserver(this)
JVM
5.2:modifica()
m:ModelObservabil
5.4:update()
5.3:setChanged()
notifyObservers()
f02: Observator

-
- //Trei ferestre si un model observabil m
- // Doar fo1 si fo2 sunt observatori ai lui m // fn nu este observator
- // Apasarea butonului modifica din oricare cele trei ferestre // incrementeaza cu 1 un atribut al modelului
- // Dar numai fo1 si fo2 sesizeaza acest lucru.
- SEE
- C:\TUDOR\JAVA\Observator

Comunicaţii Java, protocol TCP (1/2)

Lansare Server Program:
>java Server PORT



0 :new ServerSocket(PORT)
creates a socketGenerator

Server
Program

1:accept()

socketGenerator: ServerSocket

addr=0.0.0.0 / 0.0.0.0
port=0
local port=PORT(8080), where is bound

1:
listen for a
connection

2.1 : client ask
for
a connection
as created

3:
getInput Stream()
getOutputstream()

1.1.1,
2.1.1 :
creates a server
socket

A Server Computer
SERVER_ADDRESS
192.168.1.100

serverSocket: Socket
addr=CLIENT_ADDRESS(192.168.1.101)
port=3227, where is connected
local port=PORT, where is bound

(Server)
OutputStream

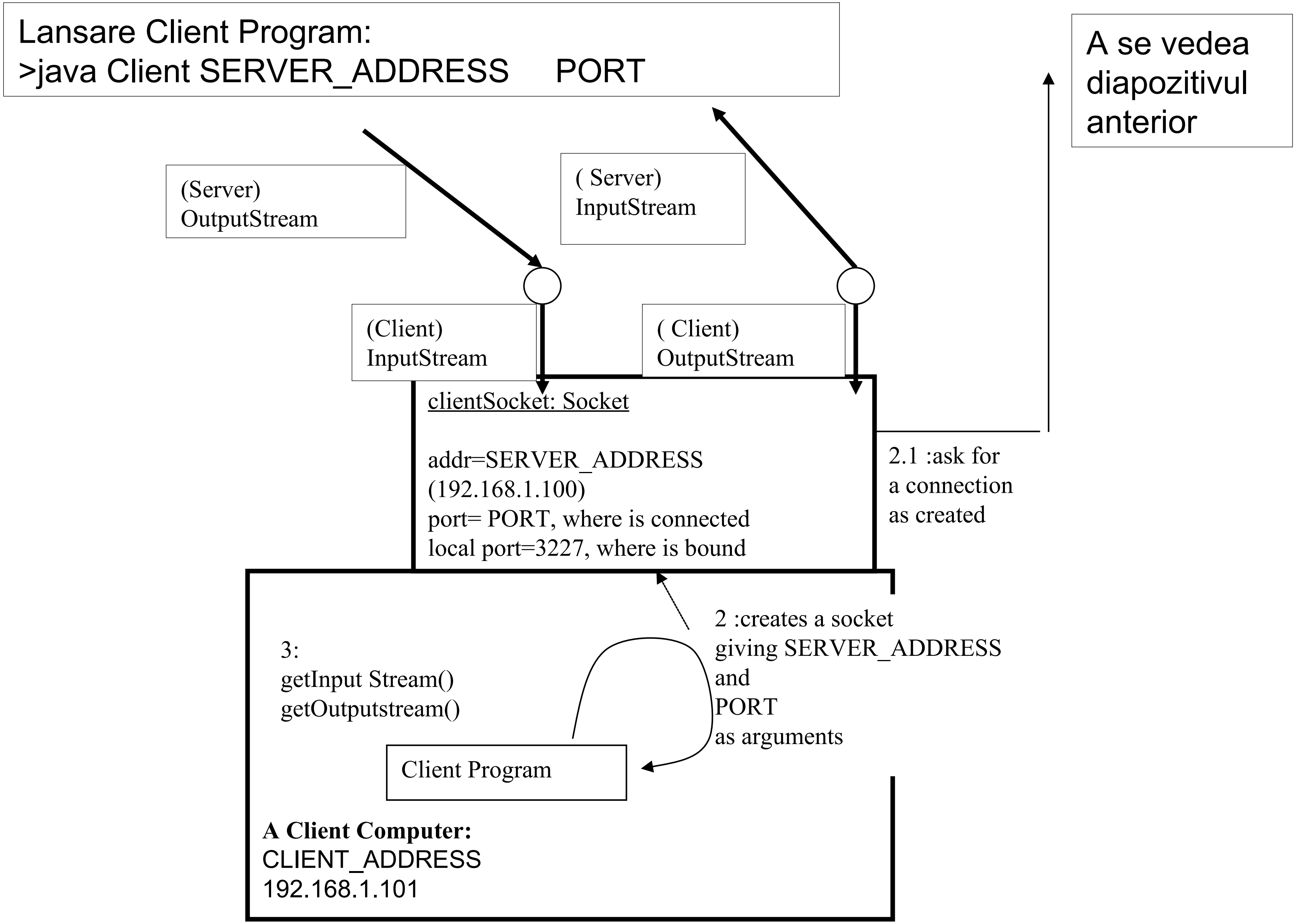
(Server)
InputStream

(Client)
InputStream

(Client)
OutputStream

A se vedea
diapozitivul următor

Comunicaţii Java, protocol TCP (2/2)



Lansare Client Program:
>java Client SERVER_ADDRESS PORT

A se vedea diapozitivul anterior

(Server) OutputStream

(Server) InputStream

(Client) InputStream

(Client) OutputStream

clientSocket: Socket

addr=SERVER_ADDRESS
(192.168.1.100)
port= PORT, where is connected
local port=3227, where is bound

2.1 :ask for a connection as created

2 :creates a socket giving SERVER_ADDRESS and PORT as arguments

3:
getInput Stream()
getOutputstream()

Client Program

A Client Computer:
CLIENT_ADDRESS
192.168.1.101

A Server Program: Varianta simplă: un server, un client (1/2)

Java\Socket\SingleClient\Server.java

```
import java.io.*;
import java.net.*;
public class Server{
   static  int PORT=8080;
   public static void main(String[] args)throws IOException{
   if(args.length!=0)PORT= Integer.parseInt(args[0]);
   ServerSocket socketGenerator=new ServerSocket(PORT);
   System.out.println("Created ServerSocket generator: ‟
   + socketGenerator);
   // Aceasta adresa va fi indicata de client pentru conectare
   System.out.println("Server address to be used by clients: ‟
   + InetAddress.getLocalHost());
   try{
      System.out.println("Waiting for a connection! ");
      Socket serverSocket=socketGenerator.accept();
   // blocks until a connection occurs
      System.out.println("Connection accepted, the following   server
   Socket was created: "
   + serverSocket);
```

A Server Program(2/2)

```
                 try{

                        BufferedReader in=
                          new     BufferedReader(new

    InputStreamReader(serverSocket.getInputStream()));
                        BufferedWriter bw=
                          new     BufferedWriter(new

    OutputStreamWriter(serverSocket.getOutputStream()));
                        PrintWriter out=
                          new PrintWriter(bw,true);
                      out.println("Your Password, please: " );
                      String str=in.readLine();
                      if( str.equals("password"))out.println("Password Accepted!");
                      else out.println("Wrong Password");

                 }finally{
                        System.out.println("Closed client socket: " +
    serverSocket);
                        serverSocket.close();
                 }
              }finally{
                        System.out.println("Close socketGenerator: "
                                                                + socketGenerator);
                        socketGenerator.close();
              }
    }
}
```

A Client Program (1/2)

- Java\Socket\SingleClient\Client.java

```
import java.net.*;
import java.io.*;
public class Client{

   static InetAddress addr;// adresa (implicit, gazda clientului) a calculatorului
    server la care se va conecta
   static  int PORT=8080;// si portul (implicit 8080) de conectare al serverului
/**     Adresa serverului si portul PORT vor fi date ca argumente in linia de
    comanda
    Se specifica amandoua sau se omit amandoua!
    Daca nu sunt specificate, serverul trebuie sa fie pe aceeasi masina cu clientul
    iar portul este 8080.
    Atentie la conectare: programe precum Norton Internet Security  blocheaza
    comunicarea
*/
   public static void main(String[] args) throws IOException {
    // adresa client:
    System.out.println("Client InetAddress: " + InetAddress.getLocalHost());
    addr= InetAddress.getLocalHost();
    if(args.length==2){
              addr=InetAddress.getByName (args[0]);
              PORT=Integer.parseInt(args[1]);
    }
    System.out.println("Called Server InetAddress: " + addr+ ", specified port: "+
    PORT);
    Socket clientSocket= new Socket( addr, PORT);
    System.out.println("Created a client socket:" + clientSocket);
```

A Client Program (2/2)

```
    try{
              BufferedReader in=
                        new       BufferedReader(new

    InputStreamReader(clientSocket.getInputStream()))
              BufferedWriter bw=
                        new       BufferedWriter(new

    OutputStreamWriter(clientSocket.getOutputStream()));        PrintWriter out=
                        new PrintWriter(bw,true);

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

              System.out.println(in.readLine());
              out.println(consoleIn.readLine());
              System.out.println(in.readLine());

    }finally{
              System.out.println("The   client socket:" +
                        clientSocket +" is going to be closed");
                        clientSocket.close();
    }
   }
}
```

MultiClient (1/5)

- Java\Socket\MultipleClient with Password\MultiClientServer.java

```
    import java.io.*;
import java.net.*;
public class MultiClientServer{
    static  int PORT=8080;
    public static void main(String[] args) throws Exception{
              Thread aClient;
              if(args.length!=0)PORT= Integer.parseInt(args[0]);
              ServerSocket socketGenerator=new ServerSocket(PORT);
              System.out.println("Created ServerSocket generator: " +
    socketGenerator);
              // Aceasta adresa va fi indicata de client pentru conectare
              System.out.println("Server address to be used by clients: "+
    InetAddress.getLocalHost());
              try{
                 while (true){
                        System.out.println("Waiting for a connection! ");
                        Socket serverSocket=socketGenerator.accept(); // blocks
    until a connection occurs
                        System.out.println("Connection accepted, the following
    server Socket was created: "
                                  + serverSocket);
                        aClient=new ServeOneClient(serverSocket);
                        aClient.start();
                 }
              }finally{
                        System.out.println("Close socketGenerator: " +
    socketGenerator);
                        socketGenerator.close();}
              }
}
```

Multiclient (2/5)

-
  ```
  class ServeOneClient extends Thread{
  private Socket serverSocket;
  private BufferedReader in;
  private BufferedWriter bw;
  private PrintWriter out;
  public ServeOneClient(Socket s) throws IOException{
     serverSocket=s;
     in=new BufferedReader(new InputStreamReader(serverSocket.getInputStream()));
     bw=new BufferedWriter(new 
  OutputStreamWriter(serverSocket.getOutputStream()));
     out=new PrintWriter(bw,true);
  }
  ```

Multiclient (3/5)

```
    public void run() {
       String str;
       try{
       int correctpsw=0;
       out.println("Your password, please: " );
       for (int i=0; i<3 ; i++){
          str=in.readLine();
          if( str.equals("password")){correctpsw=1; out.println("CORRECT!"); break;}
          else out.println("WRONG: "+ (2-i) +" allowed tries!" );
       }
    if(correctpsw==1){
       out.println("Enter lines of text, please! The last one, END!");
       out.println("You will be echoed with a UPPER CASE COPY!");
       while(true){
          try {sleep(100);} catch(InterruptedException i){}
          str=in.readLine();
          out.println("Echoing: " + str.toUpperCase());
          if(str.equals("END")) break;
       }
    }
    System.out.println("Closing a client thread...");
       }catch (IOException e){
       }finally{
    try{
    System.out.println("Closed client socket: " + serverSocket);
    serverSocket.close();
    }catch(IOException e){}
       }
    }

}
```

Multiclient (4/5)

Java\Socket\MultipleClient with Password\Client.java

```
import java.net.*;
import java.io.*;
public class Client{

   static InetAddress addr;// adresa (implicit, gazda clientului) a calculatorului server la care se va conecta
   static  int PORT=8080;// si portul (implicit 8080) de conectare al serverului
/**     Adresa serverului si portul PORT vor fi date ca argumente in linia de comanda
    Se specifica amandoua sau se omit amandoua!
    Daca nu sunt specificate, serverul trebuie sa fie pe aceeasi masina cu clientul
    iar portul este 8080.
    Atentie la conectare: programe precum Norton Internet Security  blocheaza  comunicarea
*/
   public static void main(String[] args) throws IOException {
    // adresa client:
    System.out.println("Client InetAddress: " + InetAddress.getLocalHost());
    addr= InetAddress.getLocalHost();
    if(args.length==2){
    addr=InetAddress.getByName (args[0]);
    PORT=Integer.parseInt(args[1]);
    }
    System.out.println("Called Server InetAddress: " + addr+ ", specified port: "+ PORT);
    Socket clientSocket= new Socket( addr, PORT);
    System.out.println("Created a client socket:" + clientSocket);


    BufferedReader in=
    new       BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
    BufferedWriter bw=
    new       BufferedWriter(new OutputStreamWriter(clientSocket.getOutputStream()));
    PrintWriter out=
    new PrintWriter(bw,true);

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


    // parola
```

Multiclient (5/5)

```
System.out.println(in.readLine());// From server: Your password...
    String answer="WRONG";
    for (int i=0; i<3 && answer.startsWith("WRONG") ; i++){
    out.println(consoleIn.readLine());// trimite parola
    System.out.println(answer=in.readLine()); // primeste raspuns
    }
    // answer contine CORRECT sau WRONG (daca s-au facut mai mult de 3 incercari)
    if (answer.startsWith("CORRECT")){
    System.out.println("Start of processing");
    System.out.println(in.readLine());// From server: enter lines
    System.out.println(in.readLine());// From server: You will be...

    // comunicare
    String fromConsole;
    do{
    out.println(fromConsole=consoleIn.readLine());// trimite date
    System.out.println(in.readLine()); // primeste date

    }while (! fromConsole.toUpperCase().equals("END"));
    // ATENTIE, NU (fromConsole.toUpperCase() !="END");
    }else     System.out.println("No processing");
    System.out.println("The   client socket:" +
    clientSocket +" is going to be closed");
    clientSocket.close();

     }
}
```

Threads

- **Thread mechanism is used to execute multiple tasks at the same time.**
- **Thread Scheduling**
  - **When we say that threads are running concurrently, in practice it may not be so. On a computer with single CPU, threads actually run one at a time giving an illusion of concurrency.**
  - **The execution of multiple threads on a single CPU based on some algorithm is called thread scheduling.**
  - **Thread scheduler maintains a pool of all the ready-to-run threads. Based on fixed priority algorithm, it allocates free CPU to one of these threads.**

Creating threads

- Threads are objects in the Java language. A thread can be defined:
  - by extending the java.lang.Thread class or
  - by implementing the java.lang.Runnable interface.
- The run() method should be overridden and should have the code that will be executed by the new thread.

Thread example

```
class MyThread extends Thread{
  public void run(){
    System.out.println(“Thread: Inside run()");
  }
}
```

```
class MyRunnable implements Runnable{
  public void run(){
    System.out.println(“Runnable:Inside run()");
  }
}
```

```
…
public static void main(String[] args){
  MyThread mt = new MyThread();

  MyRunnable mc = new MyRunnable();
  Thread t = new Thread(mc);

  t.start(); mt.start();
}
```

Note:
different constructors

3 ready-to-run threads
(including main())

Thread states (1/2)

- **New .** After the thread is instantiated, the thread is in the New state until the start() method is invoked. In this state, the thread is not considered alive.

- **Ready- to- Run (Runnable) .** A thread comes into the runnable state when the start() method is invoked on it. It can also enter the runnable state from the running state or blocked state. The thread is considered alive when it is in this state.
- **Running.** A thread moves from the runnable state into the running state when the thread scheduler chooses it to be the currently running thread.

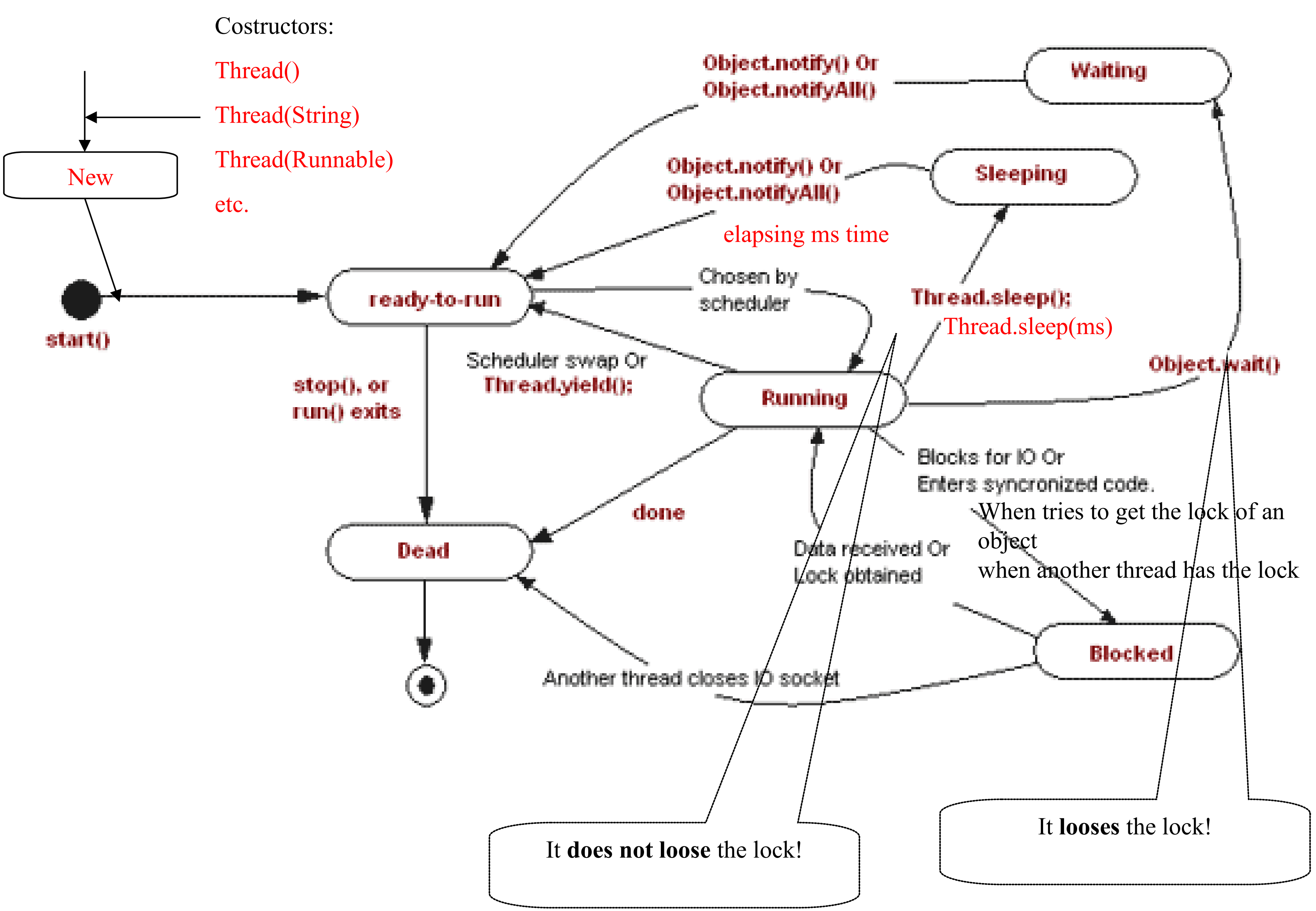
Alive, but not runnable. A thread can be alive but not in a runnable state for a variety of reasons. It may be waiting, sleeping, or blocked.

- **Waiting.** A thread is put into a waiting state by calling the wait() method. A call to notify() or notifyAll() may bring the thread from the waiting state into the runnable state.
- **Sleeping.** The sleep() method puts the thread into a sleeping state for a specified amount of time in milliseconds,
- **Blocked.** A thread may enter a blocked state while waiting for a resource like I/O or the lock of another object. In this case, the thread moves into the runnable state when the resource becomes available.

- **Dead.** A thread is considered dead when its run() method is completely executed. A dead thread can never enter any other state, not even if the start() method is invoked on it.

Thread states (2/2)

http://www.bpurcell.org



Costructors:

Thread()

Thread(String)

Thread(Runnable)

etc.

New

start()

ready-to-run

Object.notify() Or
Object.notifyAll()

Waiting

Object.notify() Or
Object.notifyAll()

Sleeping

elapsing ms time

Chosen by
scheduler

Thread.sleep();

Thread.sleep(ms)

Object.wait()

Scheduler swap Or
Thread.yield();

stop(), or
run() exits

Running

Blocks for IO Or
Enters syncronized code.

When tries to get the lock of an object
when another thread has the lock

done

Dead

Data received Or
Lock obtained

Blocked

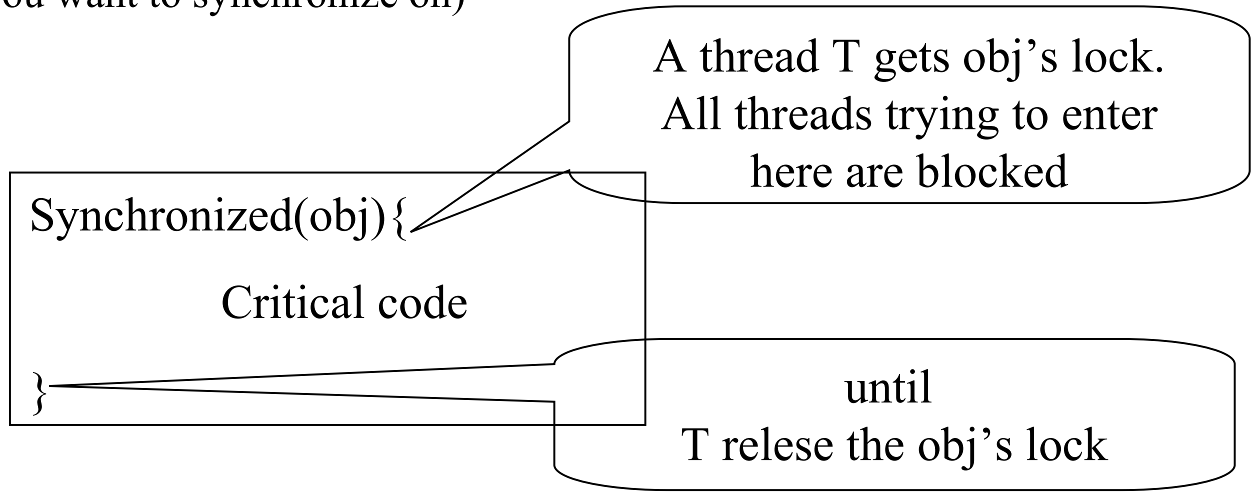
Another thread closes IO socket

It **does not loose** the lock!

It **looses** the lock!

Thread synchronization

- Every object in Java code has one lock, which is useful for ensuring that only one thread accesses critical code in the object at a time.
- If a thread has obtained the lock, no other thread can enter the synchronized code until the lock is released.
- When the thread holding the lock exits the synchronized code, the lock is released.
- If a thread tries to get the lock of an object when another thread has the lock, the thread goes into a **blocked** state until the lock is released.
- **synchronized** keyword:
  - declare a method as synchronized (synchronize on lock of the destination object)
  - mark a block of code as synchronized (the argument passed should be the object whose lock you want to synchronize on)



```
Synchronized(obj){
            Critical code
}
```

A thread T gets obj's lock.
All threads trying to enter
here are blocked

until
T relese the obj's lock

Monitors (1/?)

- Problems may occur when two threads are trying to access/modify the same object. To prevent such problems, Java uses monitors and the synchronized keyword to control access to an object by a thread.
- **Monitor**
  - Monitor is any class with synchronized code in it.
  - Monitor controls its client threads using, wait() and notify() (or notifyAll()) methods.
  - wait() and notify() methods must be called in synchronized code.
  - Monitor asks client threads to wait if it is unavailable.
  - Normally a call to wait() is placed in while loop. The condition of while loop generally tests the availability of monitor. After waiting, thread resumes execution from the point it left.

if the thread is holding a lock and went to a sleeping state, it does not loose the lock.

Monitors (2/?) http://www.artima.com/insidejvm/ed2/threadsynch.html

- Java's monitor supports two kinds of thread synchronization:
  - *mutual exclusion*
    supported in the Java virtual machine via object locks, enables multiple threads to independently work on shared data without interfering with each other.
  - *Cooperation*
    supported in the Java virtual machine via the wait and notify methods of class Object, enables threads to work together towards a common goal.
- The form of monitor used by the Java virtual machine is called a

"Wait and Notify" monitor (It is also sometimes called a "Signal and Continue")

A graphical depiction of JVM monitor

http://www.artima.com/insidejvm/ed2/threadsynch2.html

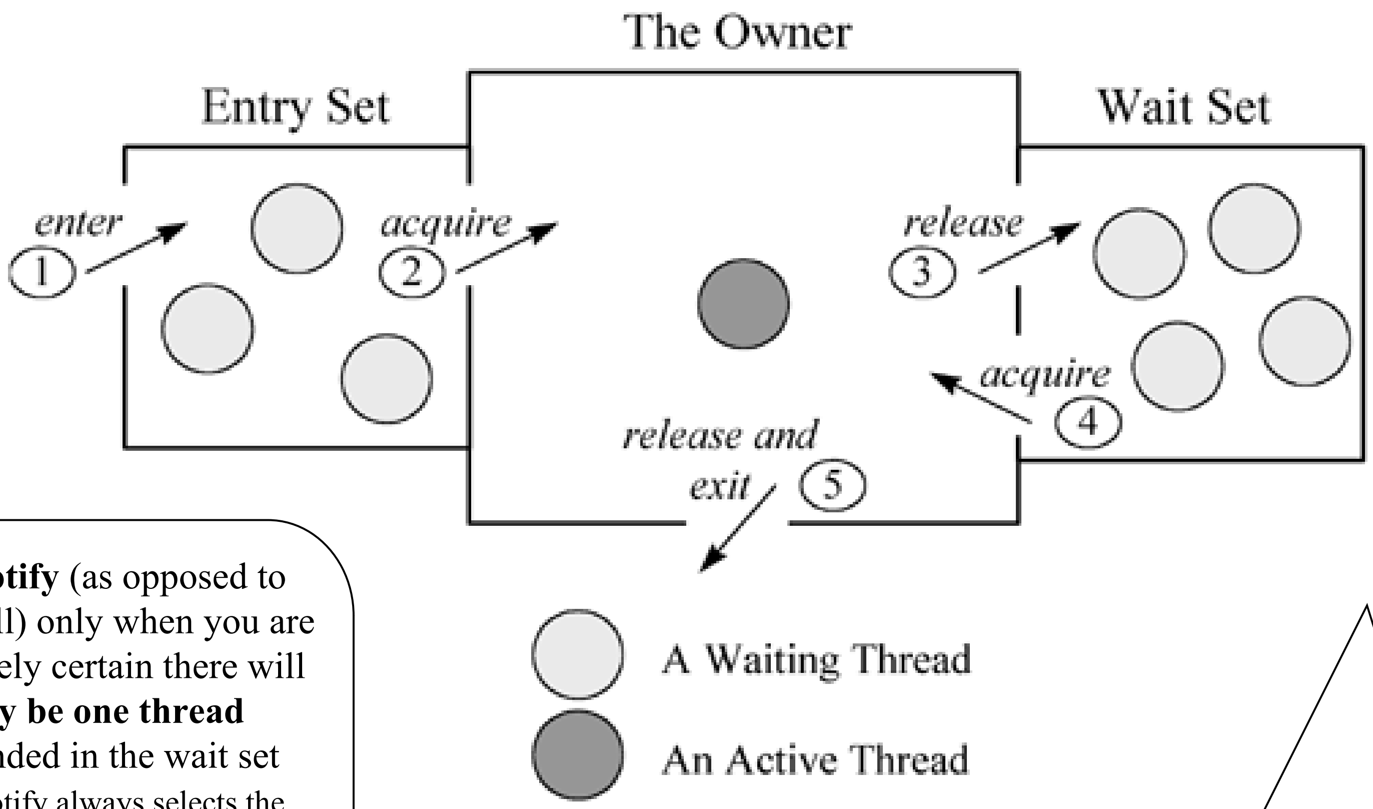


Figure 20-1. A Java monitor.

use **notify** (as opposed to notify all) only when you are absolutely certain there will **only be one thread** suspended in the wait set

(If a notify always selects the most recent arrival from the wait set and the wait set always contains multiple threads, some threads that have been waiting the longest may never be resurrected.)

the manner in which a Java virtual machine implementation selects the next thread from the wait or entry sets is a decision of

individual implementation designers.

The wait and notify methods of class Object

Method	Description
`void wait();`	Enter a monitor's wait set until notified by another thread
`void wait(long timeout);`	Enter a monitor's wait set until notified by another thread or `timeout` milliseconds elapses
`void wait(long timeout, int nanos);`	Enter a monitor's wait set until notified by another thread or `timeout` milliseconds plus `nanos` nanoseconds elapses
`void notify();`	Wake up one thread waiting in the monitor's wait set. (If no threads are waiting, do nothing.)
`void notifyAll();`	Wake up all threads waiting in the monitor's wait set. (If no threads are waiting, do nothing.)

Example. A buffer protected by a monitor

Scenario involves:

- a buffer (protected by a monitor)
- read threads, and
- write threads.
- When a read thread enters the monitor, it checks to see if the buffer is empty.
  - not empty, reads (and removes) some data from the buffer. and exits the monitor.
  - empty, the read thread executes a **wait** command.
    - the read thread is suspended and placed into the monitor's wait set, releases the monitor, which becomes available to other threads.
- When the write thread enters the monitor:
  - writes some data into the buffer,
  - executes a notify,
  - exits the monitor.
- When the write thread executes the notify, the read thread is marked for eventual resurrection. After the write thread has exited the monitor, the read thread is resurrected as the owner of the monitor.
- If there is any chance that some other thread has come along and consumed the data left by the write thread, the read thread must explicitly check to make sure the buffer is not empty.
- If there is no chance that any other thread has consumed the data, then the read thread can just assume the data exists. The read thread reads some data from the buffer and exits the monitor.

Exemplu

Transmiterea mesajelor

- cu păstrarea integrităţii;
- fara pierderea mesajelor
- un mesaj e preluat de un singur destinatar
- un destinatar nu preia acelaşi mesaj de mai multe ori

Clasa monitor: Buffer

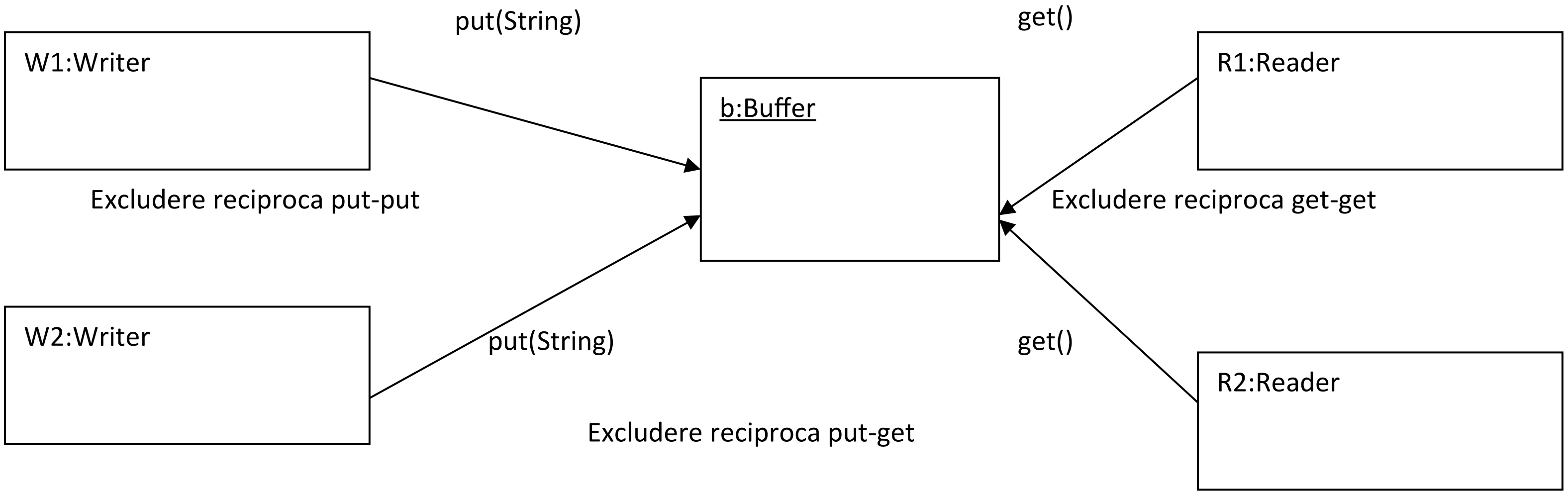
Clasa producator: Writer

Clasa consumator: Reader

See C:\TUDOR\JAVA\Concurrent\waitnotify\whilewaitnotify

- **Observaţia 1**. Prin clauza synchronized metodele devin zone de exculdere reciprocă.

Numai unul din firele ce au transmis mesaje synchronized la acelaşi obiect monitor ocupă monitorul pe tot parcursul executării metodei (se spune că firul este proprietarul monitorului). Când metoda synchronized se termină, firul pierde controlul asupra obiectului monitor. Firul mai poate pierde controlul monitorului şi în cazul când i se transmite mesajul wait().



W1:Writer

W2:Writer

put(String)

put(String)

b:Buffer

get()

get()

R1:Reader

R2:Reader

Excludere reciproca put-put

Excludere reciproca get-get

Excludere reciproca put-get

Observaţia 2. Dacă i se transmite mesajul wait(), monitorul trece firul de executare care are control asupra sa în starea de aşteptare wait. Fiecare obiect monitor are ataşată o mulţime proprie de fire în aşteptare. Dacă i se transmite mesajul notify() şi mulţimea firelor în aşteptare nu este vidă atunci un   fir oarecare din această mulţime preia controlul asupra obiectului monitor. Prin urmare, este executată metoda care a transmis mesajul wait(), începând cu instrucţiunea care urmează invocării acestei metode.

```
C:\TUDOR\JAVA\Concurrent\waitnotify\whilewaitnotify>java WriterReader
Message Thread[Thread-0,5,main].1 from thread Thread[Thread-0,5,main]consumed by readerThread[Thread-9,5,main]
Message Thread[Thread-0,5,main].2 from thread Thread[Thread-0,5,main]consumed by readerThread[Thread-9,5,main]
Message Thread[Thread-2,5,main].1 from thread Thread[Thread-2,5,main]consumed by readerThread[Thread-1,5,main]
Message Thread[Thread-0,5,main].3 from thread Thread[Thread-0,5,main]consumed by readerThread[Thread-9,5,main]
Message Thread[Thread-4,5,main].1 from thread Thread[Thread-4,5,main]consumed by readerThread[Thread-1,5,main]
Message Thread[Thread-6,5,main].1 from thread Thread[Thread-6,5,main]consumed by readerThread[Thread-9,5,main]
Message Thread[Thread-2,5,main].2 from thread Thread[Thread-2,5,main]consumed by readerThread[Thread-1,5,main]
Message Thread[Thread-0,5,main].4 from thread Thread[Thread-0,5,main]consumed by readerThread[Thread-7,5,main]
```

- Ordinea de afişare depinde de contextul de executare, Clauza synchronized asigură transmiterea si afişarea liniilor fără pierderi de caractere din linie.
- Instrucţiunile de sincronizare care utilizează mesajele wait(), notify() şi variabila de control contor au următorul efect:
- împiedică afişarea repetată a unui mesaj
  (prin instrucţiunea while (count==0) wait();)
- nici-un mesaj transmis nu este pierdut
  (prin instrucţiunea while (count==maxSize) wait();)

- **Observaţia 4.** Aparent, acelaşi efect de sincronizare se poate obţine prin înlocuind instrucţiunile de sincronizare

```
while (count==1) wait();
```

şi

```
while (count==0) wait();
```

prin

```
if (count==1) wait();
```

respectiv

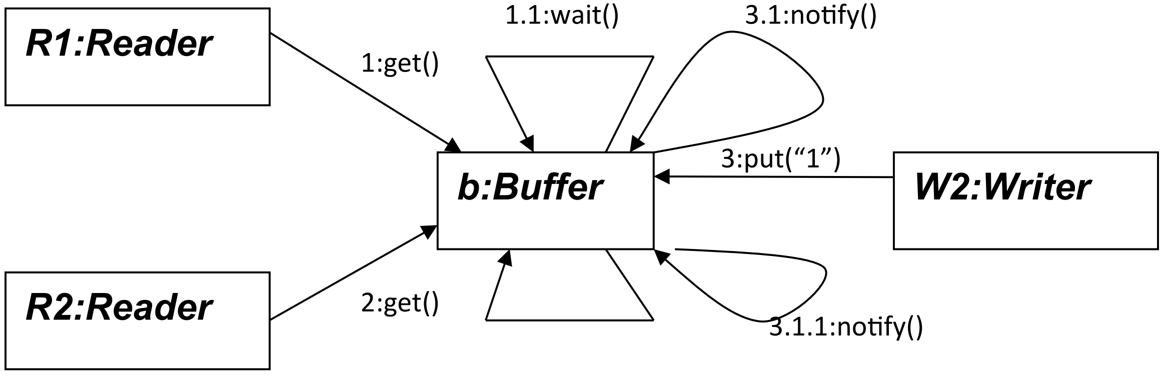
```
if (count==0) wait();
```

- În realitate, mecanismul de sincronizare astfel modificat, deşi asigură încă afişarea integrală a caracterelor dintr-o linie, nu mai este capabil să evite afişarea multiplă a unei linii
- Un posibil rezultat ar putea fi de exemplu:
- Message 2.1 from thread 2
- Message 2.1 from thread 2
- etc.
-
- O colaborare ipotetică intre obiecte, care să justifice acest rezultat, este următoarea:

- Un posibil rezultat ar putea fi de exemplu:

```
Message Thread[Thread-0,5,main].1 from thread Thread[Thread-0,5,main]consumed by readerThread[Thread-9,5,main]
nullconsumed by readerThread[Thread-7,5,main]
nullconsumed by readerThread[Thread-5,5,main]
nullconsumed by readerThread[Thread-3,5,main]
nullconsumed by readerThread[Thread-1,5,main]
nullconsumed by readerThread[Thread-1,5,main]
nullconsumed by readerThread[Thread-3,5,main]
…..]
nullconsumed by readerThread[Thread-1,5,main]
nullconsumed by readerThread[Thread-9,5,main]
nullconsumed by readerThread[Thread-1,5,main]
nullconsumed by readerThread[Thread-5,5,main]
nullconsumed by readerThread[Thread-3,5,main]
nullconsumed by readerThread[Thread-1,5,main]
nullconsumed by readerThread[Thread-1,5,main]
nullconsumed by readerThread[Thread-7,5,main]
Message Thread[Thread-0,5,main].1 from thread Thread[Thread-0,5,main]consumed by readerThread[Thread-3,5,main]
nullconsumed by readerThread[Thread-1,5,main]
nullconsumed by readerThread[Thread-5,5,main]
```

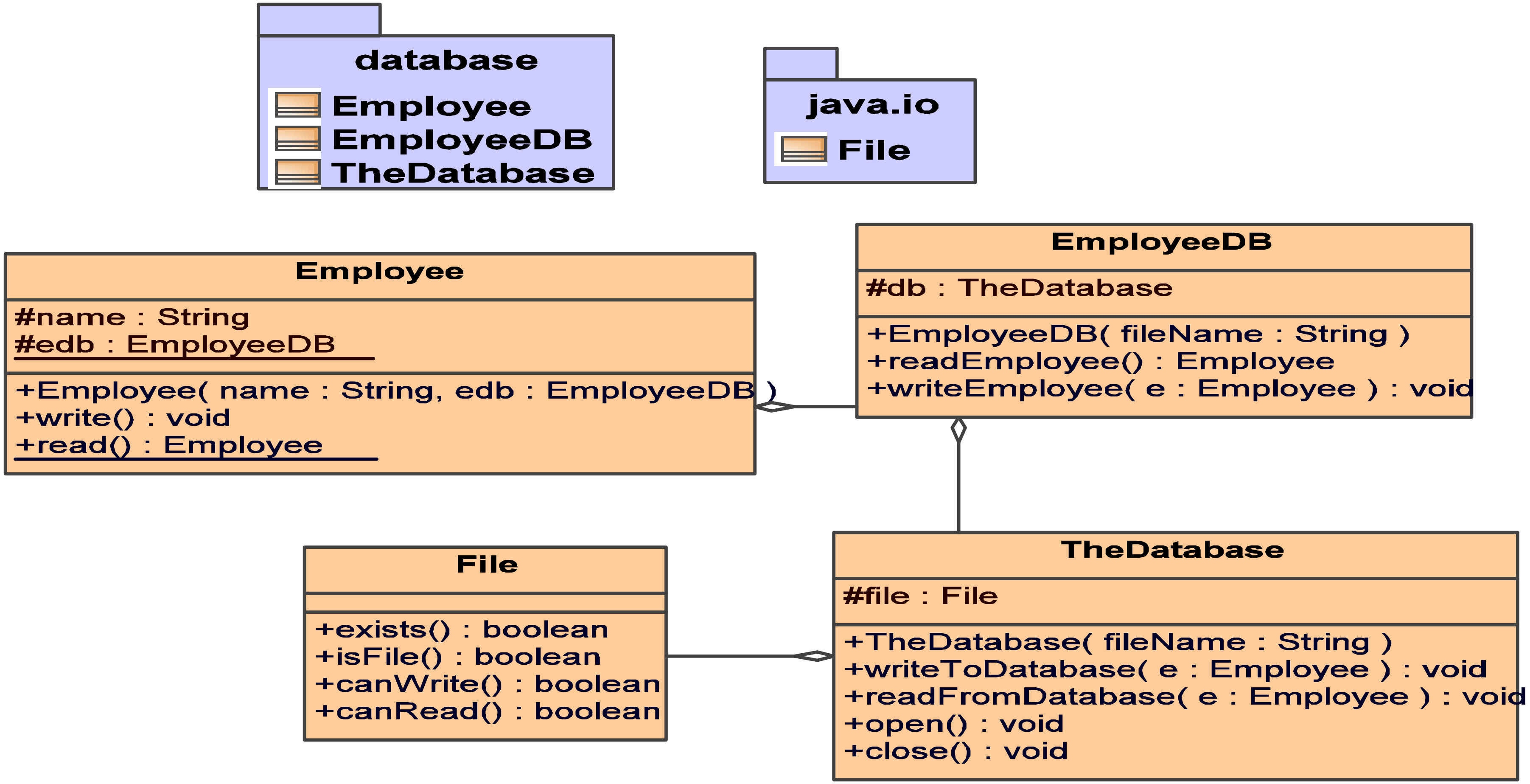
O colaborare ipotetică intre obiecte, care să justifice acest rezultat, este următoarea:



Open Closed Principle

Principii ale Proiectării Orientate pe Obiecte (OO Design) (1/3)

- **Open Closed Principle (OCP):** *it should be possible to change the environment of a class without changing the class (open for extension, closed for modification)*
- Exemplu: sistemul următor **nu** respectă OCP



database
Employee
EmployeeDB
TheDatabase

java.io
File

Employee
#name : String
#edb : EmployeeDB
+Employee(name : String, edb : EmployeeDB)
+write() : void
+read() : Employee

EmployeeDB
#db : TheDatabase
+EmployeeDB(fileName : String)
+readEmployee() : Employee
+writeEmployee(e : Employee) : void

File
+exists() : boolean
+isFile() : boolean
+canWrite() : boolean
+canRead() : boolean

TheDatabase
#file : File
+TheDatabase(fileName : String)
+writeToDatabase(e : Employee) : void
+readFromDatabase(e : Employee) : void
+open() : void
+close() : void

Implementare Java

- Clasa `Employee:` Java\OOD\OpenClosed\Principiul OpenClosed incalcat\database\Employee.java

```
// fisierul este in directorul database
// se compileaza din directorul parinte al lui database, prin
// javac database\Employee.java
package database;
public class Employee{
    public Employee(String name, EmployeeDB edb){this.name=name; this.edb=edb;}
    public Employee(String name){this.name=name;}
    public static Employee read(){
              return edb.readEmployee();
    }
    public void write(){
              edb.writeEmployee(this);
    }
    protected String name;
    protected static EmployeeDB edb=new EmployeeDB("c:\\Database");

}
```

Implementare Java

- Clasa `EmployeeDB`: Java\OOD\OpenClosed\Principiul OpenClosed incalcat\database\EmployeeDB.java

```
// fisierul este in directorul database
// se compileaza din directorul parinte al lui database, prin
// javac database\EmployeeDB.java
package database;
public class EmployeeDB{
   public EmployeeDB(String fileName){
            db=new TheDatabase(fileName);
   }
   public Employee readEmployee(){
            Employee e=new Employee("XXXX");
            db.open();
            db.readFromDatabase(e);
            db.close();
            return e;
   }
   public void writeEmployee(Employee e){
            db.open();
            db.writeToDatabase(e);
            db.close();
   }
   protected TheDatabase db;

}
```

Implementare Java

- Clasa TheDatabase: Java\OOD\OpenClosed\Principiul OpenClosed incalcat\database\TheDatabase.java

```
// fisierul este in directorul database
// se compileaza din directorul parinte al lui database, prin
// javac database\TheDatabase.java
package database;
import java.io.*;
public class TheDatabase{
    public TheDatabase(String fileName){file=new File(fileName);}
    public void writeToDatabase(Employee e){
    //metoda complexa, elemente SQL si JDBC
    //...
    if (file.exists() && file.isFile()){

    if (file.canWrite()){
    System.out.println("A real writeToDatabase");
    }else System.out.println("Error: can not write");
    }
    }
    public void readFromDatabase(Employee e){
    //metoda complexa, elemente SQL si JDBC
    //...
    if (file.exists() && file.isFile()){

    if (file.canRead()){
    // se citeste din fisier
    System.out.println("A real readFromDatabase");
    }else System.out.println("Error: can not read");
    }
    }
    // continua pe urmatorul slide
```

Continuare implementare clasa TheDatabase

```
// continuare
public void open(){
            //metoda complexa, elemente SQL si JDBC
            //...
            if (file.exists() && file.isFile()){
                     // se deschide fisierul
                     System.out.println("A real open");
            }else System.out.println("Error: can not open");
   }
   public void close(){
            //metoda complexa, elemente SQL si JDBC
            //...
            if (file.exists() && file.isFile()){
                     // inchide fisierul
                     System.out.println("A real close");
            }else System.out.println("Error: can not close");
   }
   protected File file;

}
```

OCP încălcat(2/3)

- Într-adevăr, fie următoarea aplicaţie, în care sistemul de clase este utilizat într-un context (environment) presupus a fi cel real: Java\OOD\OpenClosed\Principiul OpenClosed incalcat\Aplicatie.java

```
//import database.*;
public class Aplicatie{
    public static void main(String[] args){
              database.Employee e;
              database.EmployeeDB edb=new database.EmployeeDB("K:\\java\\OpenClosed\\
    Database.db");
              e=new database.Employee("Tudor", edb);
              e.write();
              e=new database.Employee("Andrei", edb);
              e.write();
              database.Employee x,y;
              x=database.Employee.read();
              y=database.Employee.read();


    }
}
```

OCP încălcat (3/5)

- **Cazul în care principiul Open Close nu este respectat.**
- În acest exemplu, clasa TheDatabase conţine elemente complexe care includ printre altele accesul la fişiere şi la baze de date. Să presupunem că metodele sale sunt în curs de elborare iar fişierul cu care lucrează (database.db) nu a fost creat la acest moment.
- Cum ar trebui să procedăm dacă vrem să testăm, în acest stadiu, în care baza de date nu există, măcar aspectele esenţiale ale procedurilor de citire sau scriere a datelor despre un obiect Employee? Să spunem de pildă că ne-ar interesa să urmărim dacă procedura de scriere urmează paşii următori:
  - deschidere fişier (open)
  - scriere efectivă (writeToDatabase)
  - închidere fişier(close)
(a se vedea EmployeeDB.writeToDatabase).
- **Soluţie:** clasa TheDatabase ar trebui eliminată din pachetul `database` şi înlocuită cu o altă clasa, **cu acelaşi nume,** dar care să joace rolul unei schiţe (stub) a clasei reale.(a se vedea diapozitivul următor)
- **Obiecţie.** Modificarea contextului în care este folosită clasa Employee (înlocuirea contextului real cu unul de test) atrage modificarea sistemului de module (clase). Coexistenţa unor module cu acelaşi nume, dar cu funcţionalităţi distincte face ca gestionarea proiectului să fie greoaie. Există riscul ca unele module de test să rămână încorporate şi să substituie în aplicaţia reală modulele finale.

Schiţă de implementare (stub) TheDatabase, numai pentru test

- Java\OOD\OpenClosed\Principiul OpenClosed incalcat\database\TheDatabase for test\TheDatabase.java

```
// fisierul este in directorul database
// se compileaza din directorul parinte al lui database, prin
// javac database\TheDatabse.java
package database;
import java.io.*;
public class TheDatabase{ // a stub class, just for testing
   public TheDatabase(String fileName){file=new File(fileName);}
   public void writeToDatabase(Employee e){
   System.out.println("A simulated writeToDatabase");
   }
   public void readFromDatabase(Employee e){
   System.out.println("A simulated readFromDatabase");
   }
   public void open(){
   System.out.println("A simulated open");
   }
   public void close(){
   System.out.println("A simulated close");
   }
   protected File file;
}
```

C++/Java: destructor versus finalize

- Un destructor C++ realizează acţiunile:

1. Eliberarea resurselor alocate explicit(memorie alocată dinamic, fişiere etc.)
2. Eliberarea memoriei alocate implicit pentru atribute

Este responsabilitatea programatorului să asigure eliberarea resurselor alocate explicit (operatorul delete).

- În Java, eliberarea resurselor se face automat, prin acţiunea unui proces (activat când există o criză de memorie sau la terminarea programului). Pentru a elibera anumite resurse înainte (fişiere etc.) înainte de intervenţia procesului *garbage collector* poate fi utilizată metoda finalize().

Java\finalize\TestFinalize.java

```
public class TestFinalize{

        public static void main(String[] args){
        C f=new C(); // un obiect este creat si este referit de f
        f=null; // obiectul creat nu mai este referit
        C g=new C();
        C h=new C();

        System.out.println("Un obiect nereferit, dar necolectat inca");
        h.finalize();
        System.out.println("Au fost eliberate explicit (finalize()) resursele obiectului creat ultima data");
        System.out.println("Desi cel creat naintea lui nu mai este referit, resursele nu au fost inca eliberate");
        System.gc();
        System.runFinalization();

        }
}

class C{
        static int nr=0;
        private int id=0;
        C(){            id=++nr;
        System.out.println("A fost creat obiectul " + id);
        }
        protected void finalize(){
        System.out.println("Eliberare resurse obiectul "+id);
        }

}
```

```
D:\Tudor\POO, curs 2005-2006\Java\finalize>java TestFinalize
A fost creat obiectul 1
A fost creat obiectul 2
A fost creat obiectul 3
Un obiect nereferit, dar necolectat inca
Eliberare resurse obiectul 3
Au fost eliberate explicit (finalize()) resursele obiectului creat ultima data
Desi cel creat inaintea lui nu mai este referit, resursele nu au fost inca eliberate
Eliberare resurse obiectul 1
```