Aplicatii JAVA

2

JAVA
Colecții în Java

Adrian Runceanu
www.runceanu.ro/adrian

2016

Curs 2

Colecții în Java

6. Colecții

1. Elemente introductive
2. Colecții. Avantaje ale colectiilor
3. Tipuri de colecții
4. Interfețe ce descriu colecții
5. Interfața Collection
    1. Interfața Set
    2. Interfața List
    3. Interfața Map
6. Clasa ArrayList
7. Iteratori

Elemente introductive

Java dispune de 2 metode de pastrare a obiectelor:

1. **Tablourile**, care sunt tipuri predefinite (built-in) ale limbajului.
2. **Librariile (pachetele)** care contin mai multe categorii de clase de **colectii (clase container)** care ofera o mare varietate de modele de date si structuri de reprezentare a acestora.

Într-o colectie, numarul de elemente poate varia prin introducerea unor elemente noi sau prin extragerea unor elemente.

Elemente introductive

Modelele de date cele mai folosite (si implementate în arhitectura colectiilor *Java – Java Collections Framework*) sunt:

1. lista = **list** - o secventa de elemente, memorate într-o anumita ordine
2. multimea = **set** - o grupare de elemente în care nu exista doua sau mai multe elemente de acelasi fel
3. vectorul asociativ = **map** - o grupare de elemente, în care fiecare element contine doua parti asociate, cheia si valoarea

Elemente introductive

La rândul lui, fiecare model de date (forma colectiei) poate fi realizat prin diferite structuri de date.

De exemplu:

- ✓ o lista poate fi realizata printr-un tablou sau printr-o lista înlantuita

- ✓ o multime poate fi realizata printr-o lista înlantuita, printr-un arbore sau printr-o tabela de dispersie etc.

Tehnologii de memorare a datelor

Exista cateva *tehnologii folosite pentru stocarea obiectelor in memorie*:

1. Tablouri (array)
2. Liste inlantuite (linked-lists)
3. Arbori (tree)
4. Tabele de hashing (hash table)

Indiferent ce tehnologie se foloseste, operatiile pentru colectiile de obiecte sunt aceleasi, ele pot fi doar mai performante in functie de tehnologia de stocare

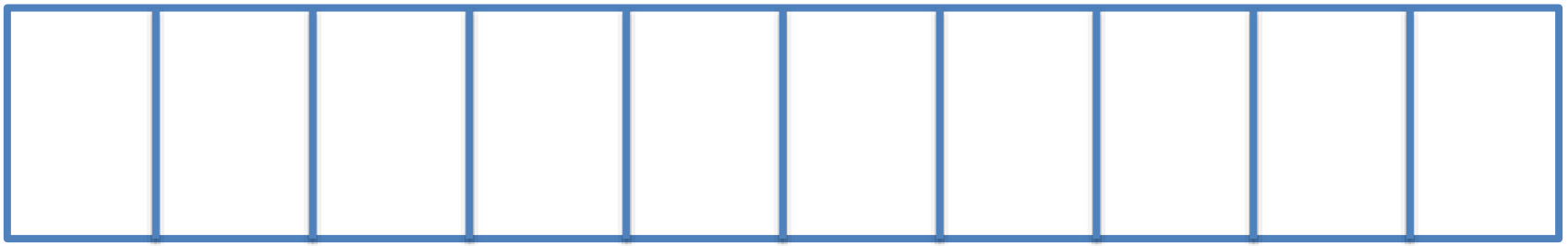
Tehnologii de memorare a datelor

1. Tablouri (array)

- Elementele sunt ordonate si pot exista valori care se repeta (valori duplicate)
- Dimensiunea tabloului este fixa

Avantaj: Este performant la accesarea datelor

Dezavantaj: Este incet la adaugarea si stergerea obiectelor



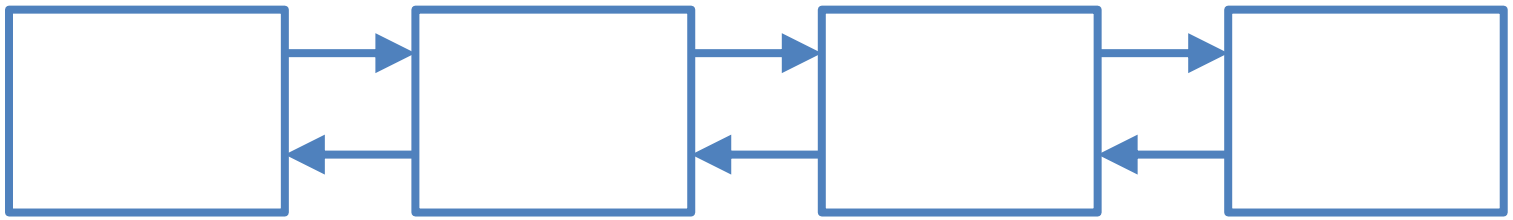
Tehnologii de memorare a datelor

2. Liste inlantuite (linked-lists)

- Elementele sunt ordonate si pot exista valori care se repeta (valori duplicate)
- Dimensiunea este variabila

Avantaj: Adaugarea si stergerea obiectelor este rapida

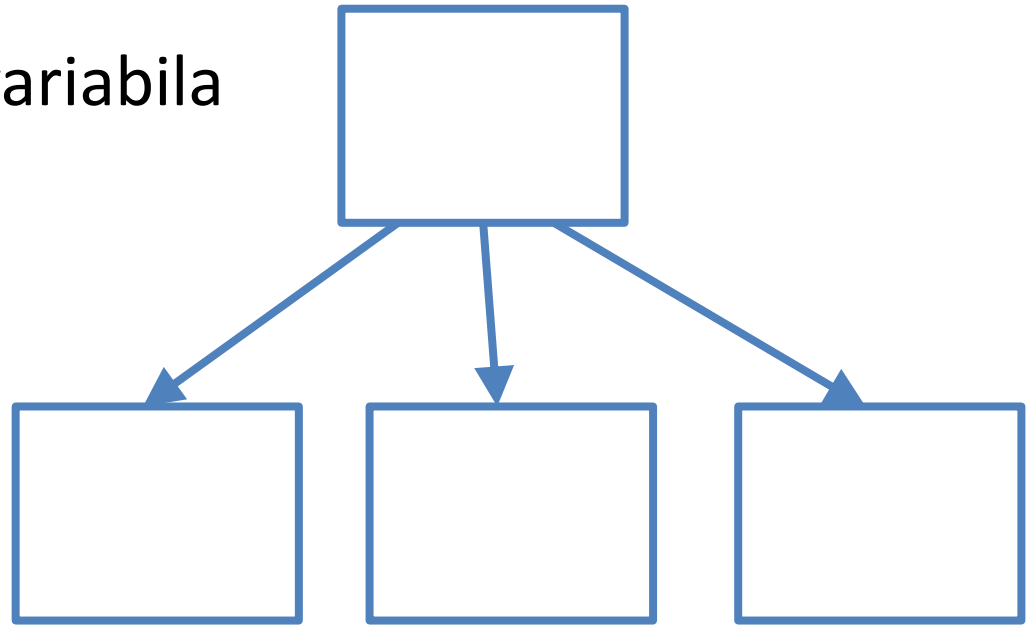
Dezavantaj: Este incet la accesarea datelor



Tehnologii de memorare a datelor

3. Arbori (tree)

- Este o structura neliniara
- Datele sunt **sortate** si nu exista valori care se repeta (duplicate)
- Dimensiunea este variabila

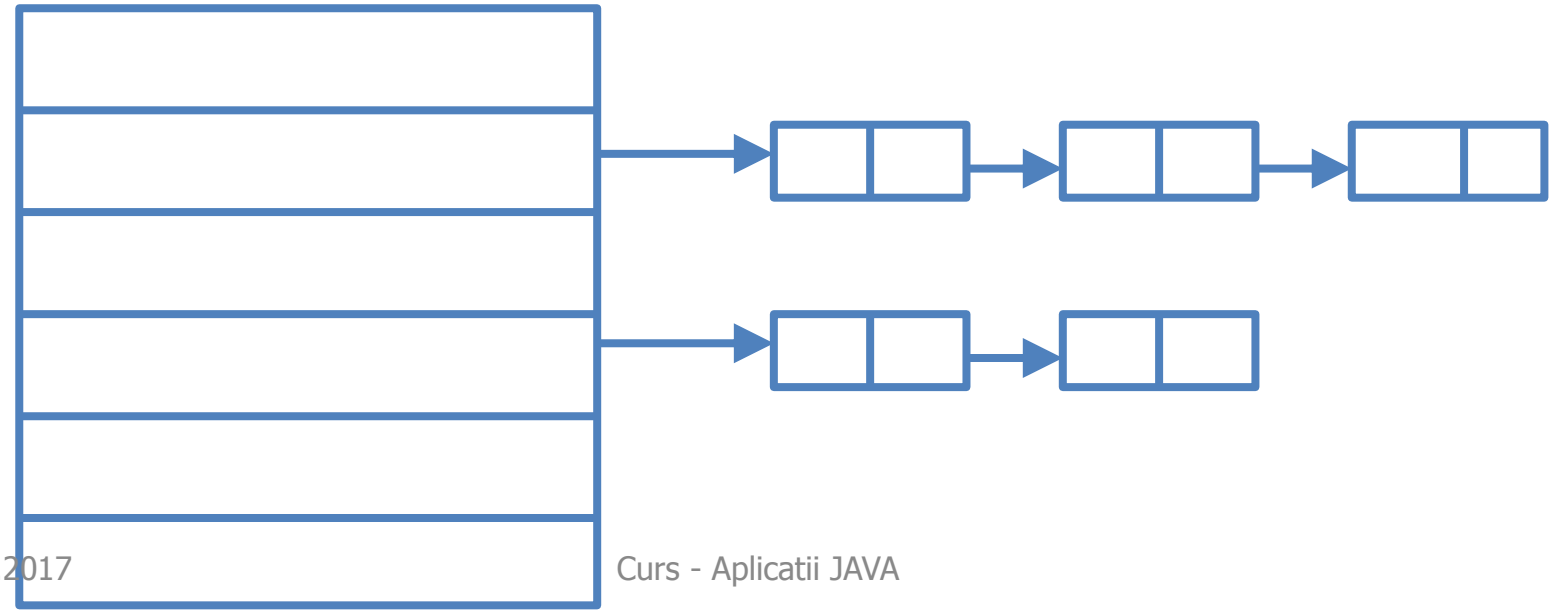


Tehnologii de memorare a datelor

4. Tabele de hashing (hash table)

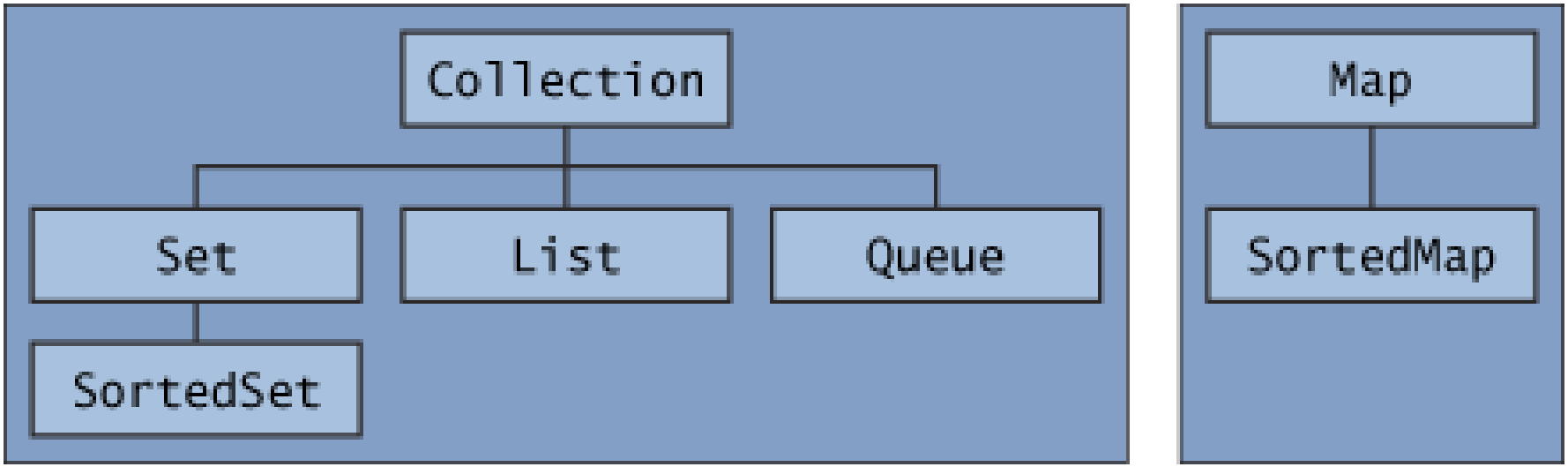
- Fiecare element este format dintr-o pereche cheie-valoare, cu cheie unica
- Dimensiunea este variabila

Avantaj: Este performant la toate operatiile



Elemente introductive

- ✓ *Arhitectura colectiilor Java* consta din mai multe interfete, clase abstracte si clase instantiabile în **pachetul java.util**, prin care se diferentiaza doua categorii de containere, în functie de numarul de valori pe care îl contine fiecare element al containerului:



6. Colecții

1. Elemente introductive
2. Colecții. Avantaje ale colectiilor
3. Tipuri de colecții
4. Interfețe ce descriu colecții
5. Interfața Collection
    1. Interfața Set
    2. Interfața List
    3. Interfața Map
6. Clasa ArrayList
7. Iteratori

Colectii

O colectie este un obiect care grupeaza mai multe elemente intr-o singura unitate.

Prin intermediul colectiilor vom avea acces la diferite tipuri de date cum ar fi:

- vectori
- liste inlantuite
- stive
- multimi matematice
- tabele de dispersie, etc.

Colectiile sunt folosite atat pentru memorarea si manipularea datelor, cat si pentru transmiterea unor informatii de la o metoda la alta.

Arhitectura colecțiilor

- Interfață
- Clasă abstractă
- Implementări concrete

List

AbstractList

ArrayList
LinkedList
Vector

Avantaje ale utilizarii colectiilor

- **Reducerea efortului de programare**: *prin punerea la dispozitia programatorului a unui set de tipuri de date si algoritmi ce modeleaza structuri si operatii des folosite in aplicatii*
- **Cresterea vitezei si calitatii programului**: *implementarile efective ale colectiilor sunt de inalta performanta si folosesc algoritmi cu timp de lucru optim*

- Astfel, la scrierea unei aplicatii putem sa ne concentram eforturile asupra problemei in sine si nu asupra modului de reprezentare si manipulare a informatiilor.

6. Colecții

1. Elemente introductive
2. Colecții. Avantaje ale colectiilor
3. Tipuri de colecții
4. Interfețe ce descriu colecții
5. Interfața Collection
    1. Interfața Set
    2. Interfața List
    3. Interfața Map
6. Clasa ArrayList
7. Iteratori

Tipuri de colectii

Tipul **Collection** defineste câte o valoare în fiecare element (denumirea este putin derutanta, dat fiind ca întreaga colectie de clase se numeste colectii).
Aceast tip cuprinde mai multe subtipuri (prin interfete derivate), în functie de restrictiile impuse.

1. Tipul **List** (List este o interfata care extinde interfata Collection) *defineste modelul de date lista*, adica un container care contine o secventa de elemente aflate într-o anumita ordine;
2. Tipul **Set** (Set este o interfata care extinde Collection) *defineste modelul de date multime*, adica un container în care nu exista elemente duplicat.

Tipuri de colectii

- Tipul Map defineste *modelul de date vector asociativ*, adica un container în care fiecare element contine doua parti asociate (cheie, valoare).

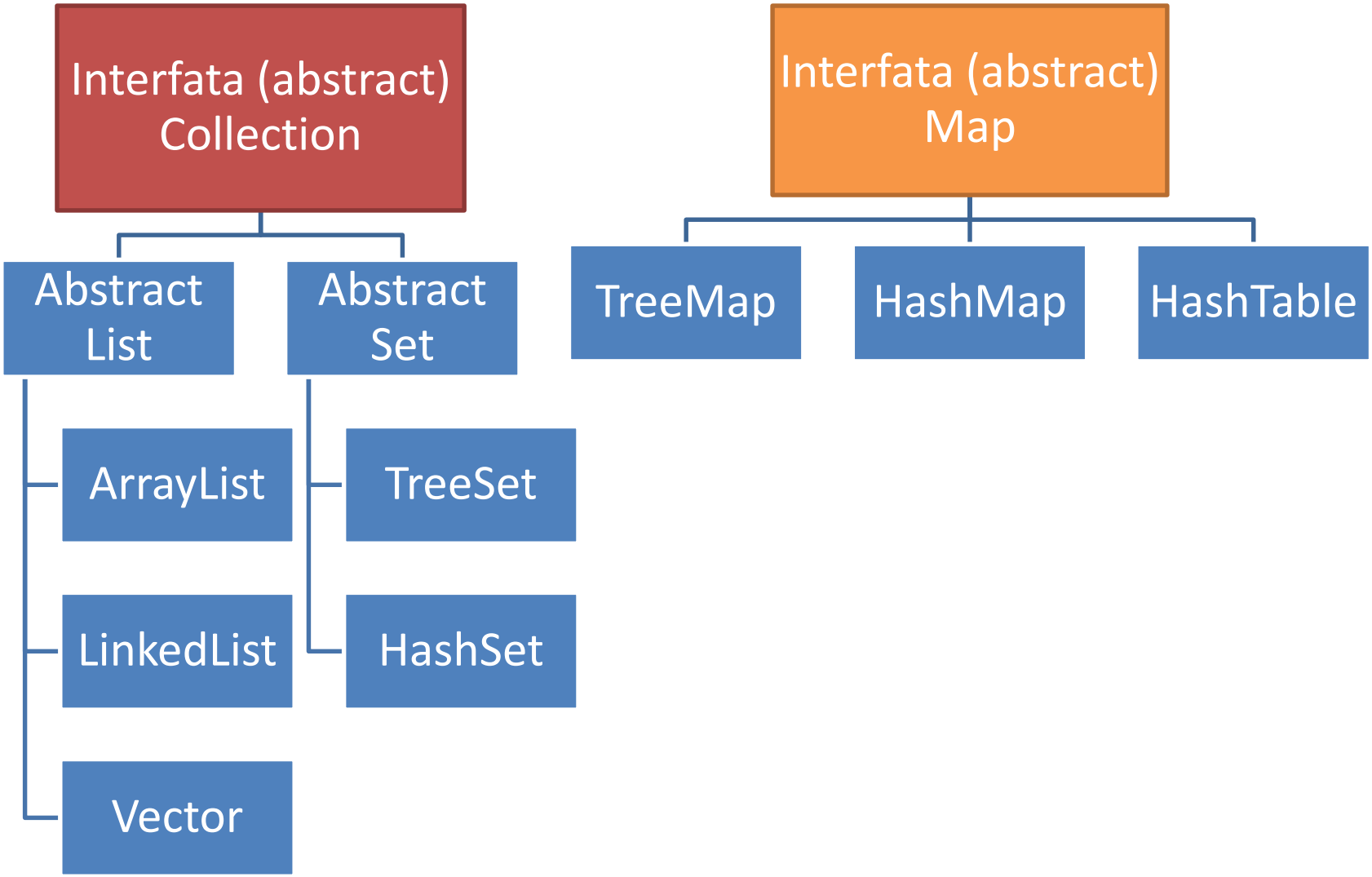
- *Forma unui container* (modelul de date) este impusa prin *interfata* (List, Set, Map) pe care o implementeaza clasa containerului, iar *structura de date a containerului este definita în clasa*, obtinându-se astfel mai multe clase de colectii care difera atât prin modul de organizare cât si prin structura de date folosita.

Tipuri de colectii

- Denumirile claselor de colectii sunt formate din doua parti, *prima parte reprezinta structura de date folosita*, iar cea de *a doua parte reprezinta forma colectiei (interfata implementata).*

- De exemplu:
  - clasa ArrayList - este o lista reprezentata printr-un tablou de elemente
  - si clasa LinkedList - este o lista reprezentata printr-o lista înlantuita de elemente, implementeaza interfata **List**.

Tipuri de colectii



6. Colecții

1. Elemente introductive
2. Colecții. Avantaje ale colectiilor
3. Tipuri de colecții
4. Interfețe ce descriu colecții
5. Interfața Collection
    1. Interfața Set
    2. Interfața List
    3. Interfața Map
6. Clasa ArrayList
7. Iteratori

Arhitectura unei colecții

- Tipul de date al elementelor dintr-o colectie este **Object**, ceea ce inseamna ca multimile reprezentate sunt eterogene, putand include obiecte de orice tip.
- Incepand cu versiunea 1.2, in Java colectiile sunt tratate intr-o maniera unitara, fiind organizate intr-o arhitectura foarte eficienta si flexibila ce cuprinde:

1. *Interfete*
2. *Implementari*
3. *Algoritmi*

Arhitectura unei colecții

1. **Interfete**: tipuri abstracte de date ce descriu colectiile si permit utilizarea lor independent de detaliile implementarilor.
2. **Implementari**: implementari concrete ale interfetelor ce descriu colectii. Aceste clase reprezinta tipuri de date reutilizabile.
3. **Algoritmi**: metode care efectueaza diverse operatii utile cum ar fi cautarea sau sortarea, definite pentru obiecte ce implementeaza interfetele ce descriu colectii. Acesti algoritmi se numesc si polimorfici deoarece *pot fi folositi pe implementari diferite ale unei colectii*, reprezentand elementul de functionalitate reutilizabila.

Interfete ce descriu colectii

- Interfetele reprezinta nucleul mecanismului de lucru cu colectii, scopul lor fiind de a permite utilizarea structurilor de date independent de modul lor de implementare.

- **Interfata Collection** modeleaza o colectie la nivelul cel mai general, descriind un grup de obiecte numite si *elementele sale*.

Interfete ce descriu colectii

- Unele implementari ale acestei interfete permit existenta elementelor duplicate, alte implementari nu.
- Unele au elementele ordonate, altele nu.
- Platforma Java nu ofera nici o implementare directa a a interfetei **Collection**, ci exista doar implementari ale unor subinterfete mai concrete, cum ar fi **Set** sau **List**.

6. Colecții

1. Elemente introductive
2. Colecții. Avantaje ale colectiilor
3. Tipuri de colecții
4. Interfețe ce descriu colecții
5. Interfața Collection
    1. Interfața Set
    2. Interfața List
    3. Interfața Map
6. Clasa ArrayList
7. Iteratori

Interfete ce descriu colectii - Interfata Collection

- Interfata **Collection** contine mai multe declaratii de metode care sunt definite în toate clasele care implementeaza interfetele List si Set (derivate din interfata Collection).

- Dintre acestea:
  - metodele add(Object o) si remove(Object o) permit adaugarea unui nou element, respectiv eliminarea unui element existent în colectie
  - metoda int size() returneaza numarul de elemente al colectiei

Interfete ce descriu colectii - Interfata Collection

```
public interface Collection {
// Metode cu caracter general
        int size();
        boolean isEmpty();
        void clear();
        Iterator iterator();
// Operatii la nivel de element
        boolean contains(Object element);
        boolean add(Object element);
        boolean remove(Object element);
```

Interfete ce descriu colectii - Interfata Collection

```
// Operatii la nivel de multime
        boolean containsAll(Collection c);
        boolean addAll(Collection c);
        boolean removeAll(Collection c);
        boolean retainAll(Collection c);
// Metode de conversie in vector
        Object[] toArray();
        Object[] toArray(Object a[]);
}
```

Interfete ce descriu colectii - Interfata Set

- **Set** modeleaza notiunea de multime in sens matematic.
- O multime nu poate avea elemente duplicate, mai bine zis nu poate contine doua obiecte o1 si o2 cu proprietatea o1.equals(o2).
- Mosteneste metodele din **Collection**, fara a avea alte metode specifice.
- Doua dintre clasele standard care ofera implementari concrete ale acestei interfete sunt **HashSet** si **TreeSet**.

Interfete ce descriu colectii - Interfata SortedSet

- **SortedSet** este asemanatoare cu interfata Set, diferenta principala constand in faptul ca *elementele dintr-o astfel de colectie sunt ordonate ascendent*.
- Pune la dispozitie operatii care beneficiaza de avantajul ordonarii elementelor.
- Ordonarea elementelor se face conform ordinii lor naturale, sau conform cu ordinea data de un comparator specificat la crearea colectiei si este mentinuta automat la orice operatie efectuata asupra multimii.

Interfete ce descriu colectii - Interfata SortedSet

- Singura conditie este ca, pentru orice doua obiecte o1, o2 ale colectiei, apelul o1.compareT o(o2) (sau comparator.compare(o1, o2), daca este folosit un comparator) trebuie sa fie valid si sa nu provoace exceptii.

- Fiind subclasa a interfetei Set, mosteneste metodele acesteia, oferind metode suplimentare ce tin cont de faptul ca multimea este sortata:

Interfete ce descriu colectii - Interfata SortedSet

```
public interface SortedSet extends Set {
// Subliste
SortedSet subSet(Object fromElement, Object toElement);
SortedSet headSet(Object toElement);
SortedSet tailSet(Object fromElement);
// Capete
Object first();
Object last();
Comparator comparator();
}
```

Clasa care implementeaza aceasta interfata este TreeSet.

Interfete ce descriu colectii – interfata List

- **List** descrie liste (secvente) de elemente indexate.
- Listele pot contine duplicate si permit un control precis asupra pozitiei unui element prin intermediul indexului acelui element.
- In plus, fata de metodele definite de interfata Collection, avem metode pentru:
  - acces pozitional
  - cautare
  - iterare avansata

Interfete ce descriu colectii – interfata List

Definitia interfetei este:

```
public interface List extends Collection {
// Acces pozitional
        Object get(int index);
        Object set(int index, Object element);
        void add(int index, Object element);
        Object remove(int index);
        abstract boolean addAll(int index, Collection c);
```

Interfete ce descriu colectii – interfata List

```
// Cautare
        int indexOf(Object o);
        int lastIndexOf(Object o);
// Iterare
        ListIterator listIterator();
        ListIterator listIterator(int index);
// Extragere sublista
        List subList(int from, int to);
}
```

Clase standard care implementeaza aceasta interfata sunt:
ArrayList, LinkedList, Vector.

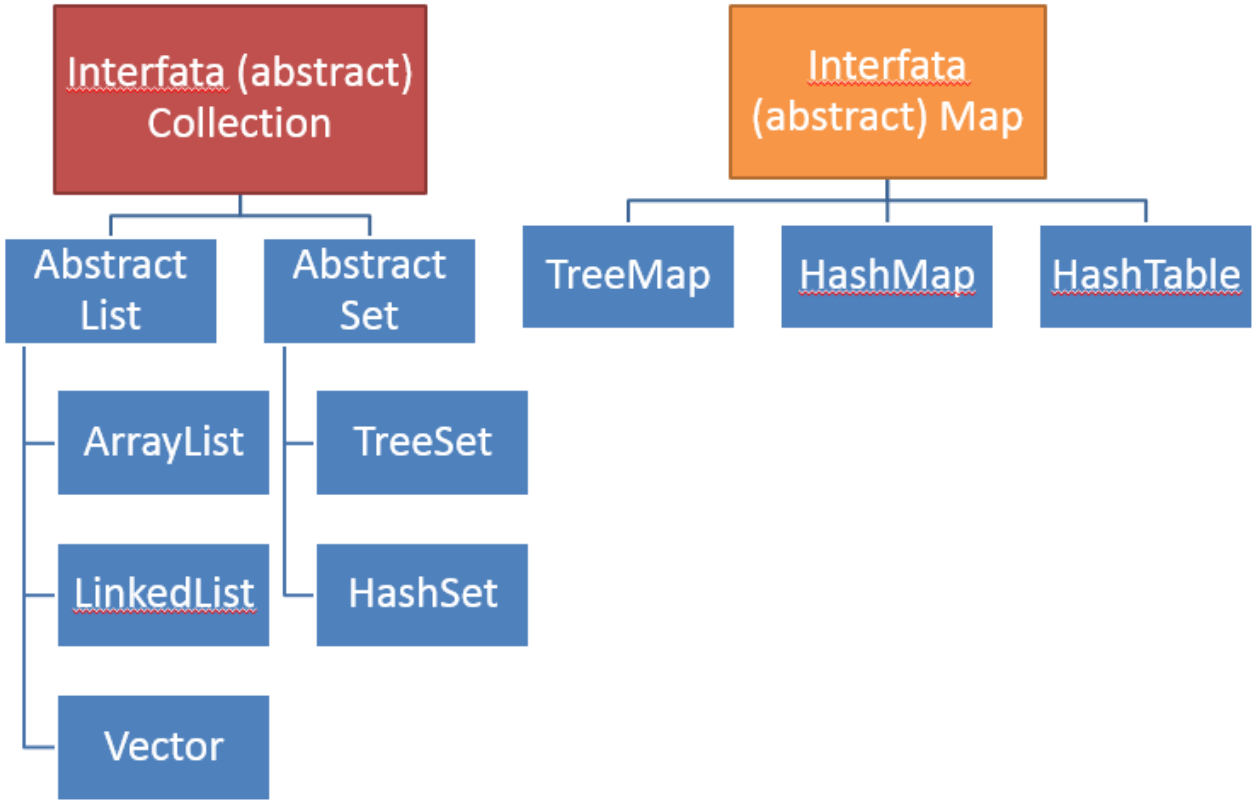
Interfete ce descriu colectii – interfata Map

- **Map** descrie structuri de date ce asociaza fiecarui element o cheie unica, dupa care poate fi regasit.

- *Obiectele de acest tip nu pot contine chei duplicate si fiecare cheie este asociata la un singur element.*

Interfete ce descriu colectii – interfata Map

- Ierarhia interfetelor derivate din Map este independenta de ierarhia derivata din **Collection**.



Interfete ce descriu colectii – interfata Map

Definitia interfetei este:

```
public interface Map {
// Metode cu caracter general
        int size();
        boolean isEmpty();
        void clear();
// Operatii la nivel de element
        Object put(Object key, Object value);
        Object get(Object key);
        Object remove(Object key);
        boolean containsKey(Object key);
        boolean containsValue(Object value);
```

Interfete ce descriu colectii - interfata Map

Definitia interfetei (continuare):

```
// Operatii la nivel de multime
        void putAll(Map t);
// Vizualizari ale colectiei
        public Set keySet();
        public Collection values();
        public Set entrySet();
// Interfata pentru manipularea unei inregistrari
        public interface Entry {
        Object getKey();
        Object getValue();
        Object setValue(Object value);
        }
}
```

Clase care implementeaza interfata Map sunt HashMap, TreeMap si Hashtable.

Interfete ce descriu colectii - interfata SortedMap

- SortedMap este asemanatoare cu interfata Map, la care se adauga faptul ca multimea cheilor dintr-o astfel de colectie este mentinuta ordonata ascendent conform ordinii naturale, sau conform cu ordinea data de un comparatorul specificat la crearea colectiei.
- Este subclasa a interfetei Map, oferind metode suplimentare pentru: *extragere de subtabele*, *aflarea primei/ultimei chei*, *aflarea comparatorului folosit pentru ordonare*.

Interfete ce descriu colectii - interfata SortedMap

Definitia interfetei este:

```
public interface SortedMap extends Map {

// Extragerea de subtabele
        SortedMap subMap(Object fromKey, Object
toKey);
        SortedMap headMap(Object toKey);
        SortedMap tailMap(Object fromKey);
```

Interfete ce descriu colectii - interfata SortedMap

Definitia interfetei (continuare):

```
// Capete
        Object first();
        Object last();
// Comparatorul folosit pentru ordonare
        Comparator comparator();
}
```

Clasa care implementeaza aceasta interfata este TreeMap.

6. Colecții

1. Elemente introductive
2. Colecții. Avantaje ale colectiilor
3. Tipuri de colecții
4. Interfețe ce descriu colecții
5. Interfața Collection
    1. Interfața Set
    2. Interfața List
    3. Interfața Map
6. Clasa ArrayList
7. Iteratori

Clasa ArrayList

- Clasa ArrayList implementeaza interfata List si, indirect, interfata **Collection**, definind toate metodele prevazute în aceste interfete.

- Metodele de inserare permit:

  - adaugarea unui element la sfârsitul listei (add (Object o))

  - sau într-o pozitie dorita (add(int index, Object o)).

Clasa ArrayList

- Metodele de stergere permit:
  - eliminarea unui element de la o pozitie data (remove(int index)),
  - sau eliminarea tuturor elementelor dintr-un interval dat de pozitii (remove(int fromIndex, int toIndex)).

- Elementele se pot citi (fara a fi eliminate din lista) cu metodele get() si get(int index).

Clasa ArrayList

- Exemplul urmator (Colectie.java) evidentiaza modul de utilizare a unei colectii de tipul ArrayList, precum si problemele care pot sa apara în cazul conversiilor incorecte.

Clasa ArrayList

```
// Colectie.java - Exemplu de container ArrayList
import java.util.*;
class Masina {
   private int nr;
   Masina(int i) { nr = i; }
   void print() { System.out.println("Masina #" + nr); }
}
class Avion {
    private int nr;
    Avion(int i) { nr = i; }
    void print() { System.out.println("Avion #" + nr); }
}
```

Clasa ArrayList

```
public class Colectie {
public static void main(String[] args) {
    ArrayList masini = new ArrayList();
    for(int i = 0; i < 7; i++)
          masini.add(new Masina(i));
    // Se adauga avioane la masini!!!!!
    masini.add(new Avion(7));
    for(int i = 0; i < masini.size(); i++)
          ((Masina)masini.get(i)).print();
 // Eroarea este detectata in timpul executiei
     }
}
```

Clasa ArrayList

- Colectia este creata ca o instanta a clasei ArrayList.
- În aceasta colectie se intoduc mai multe referinte de tip Masina si mai multe referinte de tip Avion folosind metoda add.
- Dupa citirea elementelor din colectie (cu functia get()), trebuie sa fie facuta conversia referintei din referinta la tipul **Object** (asa cum a fost memorata) la tipul obiectului.
- Daca aceasta conversie se face incorect (ca în exemplul de mai sus, când referinta la un obiect de clasa Avion este convertita în referinta de clasa Masina), apare o exceptie in cursul executiei.

6. Colecții

1. Elemente introductive
2. Colecții. Avantaje ale colectiilor
3. Tipuri de colecții
4. Interfețe ce descriu colecții
5. Interfața Collection
    1. Interfața Set
    2. Interfața List
    3. Interfața Map
6. Clasa ArrayList
7. Iteratori

Iteratori

- *Un iterator este un obiect dintr-o clasa care implementeaza interfata **Iterator** si care permite parcurgerea elementelor unei colectii.*

- Interfata **Iterator** prevede trei metode care se pot folosi pentru parcurgerea colectiilor:

1. hasNext()
2. next()
3. remove()

Iteratori

1. boolean hasNext() - returneaza valoarea true daca mai exista elemente de parcurs.

2. Object next() - returneaza referinta la urmatorul element din colectie.

3. void remove() - sterge din colectie ultimul element returnat de iterator.

Iteratori

- Un iterator se creeaza cu ajutorul metodei Iterator iterator() apelata pentru un obiect container.
- Acesta metoda este declarata în interfata **Collection** si implementata în fiecare clasa de colectii.
- Parcurgerea unei colectii folosind un iterator în locul functiei get()de citire a elementelor este, în general mai usoara, deoarece nu mai este necesar sa se compare numarul de elemente extrase cu numarul de elemente disponibile în container (aceasta operatie o face metoda hasNext() a iteratorului).
- În plus, interfata Iterator este aceeasi indiferent de tipul containerului, si deci metodele folosite pentru parcurgerea elementelor colectiei ramân aceleasi chiar daca se schimba tipul containerului.

Iteratori

În exemplul urmator (Iteratori.java) se foloseste un iterator pentru parcurgerea si afisarea elementelor unei colectii ArrayList.

```
// Iteratori.java
import java.util.*;
class Numar{
    int n;
    public Numar(int k){ n = k; }
    public String toString(){ return "n = " + n; }
}
```

Iteratori

```
public class Iteratori {
    public static void main(String[] args) {
    ArrayList numere = new ArrayList();
    for(int i = 0; i < 7; i++)
          numere.add(new Numar(i));
    // Se listeaza elementele folosind un iterator
    Iterator it = numere.iterator();
    while (it.hasNext()){
        Numar nr = (Numar)it.next();
        System.out.println(nr);
   }
}
}
```

Iteratori – exemplul 2

Intr-un vector adaugam numerele de la 1 la 10, apoi le amestecam, dupa care le parcurgem element cu element folosind un iterator, inlocuind numerele pare cu 0.

```
import java . util .*;
class TestIterator {
        public static void main ( String args []) {
        ArrayList a = new ArrayList ();
// Adaugam numerele de la 1 la 10
        for (int i=1; i <=10; i++)
                a.add(new Integer (i));
```

Iteratori – exemplul 2

```
// Amestecam elementele colectiei
        Collections.shuffle (a);
        System.out.println (" Vectorul amestecat : " + a);
// Parcurgem vectorul
        for ( ListIterator it=a.listIterator (); it.hasNext (); ) {
        Integer x = ( Integer ) it.next ();
// Daca elementul curent este numar par, atunci devine 0
        if (x.intValue () % 2 == 0)
                it.set( new Integer (0));
        }
        System.out.print (" Rezultat : " + a);
        }
}
```

Referinte

- Curs practic de Java, Cristian Frasinaru – capitolul Colectii
- http://docs.oracle.com/javase/7/docs/api/java/util/Collection.html
- http://www.tutorialspoint.com/java/java_collections.htm

Întrebări?