PROGRAMARE ORIENTATĂ PE OBIECTE

GENGE BÉLA

Capitolul 4
Pachete. Interfețe. Diagrame UML

Problema

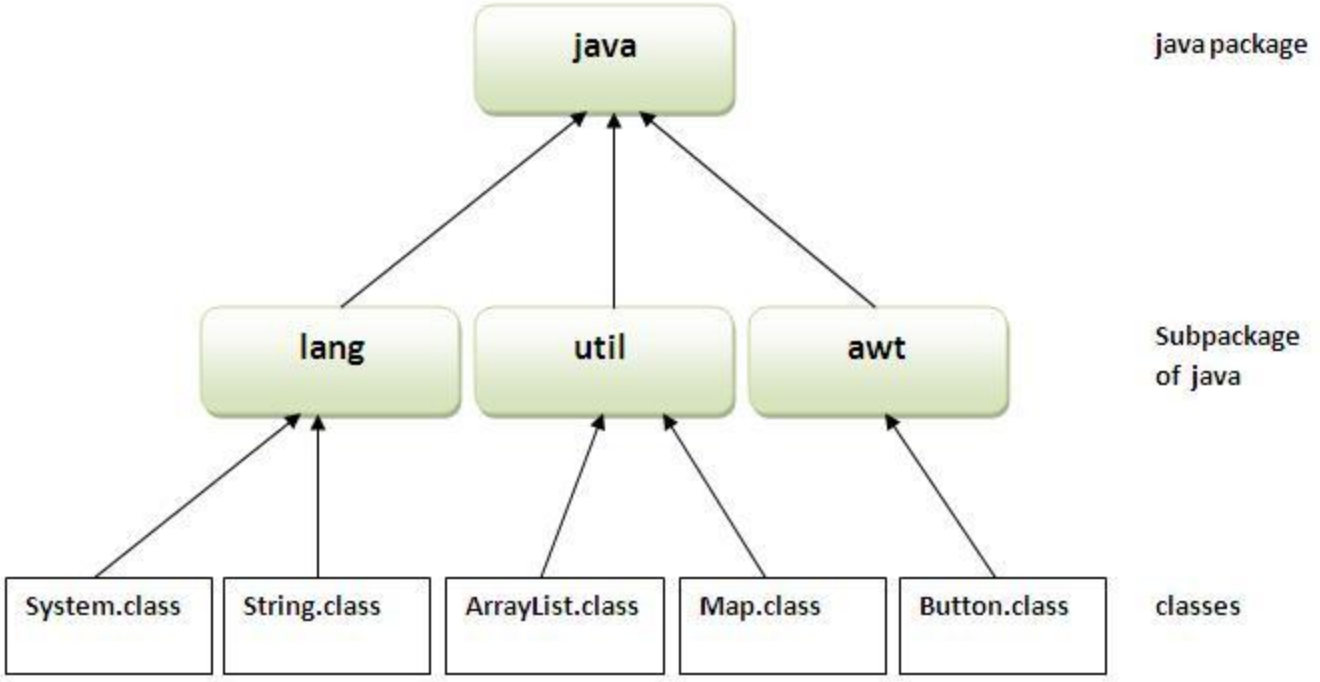
- Coliziuni de nume.
- Dacă implementăm două funcții/clase cu același nume, dar funcționalități diferite nu le putem diferenția.
- Exemplu *namespace* pentru C++.

Soluția Java

- Java fiind un limbaj modern include suport pentru *pachete*.
- Pachetele identifică un spațiu de nume de sine stătător.
- Pachetele pot conține alte pachete.
- Putem avea pachete cu același nume, însă nu și la același nivel.

Soluția Java

- Exemplu ierarhie de pachete în Java.



Sursa: http://www.javatpoint.com/package

Crearea pachetelor

- Cuvântul cheie *package*.

```
3  package automobil;
4
5  public class Automobil {
6      public Automobil() {
7          ...
8      }
9
10     ...
11 }
```

- Pachetul implicit fără nume.

Crearea pachetelor

- Crearea unei ierarhii de pachete:
  - Nivelul 0: myapplication.
  - Nivelul 1: logare, gui.
  - Nivelul 2: gui.ferestre, gui.butoane.

Utilizarea pachetelor

- Importarea conținutului pachetelor.
- Cuvântul cheie *import.*
- Caracterul *.
- Exemplu Scanner.

- Importarea statică a membrilor publici statici:
  - import static java.lang.Math.PI; - câmp
  - import static java.lang.Math.random; - metodă
  - Folosire: System.out.println(“PI: ” + PI + “ rand: ” + random());
- Vizibilitatea claselor.

Interfețe Java

- Asigură descrierea **abstractă** a metodelor.
- Interfețele listează:
  - Setul de metode **publice** ce vor fi implementate în cadrul claselor.
  - Setul de câmpuri **constante (final)** și **statice**.
- Cuvântul cheie *interface.*
- Exemplu automobil.

Interfețe Java

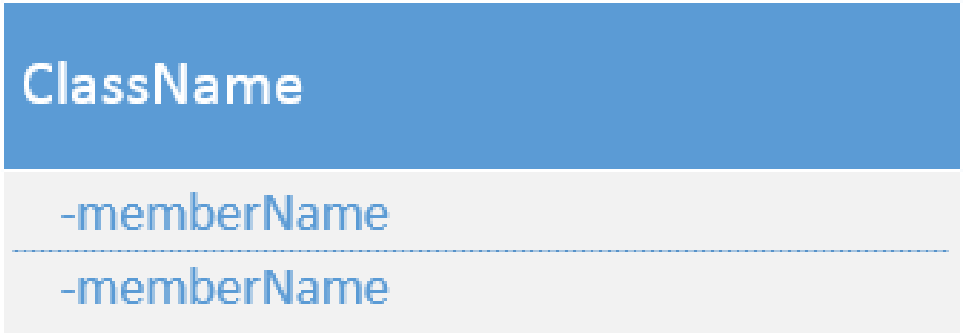
- Implementarea: *implements.*
- Aplicabilitate:
  - Operare unitară asupra obiectelor.
- Exemplu: vector cu 4 tipuri de automobile.

Diagrame UML

- Clasele și interfețele sunt reprezentate prin diagrame UML: Unified Modeling Language.
- + public
- - private
- # protected
- ~ package

Diagrame UML

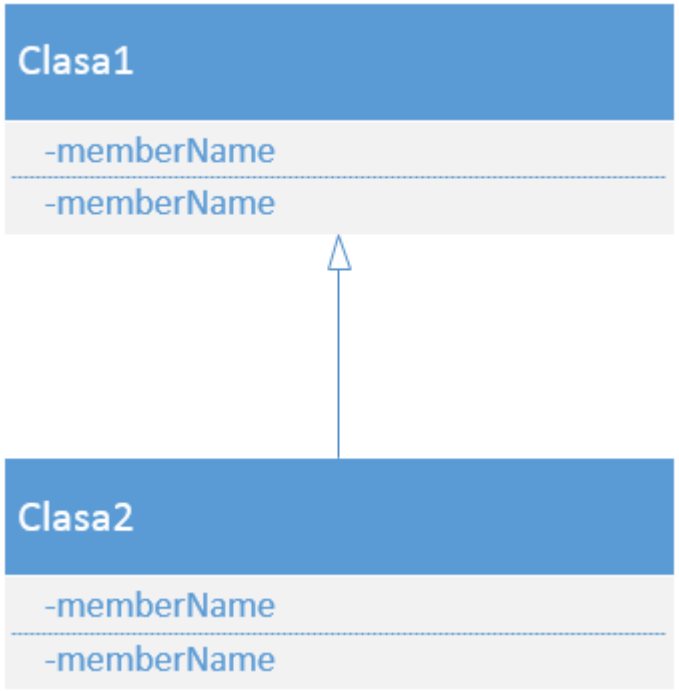
- Reprezentarea unei clase:



- Reprezentarea unei interfețe:

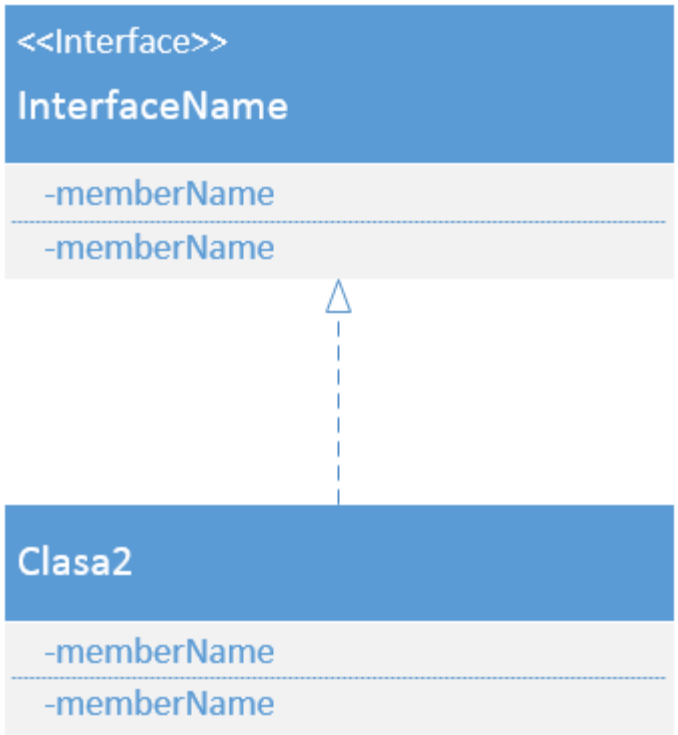
Diagrame UML

- Tipuri de relații. Moștenirea:



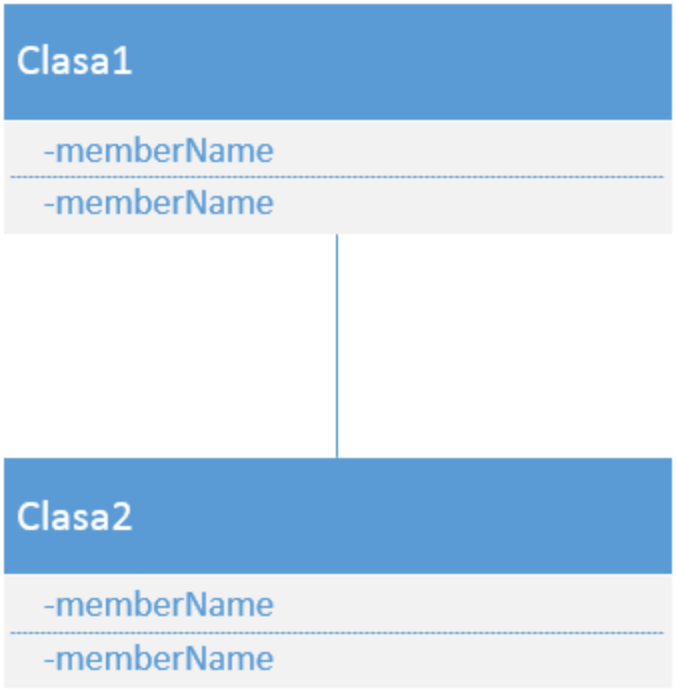
Diagrame UML

- Tipuri de relații. Implementarea unei interfețe:



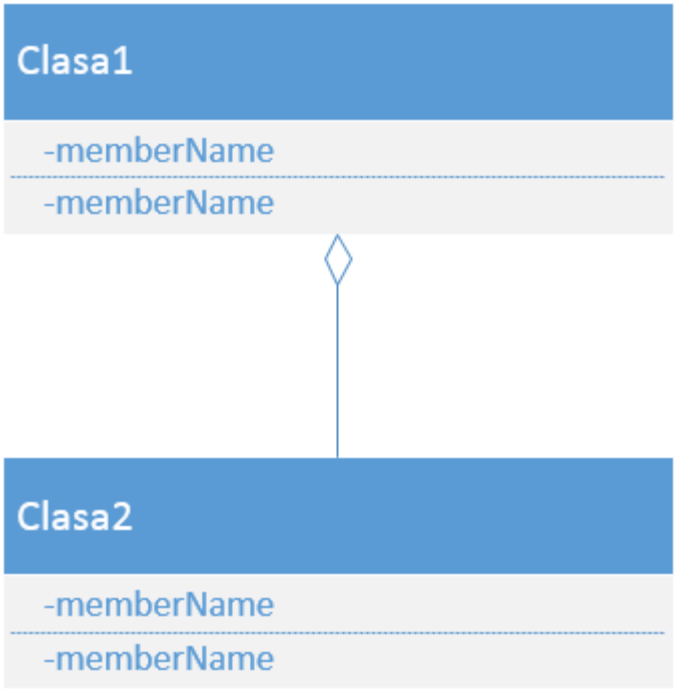
Diagrame UML

- Tipuri de relații. Asocierea:



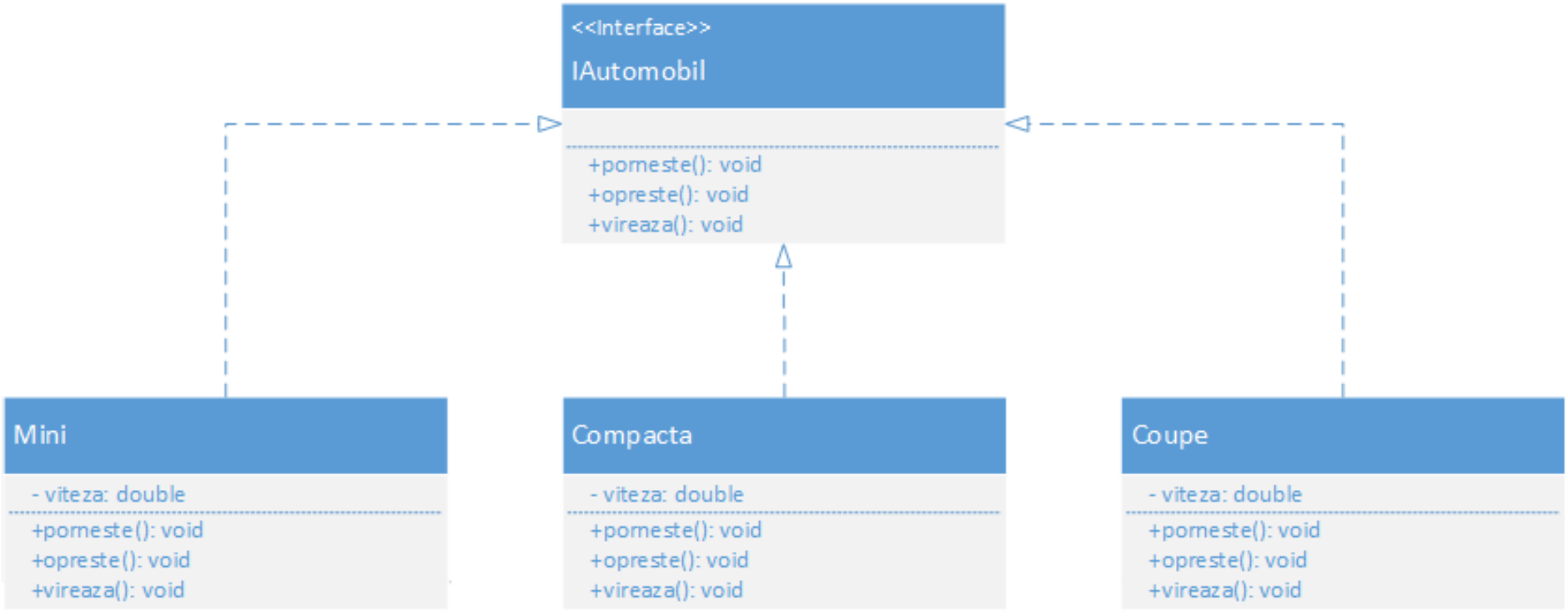
Diagrame UML

- Tipuri de relații. Agregarea:



Diagrame UML

- Diagrama UML pentru exemplul dat:



Exemple/exerciții

- Definiți și implementați interfețele pentru:
  - Stiva.
  - Lista simplu/dublu inlantuita.
  - Persoana.
  - FormaGeometrica.
  - Laptop.
  - Senzor.

Implementarea mai multor interfețe

- O clasă permite implementarea mai multor interfețe – specificarea interfețelor multiple se realizează prin separarea cu virgulă.

- Exemplu:
  - Interfețele I1, I2.
- Conflictul de nume:
  - Două interfețe definesc aceeași metodă -> eroare de compilare

Interfața Comparable

- Definește o singură metodă:
  - int compareTo(Object o).

- Permite definirea unei metode pentru compararea a două obiecte.
- Metoda returnează:
  - -1: o1 < o2
  - 0: stare(o1) == stare(o2)
  - +1: o1>o2.

- Exemplu implementare interfața Comparable în cazul entității Persoana.

Polimorfism

- Polimorfismul reprezintă o caracteristică esențială a POO.
- Reprezintă capacitatea unei entități de a lua forme diferite.
  - Este posibilă prin capacitatea limbajelor OO de a face legătura dintre variabilă și obiect prin *legare întârziată*.
  - Legarea întârziată identifică adresa metodei la apelul acesteia.
  - Spre deosebire, legarea timpurie identifică adresa metodei în momentul compilării.

Polimorfism

- Exemplu:
  - Definirea interfeței IStack.
  - Trei implementări: VectorStack, SimpleLStack, DoubleLStack.

Clase imbricate

- O clasă definită în interiorul unei alte clase este o clasă imbricată.
- Terminologie:
  - Clasa definită în interior: clasa imbricată.
  - Clasa în care este definită: clasa acoperitoare.
- După poziționare, clasele imbricate sunt clasificate:
  - La nivel de clasă.
  - La nivel de metodă.
  - La nivel de instrucțiune (clase anonime).

Clase imbricate – def. la nivel de clasă

- Clasa imbricată are acces nerestricționat la toate variabilele clasei de acoperire (indiferent de modificatorii de acces).
- Clasa imbricată există doar în contextul unei instanțe ale clasei acoperitoare.
- Modificatorii de acces ce se aplică asupra unei clase imbricate sunt aceiași ca și în cadrul membrilor unei clase (public, protected, private, implicit).
- Exemplu instanțiere:
  - class A {
    - class B {}
  - }
  - A.B v ref = new A.new B();

Clase imbricate – def. la nivel de metodă

- Clasa imbricată definită în interiorul unei metode va avea acces numai la variabilele finale (constante).
- Poate fi instanțiată doar din metoda în care a fost declarată, permite accesarea tuturor membrilor clasei acoperitoare.
- Exemplu instanțiere:
  - class A {
    - void metoda() {
      - class B {}
      - B ref = new B();
    - }

Clase imbricate – def. la nivel de instrucțiune

- Clasa imbricată definită în interiorul unei instrucțiuni utilizând o clasă *anonimă*.
- Clasele sunt definite și instanțiate o singură dată, în instrucțiunea respectivă.
- Se folosesc clasele anonime întrucât este inutilă definirea unei noi clase, având în vedere utilizarea exclusivă din instrucțiunea curentă.
- Exemplu:
  - buton.addActionListener(new ActionListener() {
    - public void actionPerformed(ActionEvent e) {

      …

      }});

Alte tipuri de clase imbricate

- Clase imbricate statice: au rolul de a ascunde starea clasei acoperitoare de clasa imbricată (interzice acces la membrii de instanță):
  - Clasa imbricată va avea acces doar la membrii de clasă a clasei acoperitoare.
- Clasele sunt definite și instanțiate o singură dată, în instrucțiunea respectivă.
- Se folosesc clasele anonime întrucât este inutilă definirea unei noi clase, având în vedere utilizarea exclusivă din instrucțiunea curentă.
- Exemplu:
  - buton.addActionListener(new ActionListener() {
    - public void actionPerformed(ActionEvent e) {

      …

      }});