Ingineria Sistemelor de Programare

Colectii de obiecte

mihai.hulea@aut.utcluj.ro

2017

Introducere

- Stocarea grupurilor de obiecte utilizand array

```
Instrument[] lista = new Instrument[2];
lista[0] = new Instrument("aa");
lista[1] = new Instrument("bb");

for(int i=0;i<lista.length;i++)
   lista[i].canta();
```

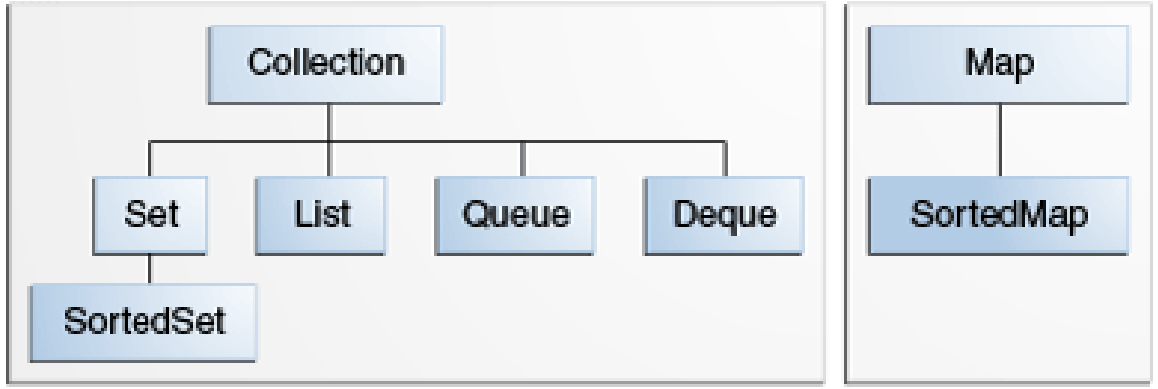
- Dezavantaje ?
  - Dimensiune fixa (nu poate fi modificata ulterior)
  - Trebuie cunoscut in avans numarul de obiecte pe care dorim sa le stocam

Colectiile Java

- Collections Framework – set de clase si interfete ce ofera programatorului uneltele necesare pentru manipularea grupurilor de obiecte.
- Fac parte din pachetul *java.util*
- Collections Framework este compus din:
  - *Interfete*
  - *Implementari*
  - *Algoritmi*
- Avantaje:
  - Reduc efortul de programare
  - Ofera mecanisme de extindere
  - Ofera algoritmi specifici pentru manipularea colectiilor de obiecte

Interfetele

- Set - colectie de elemente unice
- List - colectie de elemente ordonate
- Queue si Deque- colectii de elemente cu metode suplimentare pentru manipulare (inserare, stergere, parcurgere)
- Map - colectii de tip cheie – valoare
- SortedSet - colectie de tip Set cu elemente sortate
- SortedMap - colectie de tip Map cu elemente sortate



Interfata Collection

- Principalele metode:
  - boolean add(E o)
  - boolean contains(Object o)
  - boolean remove(Object o)
  - boolean isEmpty()
  - int size()
  - Object[] toArray()
  - Iterator<E> iterator()

Interfata Set

- interface Set<E> implements Collection, Iterable
- Lista neordonata de elemente
- Metode relevante:
  - boolean contains(Object o) // membership test
  - boolean containsAll(Collection<?> c) //subset test
  - boolean addAll(Collection<? extends E> c) // union
  - boolean retainAll(Collection<?> c) // intersection
  - boolean removeAll(Collection<?> c) // difference
- addAll, retainAll, removeAll returneaza true daca setul a fost modificat sau false in caz contrar
- Obiectele din Set trebuie sa suprascrie metodele equals si hashCode

Interfata SortedSet

- Un Set in care elementele sunt stocate sortat
- interface SortedSet<E>
  implements Set, Collection, Iterable
- Metode relevante:
  - E first()
  - E last()
- Obiectele din SortedSet trebuie sa implementeze interfata Comparable

Interfata List

- interface List<E> extends Collection, Iterable
- Lista ordonata de elemente
- Metode relevante:
  - void add(int index, E element)
  - E remove(int index)
  - boolean remove(Object o)
  - E set(int index, E element)
  - E get(int index)
  - int indexOf(Object o)
  - int lastIndexOf(Object o)
  - ListIterator<E> listIterator()

Interfata Map

- Interface Map<K,V>
- O structura de date de tip cheie – valoare
- Metode relevante:
  - V put(K key, V value) // adds a key-value pair to the map
  - V get(Object key) // given a key, looks up the associated value
  - Set<K> keySet()
  - Collection<V> values()

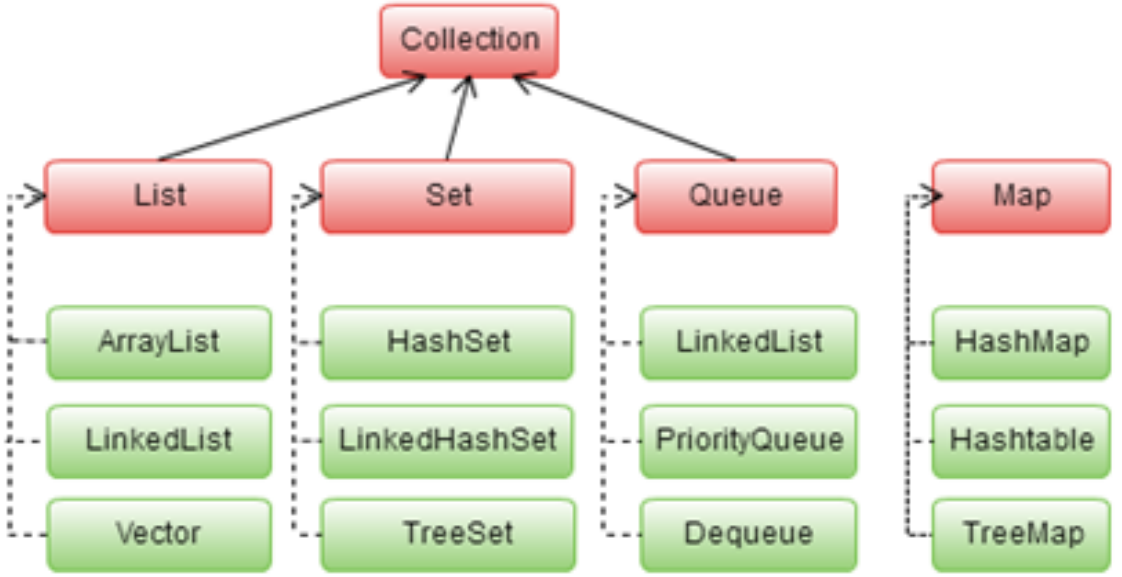
Interfata SortedMap

- Interface SortedMap<K,V> extentds M<K,V>
- Cheile sunt stocate sortate
- Metode relevante:
  - K firstKey()
  - K lastKey()
- Doar obiectele ce implementeaza interfata Comparable pot fi utilizate ca si chei

Implementarea colectiilor

Implementarea colectiilor

- class HashSet<E> implements Set
- class TreeSet<E> implements SortedSet
- class ArrayList<E> implements List
- class LinkedList<E> implements List
- class Vector<E> implements List
  - class Stack<E> extends Vector
    - push, pop, peek, isEmpty
- class HashMap<K, V> implements Map
- class TreeMap<K, V> implements SortedMap



http://fresh2refresh.com/java-tutorial/java-collections-framework/

Tipuri generice in colectii

Colectiile stocheaza obiecte eterogene. Prin adaugarea unui obiect in colectie se pierde informatia cu privire la tipul concret al obiectului.

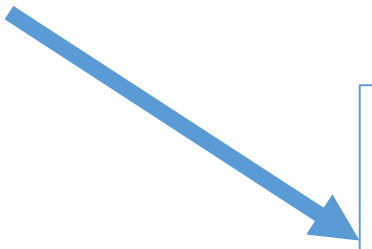
```
public class GenericExample {
   public static void main(String[] args) {
      ArrayList l1 = new ArrayList();
      l1.add(new BankAccount("123"));
      l1.add(new BankAccount("342"));
      l1.add(new Order("Client1","342"));

      Object o = l1.get(0);
      BankAccount o1 = (BankAccount)l1.get(0);
      Order o2 = (Order)l1.get(0);
      o1.displayAccount();
      o2.displayOrder();

   }
}
```

```
class BankAccount{
   //...
}

class Order {
            //...
}
```



```
Exception in thread "main" java.lang.ClassCastException:
colectii.liste.BankAccount cannot be cast to colectii.liste.Order
            at
colectii.liste.GenericExample.main(GenericExample.java:23)
Java Result: 1
```

Tipuri generice in colectii

- Eng. Generic Types
- Mecanism de parametrizare a claselor si metodelor
- Sintaxa pentru definirea unei clase generice:
  - *class name<T1, T2, ..., Tn> { /* ... */ }*
- Sectiunea tipurilor parametrice este delimitata de ‘<‘ si ‘>’
- Utilizarea unei clase generice:
  - Node<Integer> i = new Node<>();

https://docs.oracle.com/javase/tutorial/java/generics/types.html

```
class Node<T>{
    private T value;

    public T getValue() {
        return value;
    }

    public void setValue(T value) {
        this.value = value;
    }
}
```

Tipuri generice in colectii

- Utilizarea tipurilor generice asigura validarea la compilare a tipurilor obiectelor
- Tipuri generice = tipuri cu care colectiilor vor opera

```
ArrayList<BankAccount> l1 = new ArrayList<>();
l1.add(new BankAccount("123"));
l1.add(new BankAccount("342"));
l1.add(new Order("Client1","342"));
```

Eroare de compilare !

```
BankAccount o1 = l1.get(0);
Order o2 = (Order)l1.get(0);
o1.displayAccount();
o2.displayOrder();
```

Exemple

- Exemple List (ArrayList, LinkedList)
- Exemple Set (HashSet)
- Exemple Map (HashMap)

Clasa Collections

- Clasa utilitare ce ofera algoritmi de manipulare a listelor
- Algoritmi disponibil in clasa Collections:
  - Sort
  - Shuffle
  - Reverse
  - Swap
  - ReplaceAll
  - Fill
  - Copy
  - BinarySearch
  - IndexOfSublist
  - LastIndexOfSubList

Clasa Collections - Exemplu

```
ArrayList<Order> a = new ArrayList<>();
a.add(new Order("c1","1"));
a.add(new Order("c2","2"));
a.add(new Order("c3","3"));
a.add(new Order("c4","4"));

System.out.println(a);
Collections.shuffle(a);
System.out.println(a);
```

Ordonarea obiectelor

Interfata Comparable

- Nr negativ daca T < obiectul current
- 0 daca T = obiectul current
- Nr pozitiv daca T > obiectul curent

```
ArrayList<Product> list = new ArrayList<>();
list.add(new Product("A", 7));
list.add(new Product("B", 5));
list.add(new Product("C", 4));

Collections.sort(list);

for(Product p: list)
   System.out.println(p);
```

```
public interface Comparable<T> {
   public int compareTo(T o);
}
```

```
class Product implements Comparable<Product>{
   private String name;
   private int price;

            ...

   @Override
   public int compareTo(Product o) {
      return this.price - o.price;
   }

}
```

Interfata Comparator

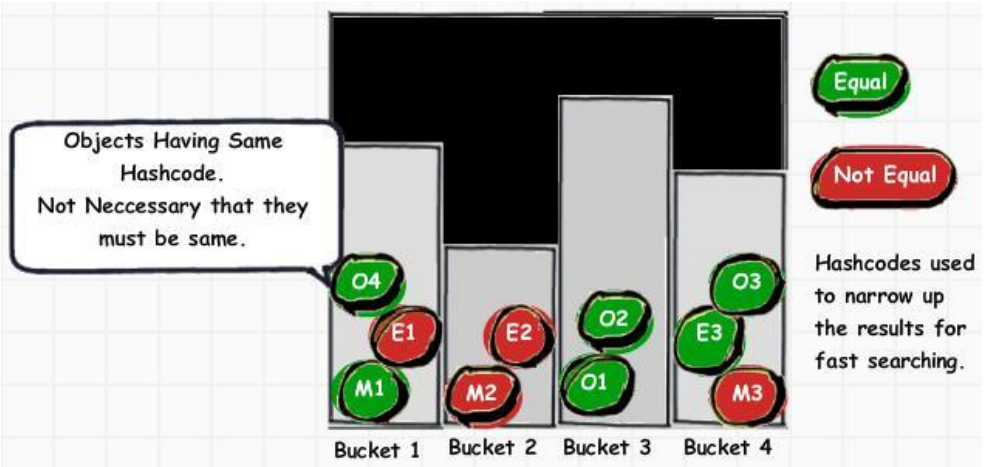
- Primeste ca argumente ambele obiecte
- Nr negativ daca T < obiectul current
- 0 daca T = obiectul current
- Nr pozitiv daca T > obiectul curent

```
public interface Comparator<T> {
   int compare(T o1, T o2);
}
```

```
Comparator<Product> NAME_ORDER =
                        new Comparator<Product>() {
   public int compare(Product e1, Product e2) {
      return e1.getName().compareTo(e2.getName());
   }
 };
Collections.sort(list, NAME_ORDER);
list.sort(NAME_ORDER); //incepand cu Java 8
```

Suprascrierea equals() si hashCode()

- Cum lucreaza colectiile hash: http://en.wikipedia.org/wiki/Hashtable
- Daca doua obiecte sunt egale atunci acestea trebuie sa returneze aceiasi valoare pentru metodele queals() si hashcode()
- Daca doua obiecte returneaza acelasi hash code nu este obligatoriu ca ele sa fie egale
- hashCode() folosit in HashTable, HashMap si HashSet



http://www.jitendrazaa.com/blog/java/what-is-the-need-to-override-hashcode-and-equals-method/

Suprascrierea equals() si hashCode()

Extras din documentatia javadocs oficiala. Regulile pentru metoda equals:

- It is reflexive: for any non-null reference value x, x.equals(x) should return true.
- It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.
- It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.
- It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.
- For any non-null reference value x, x.equals(null) should return false.

Suprascrierea equals() si hashCode()

Extras din documentatia javadocs oficiala. Regulile pentru metoda hashCode():

- Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
- If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
- It is not required that if two objects are unequal according to the equals(java.lang.Object) method, then calling the hashCode method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hash tables.

Suprascrierea equals() si hashCode()

- Daca se suprascrie metoda equals() trebuie suprascrisa si metoda hashCode()

```
class Employee {
    protected long   employeeId;
    protected String firstName;
    protected String lastName;

    public boolean equals(Object o){
        if(o == null)                  return false;
        if(!(o instanceof Employee)) return false;

        Employee other = (Employee) o;
        return this.employeeId == other.employeeId;
    }

    public int hashCode(){
        return (int) employeeId;
    }

}
```

```
class Employee {
    protected long employeeId;
    protected String firstName;
    protected String lastName;

    public boolean equals(Object o) {
        if (o == null)
            return false;
        if (!(o instanceof Employee)) return false;
        Employee other = (Employee) o;
        return this.employeeId == other.employeeId
                &&firstName.equals(other.firstName)
                &&lastName.equals(other.lastName);

    }

    public int hashCode() {
        return (int) employeeId;
    }
}
```