Curs 8 POO
© Conf. univ dr. Crenguta M. Puchianu

- Polimorfism static
- Polimorfism dinamic
- Clase abstracte
- Clase record

Polimorfism in Java

- **Polimorfismul** este o caracteristică a unei entităţi de a avea semnificaţii sau utilizări diferite în contexte diferite.

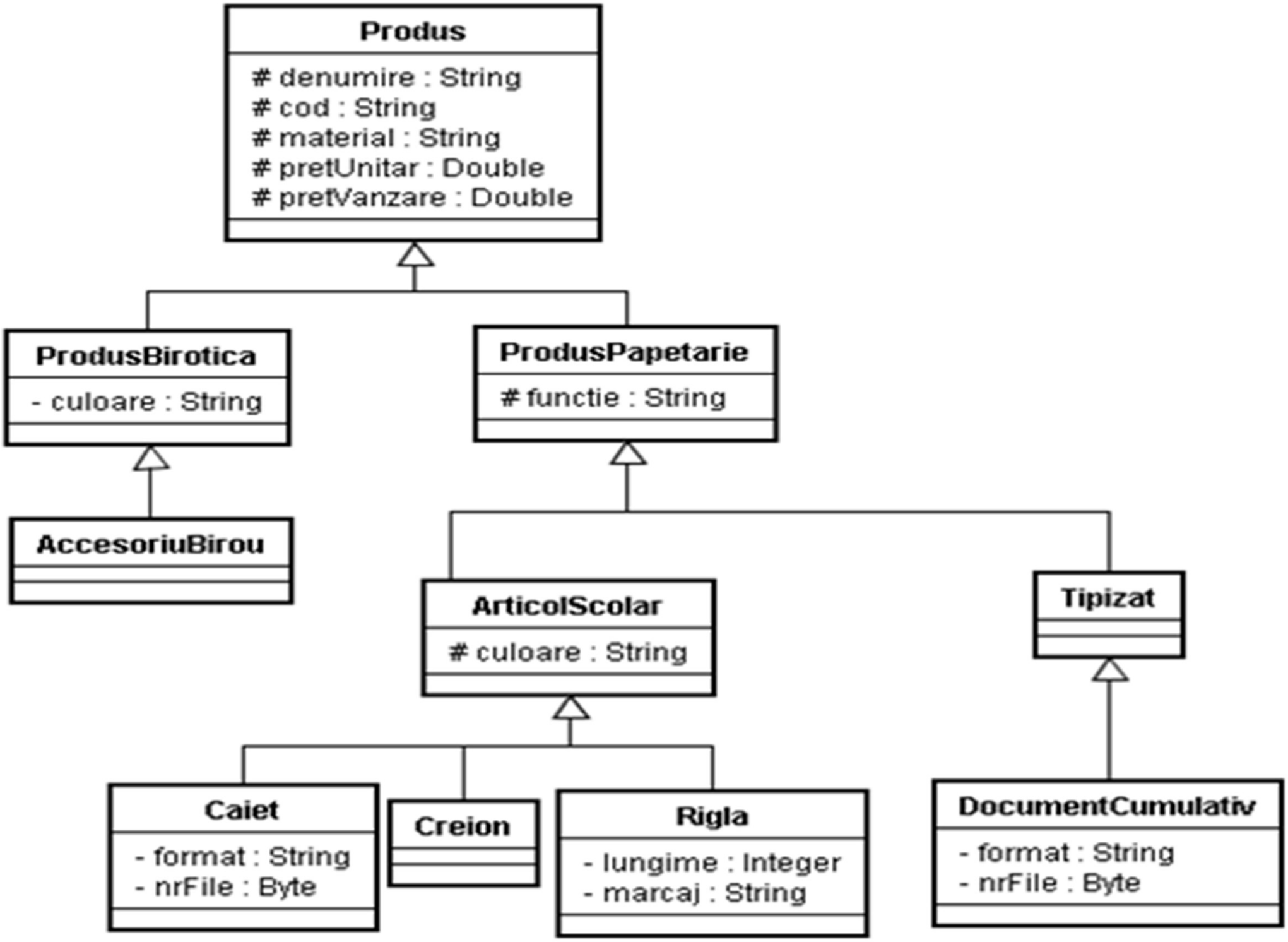
- **Polimorfism static**:
  - Operatori: +, /
  - Metode supraincarcate definite in aceeasi clasa sau mostenite de la supraclasa directa sau supraclasele indirecte ale clasei

- **Regulă**. În cazul polimorfismul static, compilatorul stie sa diferentieze metodele supraincarcate, dupa semnatura lor fara sa execute programul.

Polimorfism dinamic

- Dacă o metodă este redefinită în subclasă, trebuie să îndeplinească următoarele condiţii:
  - metoda redefinită trebuie să aibă exact aceeaşi semnătură;
  - metoda redefinită trebuie să aibă acelaşi tip al rezultatului întors;
  - dacă metoda redefinită îşi schimbă modificatorul de acces, acesta trebuie să dea cel puţin la fel de mult acces ca metoda moştenită din supraclasă.

- **Regulă**. În polimorfismul dinamic, algoritmul metodei va fi decis în momentul executării apelului după tipul obiectului şi nu după tipul variabilei ce memorează referinţa la obiectul respectiv.


Produs
denumire : String
cod : String
material : String
pretUnitar : Double
pretVanzare : Double
ProdusBirotica
- culoare : String
ProdusPapetarie
functie : String
AccesoriuBirou
ArticolScolar
culoare : String
Tipizat
Caiet
- format : String
- nrFile : Byte
Creion
Rigla
- lungime : Integer
- marcaj : String
DocumentCumulativ
- format : String
- nrFile : Byte

Clase abstracte

- O **clasă abstractă** este o clasă fără instanţe directe, dar are subclase care sunt clase concrete, adică pot fi instanţiate direct.

- Sintaxa declarării unei clase abstracte:

ClasăAbstractă::= [ModificatorVizibilitate] **abstract class** NumeClasa**{**…**}**

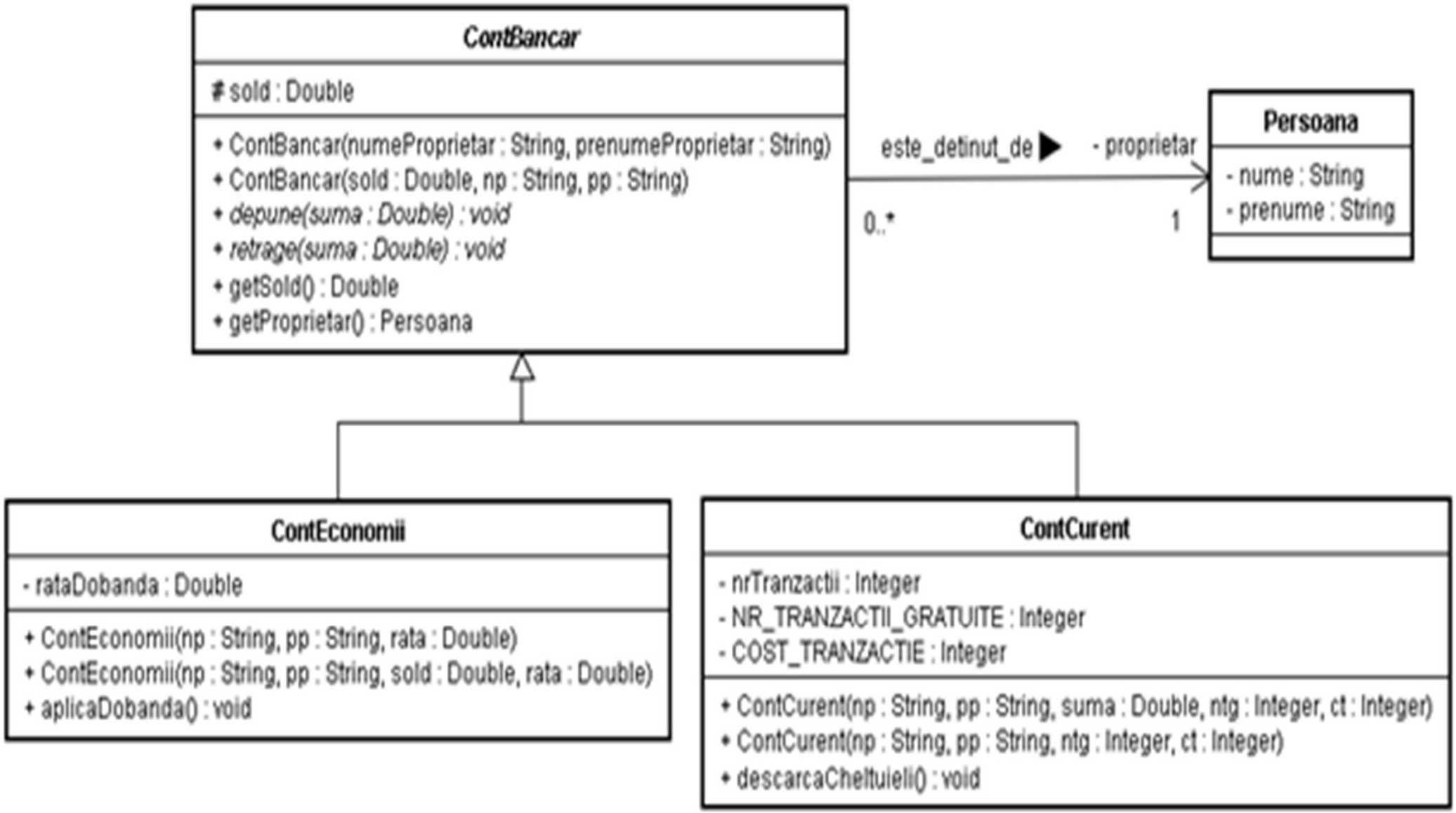
- O clasă abstractă conţine cel puţin o *metodă abstractă,* adică o metodă care nu defineşte algoritmul de implementare.

- Sintaxa declarării unei metode abstracte este următoarea:

MetodaAbstractă::=[ModificatorVizibilitate] **abstract TipDate** numeMetoda**(**
[ListaParametriFormali]**);**

- **Regula**. O subclasă a unei clase abstracte are obligaţia de a defini (implementa) metodele abstracte ale supraclasei.
- În caz contrar, subclasa respectivă trebuie să fie declarată abstractă şi este responsabilitatea subclaselor ei să implementeze metodele care au rămas abstracte din supraclasă.

Clase abstracte



Clase record

Au aparut in JDK 16.

Sintaxa:

[modificator vizibilitate] **record** identificator**(**[ListaParametriStare]**){**

//eventual constructori supraincarcati cu cel canonic

//eventual metode de clasa si/sau instanta

//eventual variabile statice

//eventual initializatori statici

}

ListaParametriStare ::= λ | Parametru | Parametru, ListaParametriStare

ParametruStare ::= TipDate identificator

Important:

1.Compilatorul genereaza automat:

- Un constructor canonic a carui semnatura mapeaza tipul si ordinea parametrilor de stare
- Cate o metoda de accesare a valorii fiecarui parametru de stare
- Metodele equals(), toString() şi hashCode().

2. Parametri de stare (variabile instanta) sunt automati finali

3. Clasa este finala

4. Este thread-safe

Clase record (cont.)

Exemplu:

```
public record SpecificatieProdus (String denumire, double
pretUnitar) {}
```

Important:

1. Putem avea constructori supraincarcati. De exemplu:

```
public SpecificatieProdus(String denumire){
    this(denumire, 5);
}
```

2. Un record poate contine metode de clasa si/sau metode instanta

3. Un record poate avea variabile statice

4. Un record extinde automat clasa abstracta Record din pachetul java.lang.

5. Un record poate implementa una sau mai multe interfete

Record SpecificatieProdus

```
public record SpecificatieProdus (String denumire, double
pretUnitar) {
    public static int TVA=19;

 public SpecificatieProdus(String denumire){
     this(denumire, 5);
 }
 public void afiseazaPretFormatat(){
     System.out.format("%.2f\n", pretUnitar);
 }
 public static void afiseazaDenumireLitereMari
                                    (SpecificatieProdus p)
{ System.out.format("Denumirea produsului transformata este
%s\n", p.denumire.toUpperCase());
 }
 public double aplicaTVA(){
     return pretUnitar*TVA/100;
 }
```

Clase record cu obiecte modificabile

Prin definitie, valorile variabilele instanta ale obiectelor claselor record nu pot fi modificate. Totusi, putem modifica starea obiectelor, prin transmiterea variabilelor colectie (statica sau dinamica).

```
public record SpecificatieProdus2 (String denumire, double pretUnitar,
double[] istoricpreturi) {
    public static int TVA=19;
    public SpecificatieProdus2(String denumire, double[] istoricpreturi){
     this(denumire, 5, istoricpreturi);
 }
public static void main(String... args){
     double[] d=new double[3];
     d[0]=1.23;
     d[1]=0.23;
     SpecificatieProdus2 sp=new SpecificatieProdus2("creion", d);
     d[2]=0.34;
     double[] d1=sp.istoricpreturi();
     for(double e:d1) System.out.print(e+" ");
     System.out.println();
 }}
```

Clase record cu obiecte immutable

Pentru a rezolva problema:

```
public record SpecificatieProdus2 (String denumire, double
pretUnitar, double[] istoricpreturi) {
    public static int TVA=19;

 public SpecificatieProdus2(String denumire, double[] istoric){
     this(denumire, 5, new double[istoric.length]);
     System.arraycopy(istoric, 0, istoricpreturi, 0,
                                  istoricpreturi.length);

 }
```