Curs 13 POO
© Crenguta M. Puchianu

- Clase si interfete generice
- Colectii. Interfata Collection<E>
- Interfata Set<E>. Clasa TreeSet<E>
- Interfata List<E>. Clasa ArrayList<E>
- Interfata Map<K,V>. Clasele HashMap TreeMap si Properties
- Tipuri enumerate

Clase generice

- O clasă generică sau parametrizată este o clasă a cărei structură şi funcţionalitate sunt generale, adică nu depind de un anumit tip de date. Acest tip de date se numeşte tip generic formal şi este specificat în declaraţia clasei generice cuprins între paranteze unghiulare (< >).

```
public class ClasaGenerica<T> {
 private T numeVariabila;
 public ClasaGenerica(T nv) {
  numeVariabila = nv;
 }
 public T getValoareVariabila() {
  return numeVariabila;
 }
 public void setValoareVariabila(T valoareNoua){
  numeVariabila=valoareNoua;
 }
}
```

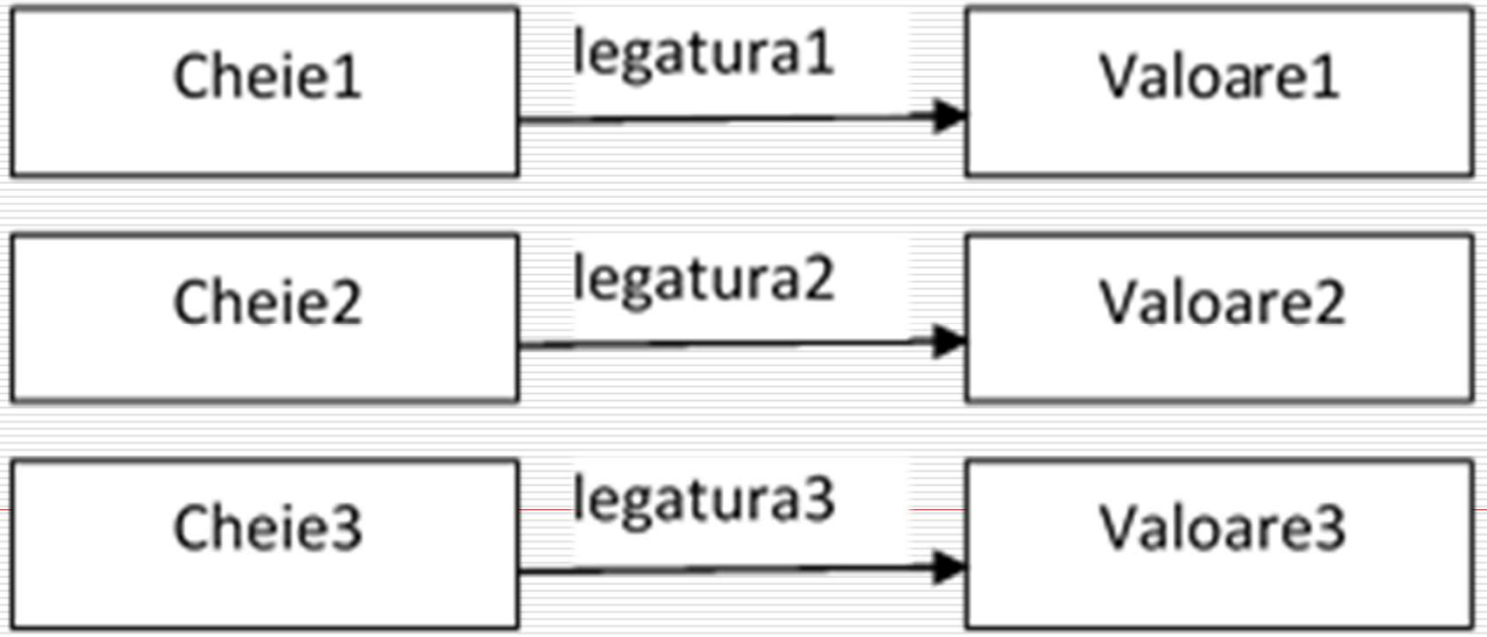
Instantierea claselor generice

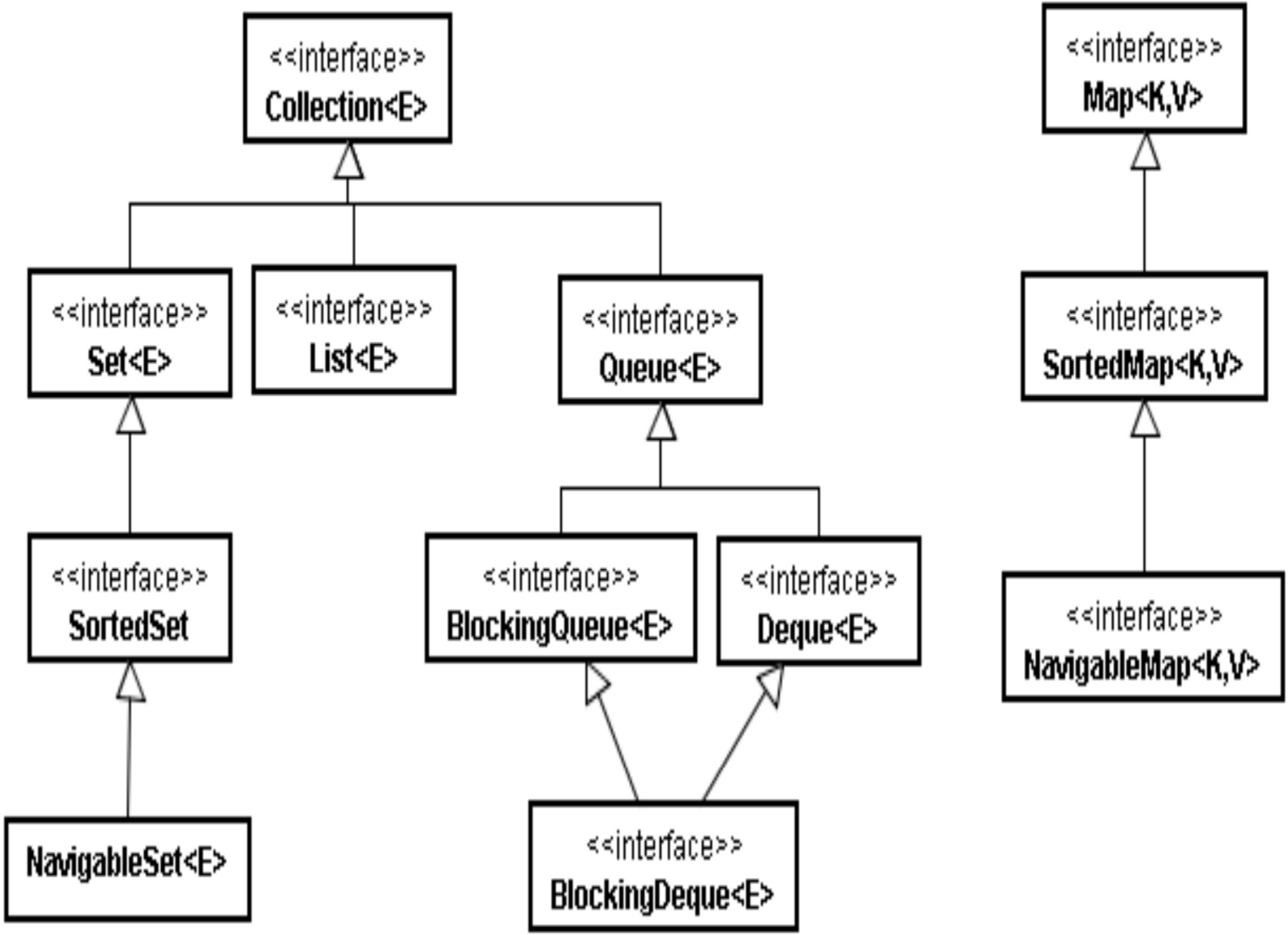
O clasă generică este instanţiată prin specificarea unui tip concret în locul tipului generic formal şi apelarea constructorului clasei. De exemplu:

```
 public static void main(String[] args){

  ClasaGenerica<Integer> ci=new ClasaGenerica<>(5);
  System.out.println(ci.getValoareVariabila());
  ci.setValoareVariabila(7);
  System.out.println(ci.getValoareVariabila());

  ClasaGenerica<Double> cd=new ClasaGenerica<>(7.5);
  System.out.println(cd.getValoareVariabila());
  cd.setValoareVariabila(9.78);
  System.out.println(cd.getValoareVariabila());
 }
}
```

Collections Framework

- Java furnizează o structură de clase şi interfeţe numită Collections Framework specializată în memorarea şi elaborarea colecţiilor de obiecte.
- Face parte din pachetul java.util.
- La baza teoriei colecţiilor stă conceptul matematic de mulţime, chiar dacă colecţia a fost definită având o semnificaţie mai generală, de grup de elemente.
- Un caz particular de mulţime este relaţia funcţională sau maparea:




<<interface>>
Collection<E>
<<interface>>
Map<K,V>
<<interface>>
Set<E>
<<interface>>
List<E>
<<interface>>
Queue<E>
<<interface>>
SortedMap<K,V>
<<interface>>
SortedSet
<<interface>>
BlockingQueue<E>
<<interface>>
Deque<E>
<<interface>>
NavigableMap<K,V>
NavigableSet<E>
<<interface>>
BlockingDeque<E>

Interfata Collection<E>

Prototip metodă	Semnificaţie
boolean add(E o)	Verifică dacă colecţia conţine elementul o. În caz pozitiv, nu mai adaugă elementul o şi întoarce false. În caz negativ, adaugă elementul la colecţie şi returnează true.
boolean remove(Object o)	Dacă elementul o se găseşte in colecţie, este eliminat şi metoda returnează true. Altfel, metoda returnează false.
boolean contains(Object o)	Returnează true dacă colecţia conţine elementul o.
Iterator<E> iterator()	Întoarce un obiect de tip Iterator care se poate utiliza pentru navigarea colecţiei.
int size()	Întoarce numărul de elemente ale colecţiei.
boolean isEmpty()	Returnează true dacă colecţia nu are elemente.
clear()	Goleste colectia
Object[] toArray()	Returnează un tablou ce conţine toate elementele colecţiei.

Interfata Iterator<E>

Prototip metodă	Semnificaţie
boolean hasNext()	Verifică dacă colecţia mai are elemente de parcurs.
E next()	Returnează următorul element al colecţiei.
void remove()	Poate fi apelată numai o singură data pentru fiecare obiect obţinut cu metoda next().

Interfata Set<E>. Clasa TreeSet<E>

- Clasa garantează că mulţimea continută este ordonată în ordine crescătoare. Ordonarea este cea naturală între elemente (care trebuie să implementeze Comparable) sau va fi specificată prin intermediul unui obiect comparator (face parte dintr-o clasă care realizează interfaţa Comparator) la crearea lui TreeSet.

Prototip constructor	Semnificaţie
TreeSet()	Creează o colecţie TreeSet vidă.
TreeSet(Collection<? extends E> c)	Creează o colecţie TreeSet ce conţine elementele colecţiei c.
TreeSet(Comparator<? super E> comp)	Creează o colecţie TreeSet vidă ce foloseşte un comparator de orice tip care este supraclasa lui E.
TreeSet(SortedSet<E> ss)	Creează o colecţie TreeSet ce conţine elementele mulţimii ss.

Interfata List<E>

- Interfaţa reprezintă o *secvenţă*. Prin intermediul acestei interfeţe, utilizatorul are un control exact al locului în care este inserat fiecare element. Utilizatorul poate accesa elementele prin intermediul iteratorilor şi a indicilor întregi care reprezintă poziţia în listă.
- Listele acceptă elemente duplicat şi elemente null.

Prototip metoda	Semnificaţie
Object get(int indice)	Întoarce obiectul de la indicele transmis ca parametru.
void add(int indice, Object o)	Inserează obiectul o la poziţia indice.
Object remove(int indice)	Elimină obiectul de la poziţia indice.
E set(int indice, Object o)	Înlocuieşte obiectul de la poziţia indice cu obiectul o.
int indexOf(Object o)	Returnează poziţia obiectului o din listă. Dacă nu-l găseşte, returnează -1.
int lastIndexOf(Object o)	Returnează poziţia ultimei apariţii a obiectului o din listă. Dacă nu-l găseşte, returnează -1.
ListIterator<E> listIterator()	Returnează un obiect de tip ListIterator al lui List.

Interfata ListIterator<E>

- Interfaţa ne permite să traversăm lista în ambele direcţii, să o modificăm în timpul traversării şi să obţinem poziţia curentă a iteratorului.

Prototip metoda	Semnificaţie
void add(Object elem)	Adaugă un element în poziţia curentă.
Object previous()	Poziţionează cursorul şi returnează elementul precendent.
boolean hasPrevious()	Verifică dacă există un element precedent.
void set(Object elem)	Înlocuieşte elementul curent (adică cel care a fost returnat prin apelarea metodei next() sau previous()) cu elementul transmis ca parametru.
int nextIndex()	Returnează indicele elementului succesiv.
int previousIndex()	Returnează indicele elementului precedent.

Clasa ArrayList<E>

- Este o clasă concretă care implementează interfaţa List şi se sprijină pe un tablou fără limite de creştere. Modelează o listă cu elemente enumerate, deoarece are un acces obişnuit, direct, relativ rapid.

Prototip metoda	Semnificaţie
ArrayList()	construieşte un ArrayList vid de capacitate 10 elemente
ArrayList(int n)	construieşte un ArrayList vid de capacitate valoarea lui n
ArrayList(Collection<? extends E> c)	construieşte un ArrayList cu elementele colecţiei c
void trimToSize()	elimină elementele nule din colectie
void clone()	clonează obiectul ArrayList

Clasa ArrayDeque<E>

- Este o clasă concretă care implementează interfaţa Deque pentru a furniza colecti bazate pe tablouri de obiecte

Prototip metoda	Semnificaţie
ArrayDeque()	construieşte un ArrayDeque vid de capacitate 16 elemente
ArrayDeque(int n)	construieşte un ArrayDeque vid de capacitate valoarea lui n
ArrayDeque(Collection<? extends E> c)	construieşte un ArrayDeque cu elementele colecţiei c
boolean add(E e)	Adauga elementul e la sfarsitul cozii implementata de acest obiect Deque
void push(E e)	Adauga elementul e la inceputul stivei implementata de acest obiect Deque

Interfata Map<K,V>

Interfaţa păstrează o colectie de legături chei-valoare şi permite căutări bazate pe chei.

Prototip metoda	Semnificaţie
V get(Object cheie)	Întoarce obiectul cu cheia transmisă ca parametru.
V put(K cheie, V valoare)	Inserează legătura cheie-valoare transmisa ca parametru.
V remove(Object cheie)	Elimină legătura indicată de cheie.
boolean containsKey(Object cheie)	Verifică în Map existenţa cheii transmisa ca parametru.
boolean containsValue(Object o)	Verifică existenţa valorii o în Map.
int size()	Returnează numărul de legături.
Set<Map.Entry<K,V>> entrySet()	Furnizează un obiect de tip Set ce conţine toate legăturile cheie-valoare. Fiecare legătură este văzută ca un element de tip Map.Entry<K,V>
Set<K> keySet()	Furnizează un Set compus din chei.
Collection<V> values()	Furnizează o colecţie a tuturor valorilor din Map.

Clasele HashMap<K,V> si TreeMap<K,V>

- Spre deosebire de HashMap, clasa TreeMap contine colecţii de legături chei-valoare ordonate după chei, implementând şi interfaţa SortedMap. Implicit, ordonarea este crescătoare (conform interfeţei Comparable), dar criteriul de comparare poate fi specificat prin constructor.

Prototip metoda	Semnificaţie
TreeMap (Comparator<? super K> c)	creează o colecţie vidă ale cărei elemente sunt ordonate conform comparatorului c
TreeMap()	creează o colecţie vidă ordonată crescător după chei
TreeMap(Map<? extends K, ? extends V> m)	creează o colecţie ce conţine legăturile colecţiei m
TreeMap(SortedMap<K, ? extends V> m)	creează o colecţie ce conţine legăturile colecţiei m

Clasa Properties<K,V>

- Furnizează o colecţie persistentă de legături chei-valori în care cheile şi valorile sunt obiecte de tip String. Din acest motiv, valorile se numesc proprietăţi.

Prototip metoda	Semnificaţie
Object setProperty(String cheie, String valoare)	Adauga proprietatea cheie-valoare transmisa ca parametru la colectia curenta
String getProperty(String cheie)	Returneaza valoarea proprietatii cu cheia transmisa ca parametru
public void store(OutputStream out, String header) public void list(PrintStream out) public void list(PrintWriter out)	Memoreaza proprietatile colectiei intr-un fisier.
public void load(InputStream in)	Incarca proprietarile colectiei din fisierul transmis ca parametru.

Tipuri de date enumerate

DeclaratieTipEnum ::= [**public**|**private**][**static**] **enum** *Identificator* [**implements** *ListaIdentificatoriInterfata*] *Bloc*

ListaIdentificatoriInterfata ::= *ListaIdentificatoriInterfata*, *IdentificatorInterfata* | *IdentificatorInterfata*

Bloc ::= *ListaIdentificatoriConstanta*; { *DeclaratieVariabilă* | *DeclaraţieMetoda*}

ListaIdentificatoriConstanta ::= *ListaIdentificatoriConstanta*, *IdentificatorConstanta*[(*Valoare*)] | *IdentificatorConstanta*[(*Valoare*)]

Exemplu:

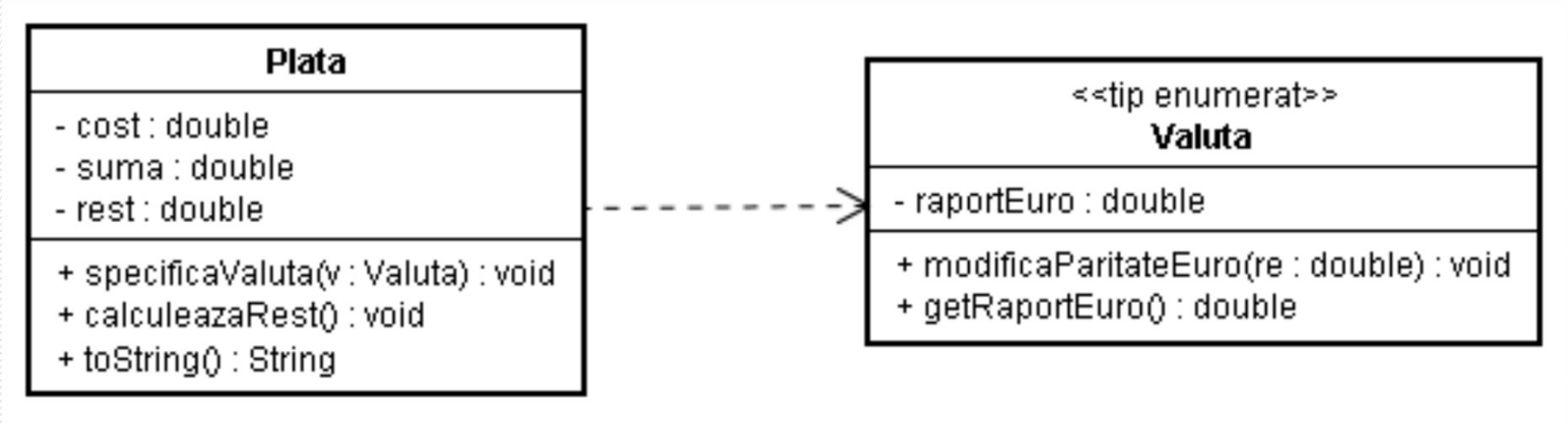
```
public enum Valuta1 {
    LEU(4.95),DOLAR(1.02),EURO(1);
    private double raporteuro;
    Valuta1(double re){
        raporteuro=re;
    }
    public double getRaport(){        return this.raporteuro;    }

    public void setRaport(double r){        this.raporteuro=r;    }

     public static void main(String[] args){
         Valuta1.DOLAR.setRaport(1.2);
         System.out.println(Valuta1.DOLAR.getRaport());
     }
}
```

Folosirea tipurilor de date enumerate

Un tip de date enumerat poate fi folosit la fel ca şi o clasă, mai precis în următoarele cazuri:

- definirea unei variabile de tip enumerat;
- declararea unui parametru formal al unei metode de tip enumerat;
- declararea tipului rezultatului returnat de către o metodă ca fiind un tip enumerat;
- un tip enumerat poate fi declarat într-o altă clasă.

Exemplu:



Clasa Enum

Prototip metodă	Semnificaţie – valoarea returnată
final String name()	Returnează numele constantei curente a tipului enumerat. De exemplu, apelul Valuta.EURO.name() returnează EURO.
String toString()	Clasa Enum redefineşte metoda toString() moştenită de la clasa Object. La fel ca name(), metoda returnează numele constantei curente a tipului enumerat.
static <T extends Enum<T>> T valueOf(Class<T> tipEnum, String n)	Returnează constanta cu numele n a tipului enumerat tipEnum transmis ca parametru.
final int ordinal()	Returnează numărul de ordine a constantei curente a tipului enumerat. Prima constantă declarată primeşte numărul de ordine 0, a doua constantă are 1, samd.
final int compareTo(E o)	Compară două constante ale tipului enumerat curent. Comparaţia se face după numărul de ordine a celor două constante.
Object[] values()	Returnează un tablou al constantelor tipului enumerat