Curs 4 PPOO

Prof. univ. dr. Cristian CIUREA

Departamentul de Informatică și Cibernetică Economică

cristian.ciurea@ie.ase.ro

Java fundamentals

- Colecții în Java (**Java Collections Framework**)

Colecții

Java Collections Framework este o arhitectură unificată pentru reprezentarea și manipularea colecțiilor ce conține:

- **interfețe**: permit colecțiilor să fie folosite independent de implementările lor;
- **implementări;**
- **algoritmi**: metode de prelucrare (căutare, sortare) pe colecții de obiecte oarecare. Algoritmii sunt *polimorfici*, adică pot fi folosiți pe implementări diferite de colecții, deoarece le abordează la nivel de interfață.

Colecții

Colecțiile oferă implementări pentru următoarele tipuri:

- **mulțime** (ordinea elementelor este neimportantă);
- **listă** (ordinea elementelor contează);
- **tabel asociativ** (perechi cheie-valoare).

și pot lucra cu orice tip de obiecte (implementările sunt *generice*).

Colecții

Tipuri de colecții:

- **sortate** (TreeMap, TreeSet, PriorityQueue);
- **nesortate** (ArrayList, HashMap, Hashtable, etc.);
- **ordonate** (LinkedHashMap, LinkedHashSet, LinkedList, etc.);
- **neordonate** (HashMap, Hashtable, HashSet).

Colecții

Class	Map	Set	List	Ordered	Sorted
HashMap	X				
Hashtable	X				
TreeMap	X			sorted	X
LinkedHashMap	X			by insertion	
HashSet		X			
TreeSet		X		sorted	X
LinkedHashSet		X		by insertion	
ArrayList			X	by index	
Vector			X	by index	
LinkedList			X	by index	
PriorityQueue				sorted	X

Colecții

Tipuri de colecții:

- **Lists** – liste de valori (**ArrayList, Vector, LinkedList**);
- **Sets** – liste de valori unice (**HashSet, TreeSet, LinkedHashSet**);
- **Maps** – liste de valori cu un ID unic (**HashTable, HashMap, TreeMap, LinkedHashMap**);
- **Queues** – liste de valori procesate într-o ordine specifică.

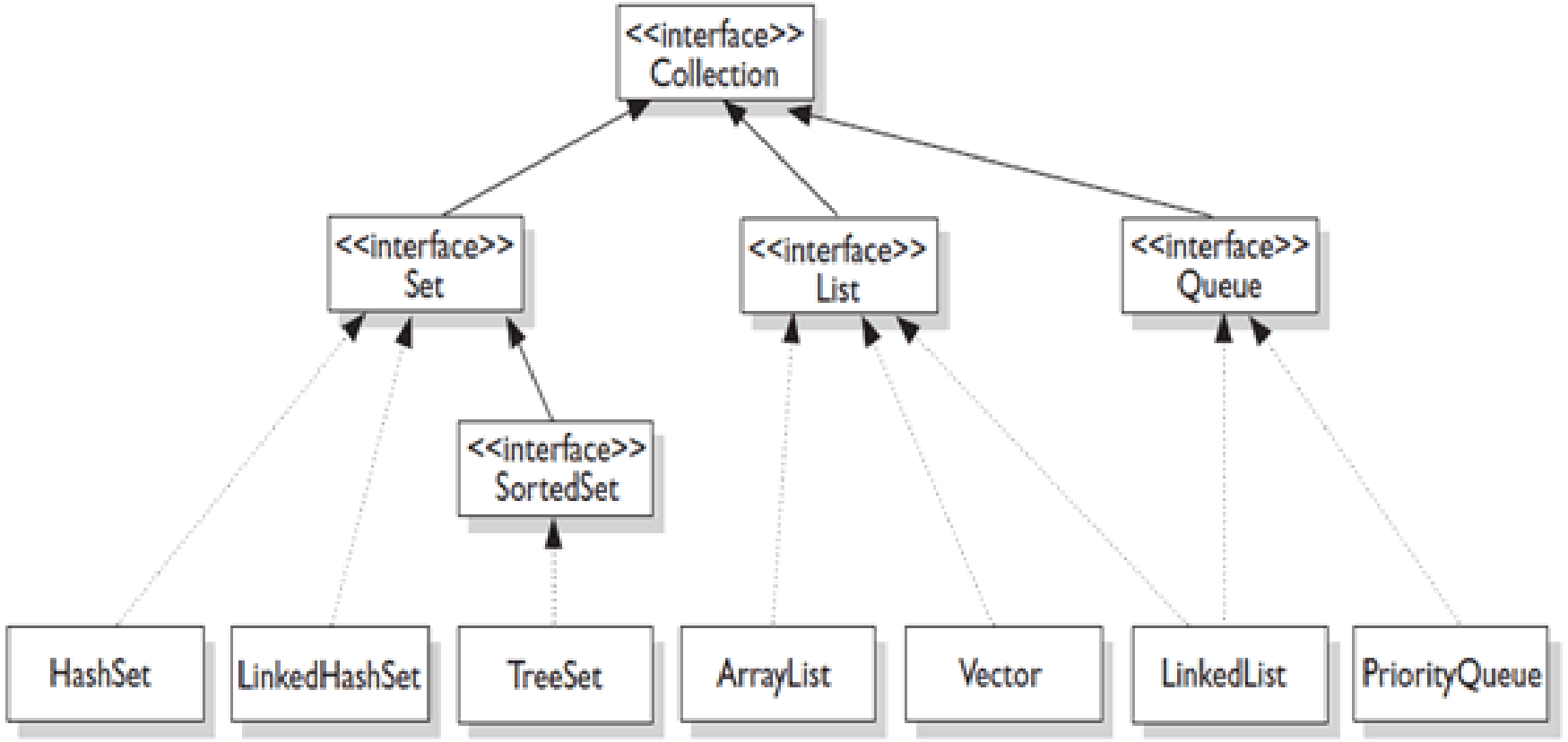
Colecții

ArrayList	LinkedList
1) ArrayList utilizează intern un vector dinamic pentru a stoca elementele.	LinkedList utilizează intern o listă dublu înlănțuită pentru a stoca elementele.
2) Manipularea datelor cu ArrayList este lentă, deoarece utilizează intern un vector. Dacă vreun element este eliminat din vector, toți biții sunt deplasați în memorie.	Manipularea cu LinkedList este mai rapidă decât ArrayList, deoarece folosește o listă dublă, deci nu este necesară schimbarea de biți în memorie.
3) O clasă ArrayList poate acționa ca o listă, pentru că implementează numai interfața List.	Clasa LinkedList poate acționa ca o listă, dar și ca o coadă, deoarece implementează interfețe List și Deque.
4) ArrayList este mai bună pentru stocarea și accesarea datelor.	LinkedList este mai bună pentru manipularea datelor.

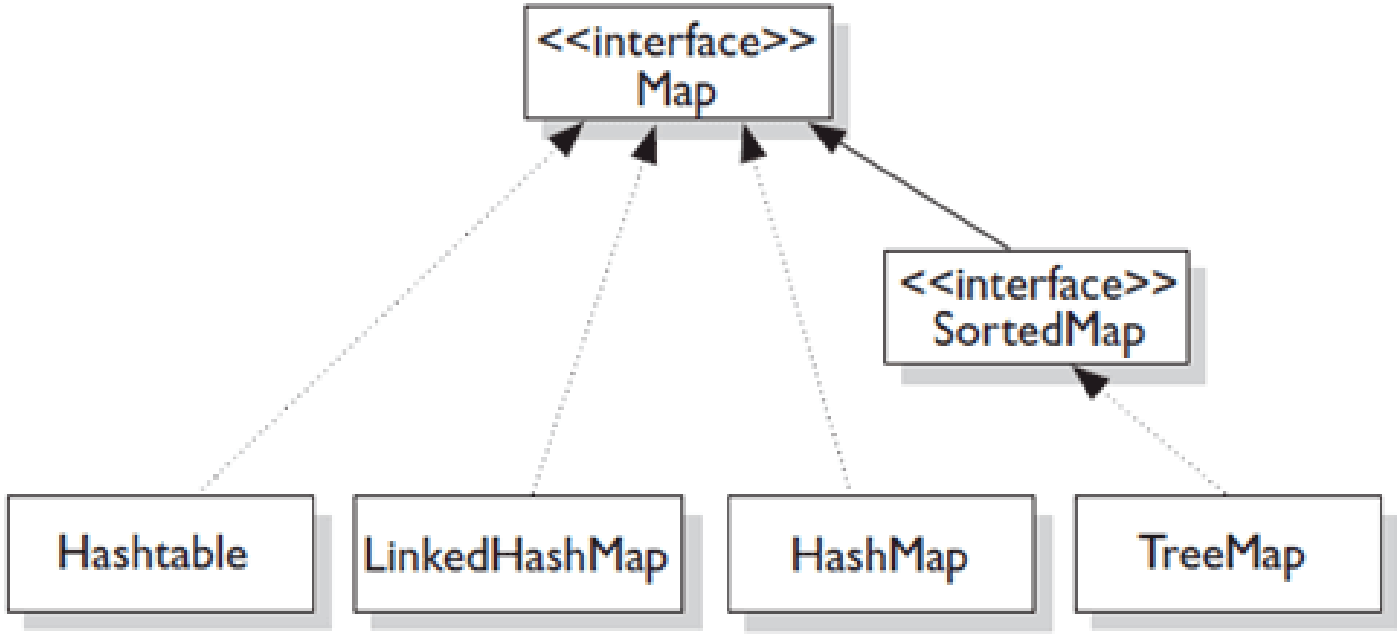
Colecții

- Ierarhia de colecții din Java este formată din două categorii distincte de interfețe.
- Prima categorie asigură funcționalitățile de bază utilizate de toate colecțiile, cum ar fi metode de adăugare și/sau eliminare de elemente.
- Subinterfețele sale - *Set*, *List* și *Queue* - furnizează metode pentru o serie de colecții specializate.
- Cea de-a doua categorie include interfața *Map* care mapează perechi cheie - valoare similar cu o tabelă de dispersie.

Colecții


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
<<interface>>
SortedSet
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue

Colecții


<<interface>>
Map
<<interface>>
SortedMap
Hashtable
LinkedHashMap
HashMap
TreeMap

Colecții

- Interfața **List** oferă metode pentru o colecție ordonată, pentru situațiile în care este nevoie de un control precis asupra locului în care se introduce fiecare element. Se pot localiza elemente dintr-o listă **List** după poziția lor exactă.
- O listă este o colecție ordonată. Listele pot conține elemente duplicate. Pe lângă operațiile moștenite de la interfața **Collection**, interfața **List** definește următoarele operații:
  - **T get(int index)** - întoarce elementul de la pozitia index;
  - **T set(int index, T element)** - modifica elementul de la pozitia index;
  - **void add(int index, T element)** - adauga un element la pozitia index:
  - **T remove(int index)** - sterge elementul de la pozitia index.

Colecții

- Interfața **Set** nu permite elemente duplicate și are o subinterfață, **SortedSet**, care oferă metode pentru ordonarea elementelor din structură.
- Exista trei implementări utile pentru **Set**:
  - **HashSet**: memorează elementele într-o tabelă de dispersie; este implementarea cea mai performantă, însă nu există garanții asupra ordinii de parcurgere.
  - **TreeSet**: memorează elementele sub formă de arbore roșu-negru; elementele sunt ordonate pe baza valorilor, iar implementarea este mai lentă decât HashSet.
  - **LinkedHashSet**: este implementat ca o tabelă de dispersie, diferența față de HashSet este că LinkedHashSet menține o listă dublu-înlănțuită peste toate elementele sale; spre deosebire de HashSet, elementele rămân în ordinea în care au fost inserate; o parcurgere a LinkedHashSet va găsi elementele mereu în această ordine.

Colecții

- Un **Map** este un obiect care mapează chei pe valori. Într-o astfel de structură nu există chei duplicate. Fiecare cheie este mapată la exact o valoare. **Map** reprezintă o modelare a conceptului de funcție: primește o entitate ca parametru (cheia) și întoarce o altă entitate (valoarea).
- Subinterfața **SortedMap** menține perechile sale de tipul cheie-valoare în ordine crescătoare sau într-o anumită ordine specificată de un **Comparator**.
- Cele trei implementari pentru **Map** sunt:
  - **HashMap**
  - **TreeMap**
  - **LinkedHashMap**

Colecții

- Interfața **Queue** permite operații suplimentare de inserare, extragere și inspecție. Elementele dintr-o coadă sunt ordonate pe baza principiului FIFO.
- Implementări utilizate frecvent pentru **Queue**:
  - **LinkedList**: pe lângă List, clasa LinkedList implementează și interfața Queue
  - **PriorityQueue**;

Colecții

- Pentru a putea utiliza propriile obiecte în cadrul colecțiilor este nevoie de suprascrierea metodelor (din clasa *Object*):
  - **boolean equals(Object obj)**
  - **int hashCode()**
- Pentru sortarea obiectelor în cadrul colecției este nevoie de implementarea interfețelor:
  - **Comparable** (metoda **int compareTo(Object obj)**) sau
  - **Comparator** (metoda **int compare(Object one, Object two)**).

Colecții

- Spre deosebire de **Comparable** care, implementată de o clasă, marchează faptul ca instanțele sale sunt comparabile, **Comparator** desemnează o entitate externă care realizează o comparație între două obiecte oarecare.
- Intuitiv, faptul că o clasă implementează **Comparator** înseamnă că ea se comportă ca un comparator pentru obiecte de un anumit tip.

Colecții

Colecțiile pot fi parcurse (element cu element) folosind:

- **iteratori**;
- o construcție *for* specială, cunoscută sub denumirea de **for-each** sau **enhanced-for**.

Colecții

Iteratorii:

- Sunt obiecte utilizate pentru a gestiona poziția curentă în cadrul unei colecții;
- Un iterator este un obiect care permite traversarea unei colecții si modificarea acesteia (ex: ștergere de elemente) în mod selectiv;
- Clasele utilizate sunt **Interator** și **ListIterator**;
- Pentru a defini un iterator peste o colecție este nevoie de implementarea interfețelor **Iterable** sau **Iterator**.

Colecții

Algoritmi de prelucrare:

- Sunt algoritmii polimorfici care implementează diferite funcționalități oferite de platforma Java;
- Sunt metode statice în clasa *Collections*;
- În majoritatea cazurilor sunt utilizați cu instanțe de *List*.

Colecții

Algorithms:

Algorithm	Method	Description
Sorting	sort()	Sorts a List by natural order or by a Comparator
Shuffling	shuffle()	Shuffles a Collection
Searching	binarySearch()	Searches a sorted list for a given value
Composition	frequency() disjoint()	The frequency of a given value Number of common elements in 2 collections
Find extreme values	min() max()	

Colecții

Algorithms:

Algorithm	Method	Description
Routine Data Manipulation	reverse()	Reverses the order of the elements in a List
	fill()	Overwrites every element in a List with the specified value
	copy()	Copies a source List into a destination one
	swap()	Swaps the elements at the specified positions in a List
	addAll()	Adds elements to a Collection

Bibliografie

- [1] Jonathan Knudsen, Patrick Niemeyer – *Learning Java, 3rd Edition*, O'Reilly.
- [2] http://www.itcsolutions.eu
- [3] http://www.acs.ase.ro
- [4] http://docs.oracle.com/javase/tutorial/ /index.html
- [5] http://cursuri.cs.pub.ro/~poo/wiki/index.php/Colectii