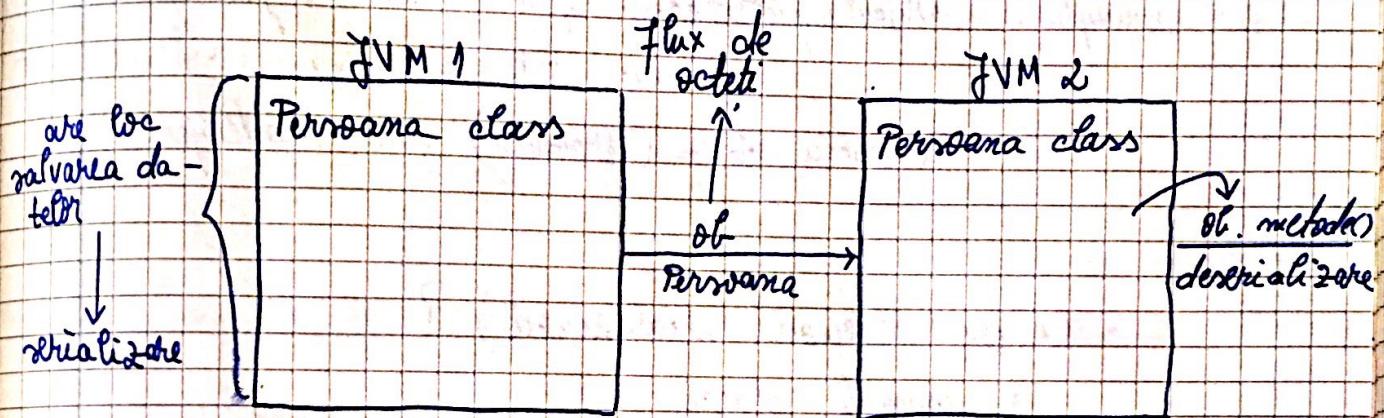


- CURS # -

- 1) Serializare
- 2) Colectiv de date (Collection Framework)

1) Serializare



Serializare → presupune transformarea obiectului într-un zecimal binar

Deserializare → Refacerea obiectului din zecimal binar

Serializarea

Pașul 1: clasa trebuie să implementeze interfața `java.io.Serializable` (marker)

Pașul 2: se deschide un flux de procesare Object OutputStream

Exemplu: `ObjectOutputStream fout = new ObjectOutputStream
(new FileOutputStream("obiecte.out"));`

Pașul 3: se apelaază metoda `void writeObject(Object ob);`

```
Persoana ob = new Persoana ("Matei", 13);  
fout.writeObject (ob);
```

Deserializarea

- 1) se deschide fluxul de procesare ObjectInputStream.
- 2) se apelaază metoda Object readObject();

Exemplu . ObjectInputStream fin = new ObjectInputStream(
new FileInputStream ("obiecte.out"));
Persoana ob = (Persoana) fin.readObject();

• Ce se salvează (serializază) ?

1. numele clasei
2. versiunea clasei (hashcode - ul clasei, sensul la care modificare a clasei)
3. datele membre de instanță (nu conțină dacă sunt private, că oricum sunt salvate read-only)
4. numărul metodelor membre

• Ce NU se salvează ?

1. datele membre statice
2. corpul metodelor
3. datele membre de instanță marcate transient

OBSERVAȚII: 1) Dacă se modifică antetul unei metode sau se adaugă/sterge o dată membru de instanță \Rightarrow \rightarrow la deserializare InvalidClassException. Pentru a evita acest lucru, se poate înlocui versiunea clasică implicită prin câmpul privat serialVersionUID. exemplu: private static final long serialVersionUID = 1L;

2) La serializarea unui obiect se salvează intreg graful asociat.

class Persoana { ... , Adresa ab; ... }
 \downarrow nă ţe serializat!!

În caz contrar, se obține o excepție NoSerializableException.

3) Dacă se serializează un obiect de tip subclasa și superclasa nu este serializabilă

\downarrow
nu se vor salva datele membre preluate din superclasa !

2) Collecții de date (java.util Collection)

\rightarrow definirea în managementul structurilor de date dinamice

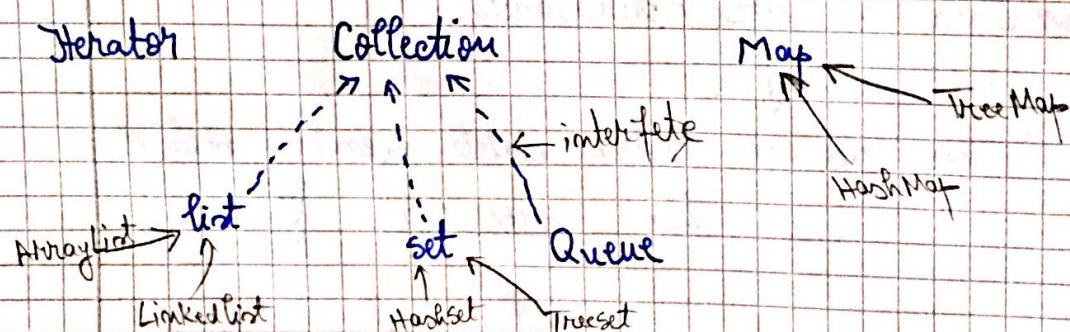
Apare noțiunea de colecție (definire): este un obiect de tip container care conține mai multe elemente grupate într-o unitate.

\rightarrow elementele se află într-o ordine specifică unei structuri de date (listă, stivă, arbori, tabele de asociere etc.)

→ accesarea unei colecții se poate efectua operări: (adăugare, modificare, ștergere, inserare, sortare etc.)

Arhitectura Collection

- 1) interfețe: implementările abstracte ale structurilor de date
 - 2) Clase concrete: → implementări ale interfețelor
→ începând cu versiunea 1.5 clasele sunt genetice:
 - se specifică tipul elem. fol. < > (ArrayList< Person > tp = new ArrayList< ());)
- algoritmi polimorfici: metode statice (Collection) care implementează optime operațiuni specifice: sortare, căutare, min, max etc.



1. interfața Collection → modelizează op. specifice
oricărei colecții

→ metode la nivel de colecție: size(), clear(), iterator()

→ metode la nivel de element: void add(T elem)

void addAll(Collection< T> col)

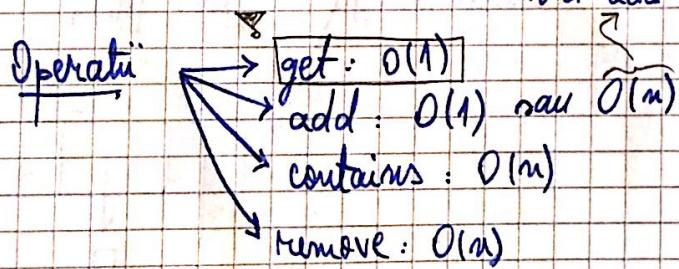
boolean contains(T elem)

void remove(T elem)

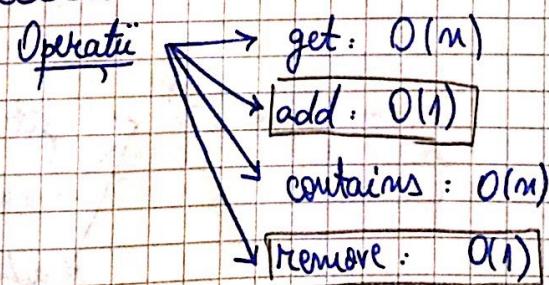
1.a) interfață list → modelază o structură de date ordonată, care permite duplicate!
 implementări: Arraylist, LinkedList (courți fără clase)
 • ArrayList ← modelază lista sub forma unui tablou redimensionabil

↳ argumente (dimensiunea este implicită), constructor cu arg. → capacitatea și colectie)

← se adaugă metode suplimentare (fără de Collection):
 - get (int index)
 void set (int index, T val)
 void add (T val, int index)



• LinkedList → lista dublu întărită



← se adaugă metode suplimentare:
 addFirst (T elem)

addLast (T elem)

- getFirst ()

- getLast ()

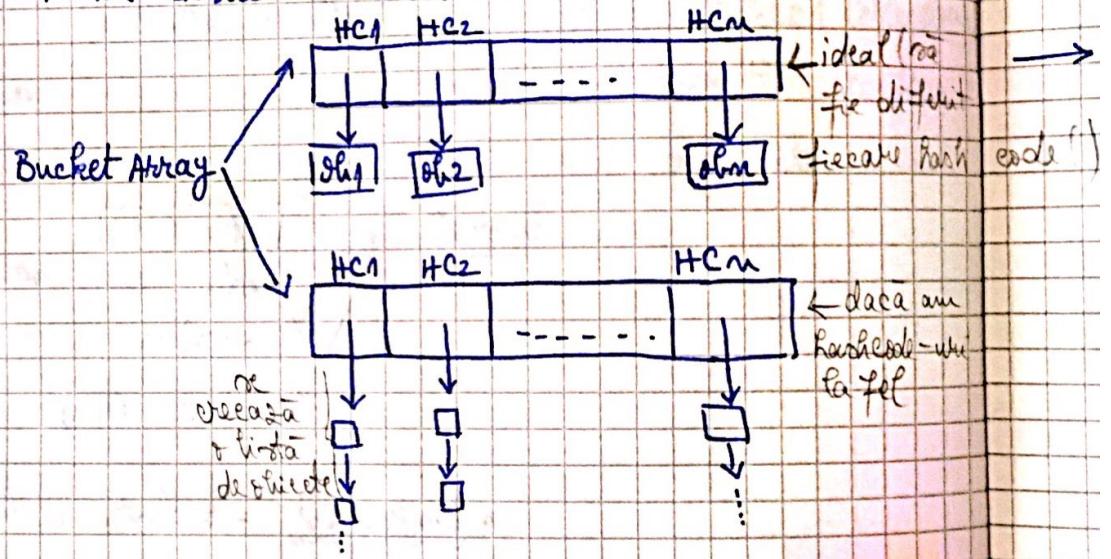
Exemplu:
`ArrayList<integer> li = new ArrayList<()>;` $\approx 10 \text{ elem}$
`li.add(7);`

1. a) interfața set \rightarrow modelază o structură de date,
 fără duplicate!

implementări: HashSet, TreeSet

- HashSet: intern, structura are asociat un tablou, înrău indexul tabloului este dat de hashCode-ul elementului (Bucket Array)

\rightarrow nu există ordine (de inserare)



\rightarrow Operării

- get: O(1)
- contains: O(1)
- add: O(1)

P1: se identifică bucket-ul prin metoda hashCode

OBS. Operatiile pot degenera în O(n) dacă există coliziuni!!

P2: se folosesc metoda equals()
 pt. a găsi elem. (se cauță în bucket-ul P1)

OBS: Clasa obiectelor din structuri TREBUIE să implementeze met. hashCode() și equals().

Exemplu: → structură cu nr. din fișier distincte :

```
Scanner fin = new Scanner(new File("numere.txt"));
HashSet <integer> mi = new HashSet <> ();
int x;
while (!fin.hasNextInt())
{
    x = fin.nextInt();
    mi.add(x); → O(1)
}
```

→ Dacă vrem să eliminăm duplicatele dintr-o structură :

HashSet <integer> mi = new HashSet <> (ti);

ArrayList <integer> ti = new ArrayList();

ti.add(1);

ti.add(1);

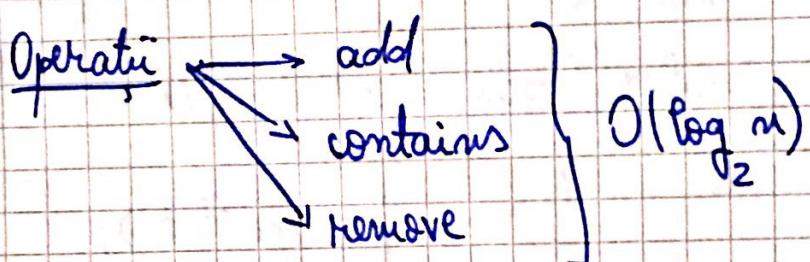
OBS: 1. capacitatea implicită pentru HashSet este 16
(nr. de buckets)

→ ②. factorul de umplere (load factor) $\Rightarrow 0.75$ (dacă este depășit, se dublează capacitatea)

• TreeSet → elemente ordonate (sortate) după un anumit criteriu

⇒ clara implementarea interfeței Comparable sau Comparator

→ intern, se folosește un arbori negru - negru



1.c) interfața Map: colecție cu elemente de tip pereche \langle cheie, valoare \rangle

Exemplu: Persoana → nr. tel

Cuvântul → frecvența de apariție

→ metode:

```
graph LR; Metode[Metode] --- put["void put(T cheie, R val)"]; Metode --- get["R get(T cheie)"]; Metode --- containsKey["boolean containsKey(T cheie)"]
```