

Pemrograman Berorientasi Obyek

Collections API

Oleh Politeknik Elektronika Negeri Surabaya
2020



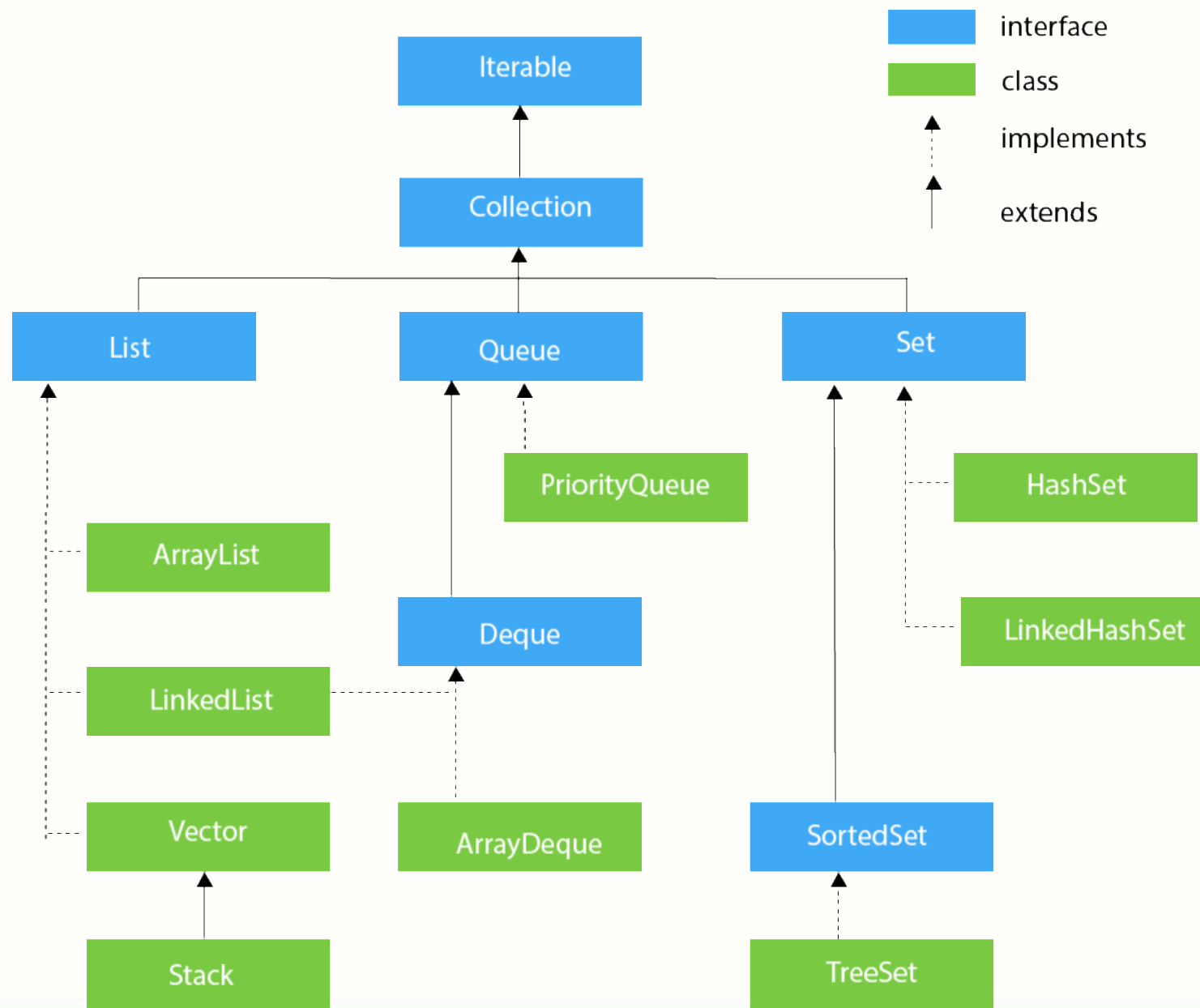
**Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer**

Collection

- Merupakan sebuah framework yang menyediakan arsitektur untuk menyimpan dan memanipulasi sekumpulan objek.
- Java Collection Framework menyediakan banyak interface (Set, List, Queue, Deque) dan class (ArrayList, Vector, LinkedList, HastSet, TreeSet)

Collection vs Arrays

NO	Key	Arrays	Collection
1	Size	Memiliki ukuran (size) yang tetap (fix). Sekali array dideklarasikan dengan ukuran tertentu kita tidak dapat merubah ukuran array tersebut.	Memiliki ukuran yang dinamis
2	Memory Consumption	Array dieksekusi dengan cepat dan memiliki kinerja yang baik, namun menghabiskan lebih banyak memori.	Konsumsi memori lebih rendah namun performa lebih rendah dibandingkan dengan Array
3	Data Type	Hanya dapat menampung data yang sejenis atau memiliki tipe data yang sama. (homogeneous data type)	Dapat menyimpan data yang homogen ataupun heterogen
4	Primitives Storage	Array dapat menampung baik tipe data primitive ataupun object	Hanya dapat menyimpan data bertipe object
5	Performance	Karena impementasi dan penyimpanannya internal, Array memiliki kinerja yang lebih baik.	Memiliki performa yang lebih rendah disbanding array



List Interface

- `List <data-type> list1 = new ArrayList();`
- `List <data-type> list1 = new LinkedList();`
- `List <data-type> list1 = new Vector();`
- `List <data-type> list1 = new Stack();`

Class ArrayList

```
ArrayList<String> list = new ArrayList<String>();
```

```
list.add("Ravi");
```

```
list.add("Vijay");
```

```
list.add("Ajay");
```

```
Iterator iterator = list.iterator();
```

```
While(iterator.hasNext){
```

```
    System.out.println(iterator.next());
```



Class LinkedList

```
LinkedList<String> list = new LinkedList<String>();  
list.add("Ravi");  
list.add("Vijay");  
list.add("Ajay");
```



Class Vector

```
Vector<String> vector = new Vector<String>();  
vector.add("Ravi");  
vector.add("Vijay");  
vector.add("Ajay");
```


Class Stack

```
Stack<String> stack = new Stack<String>();  
stack.push("Ravi");  
stack.push("Vijay");  
stack.push("Ajay");  
stack.push("Push Baru");  
stack.pop();
```



Interface Queue

```
Queue<String> q1 = new PriorityQueue();
```

```
Queue<String> q2 = new ArrayDeque();
```

Class PriorityQueue

```
PriorityQueue<String> queue=new PriorityQueue<String>();

queue.add("Amit Sharma");
queue.add("Vijay Raj");
queue.add("JaiShankar");
queue.add("Raj");
System.out.println("head:"+queue.element());
System.out.println("head:"+queue.peek());
System.out.println("iterating the queue elements:");
Iterator itr=queue.iterator();
while(itr.hasNext()){
    System.out.println(itr.next());
}

queue.remove();
queue.poll();
System.out.println("after removing two elements:");
Iterator<String> itr2=queue.iterator();
while(itr2.hasNext()){
    System.out.println(itr2.next());
}
```

Class ArrayDeque

```
Deque<String> deque = new ArrayDeque<String>();  
deque.add("Gautam");  
deque.add("Karan");  
deque.add("Ajay");  
//Traversing elements  
for (String str : deque) {  
    System.out.println(str);  
}
```



Set Interface

```
Set<data-type> s1 = new HashSet<data-type>();
```

```
Set<data-type> s2 = new LinkedHashSet<data-type>();
```

```
Set<data-type> s3 = new TreeSet<data-type>();
```

Class HashSet

```
HashSet<String> set=new HashSet<String>();  
set.add("Ravi");  
set.add("Vijay");  
set.add("Ravi");  
set.add("Ajay");
```

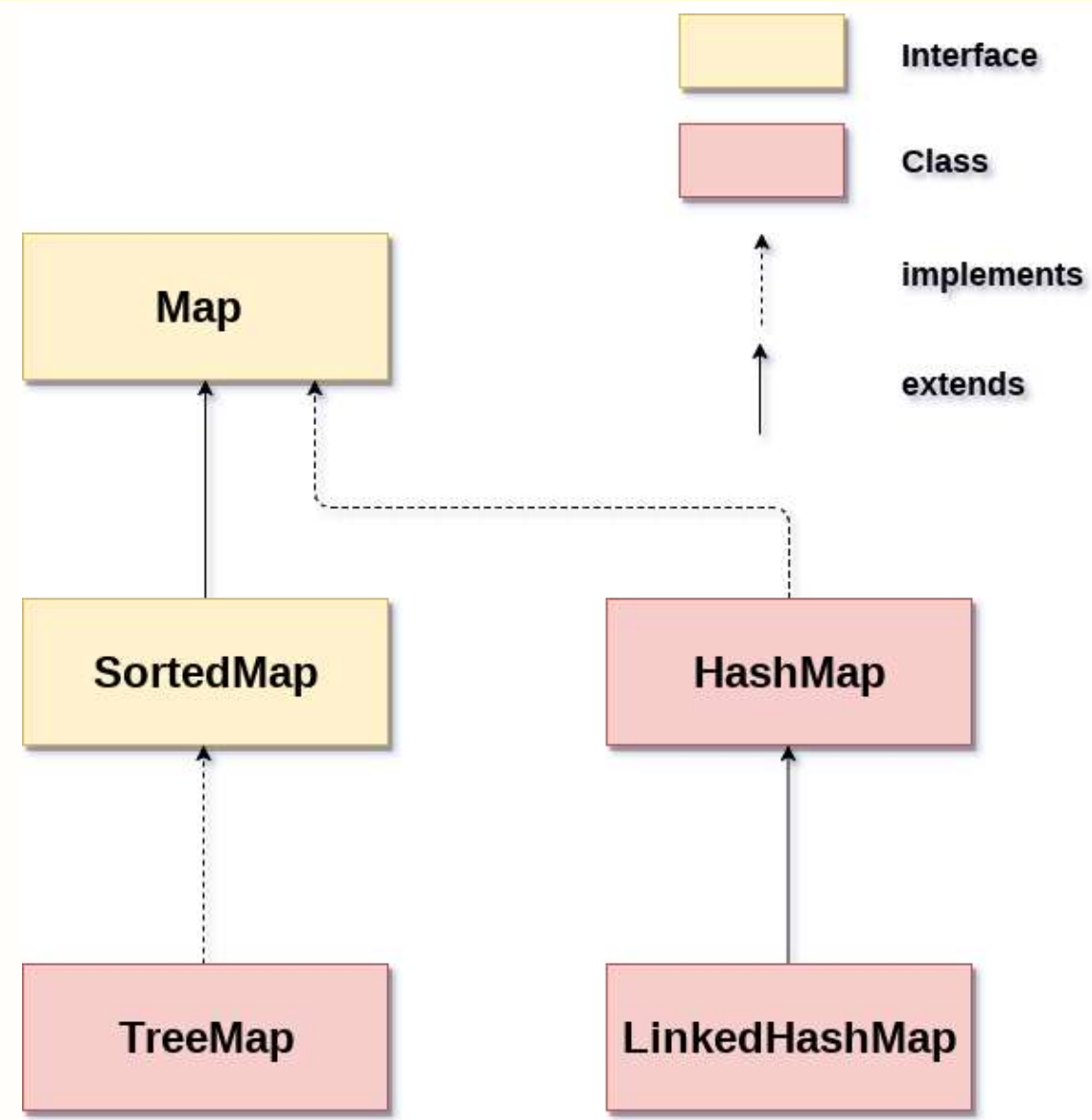
Class HashSet

```
HashSet<String> set=new HashSet<String>();  
set.add("Ravi");  
set.add("Vijay");  
set.add("Ravi");  
set.add("Ajay");
```

Class TreeSet

```
TreeSet<String> set=new TreeSet<String>();  
set.add("Ravi");  
set.add("Vijay");  
set.add("Ravi");  
set.add("Ajay");
```


Map Interface



HashMap

```
Map map=new HashMap();  
//Adding elements to map  
map.put(1,"Amit");  
map.put(5,"Rahul");  
map.put(2,"Jai");  
map.put(6,"Amit");  
//Traversing Map  
Set set=map.entrySet();  
//Converting to Set so that we can traverse
```

```
Iterator itr=set.iterator();  
while(itr.hasNext()){  
    //Converting to Map.Entry so that we can get key and value separately  
  
    Map.Entry entry=(Map.Entry)itr.next();  
    System.out.println(entry.getKey()+" "+entry.getValue());  
}
```



Sorting in Collection

```
ArrayList<String> al=new ArrayList<String>();  
al.add("Virus");  
al.add("Saurav");  
al.add("Mukesh");  
al.add("Tahir");
```

```
Collections.sort(al);  
Iterator itr=al.iterator();  
while(itr.hasNext()){  
    System.out.println(itr.next());
```



Comparable Interface

```
class Student implements Comparable<Student>{  
    int rollno;  
    String name;  
    int age;  
    Student(int rollno,String name,int age){  
        this.rollno=rollno;  
        this.name=name;  
        this.age=age;  
    }  
}
```

```
public int compareTo(Student st){  
    if(age==st.age)  
        return 0;  
    else if(age>st.age)  
        return 1;  
    else  
        return -1;  
    }  
}
```

Comparable Interface

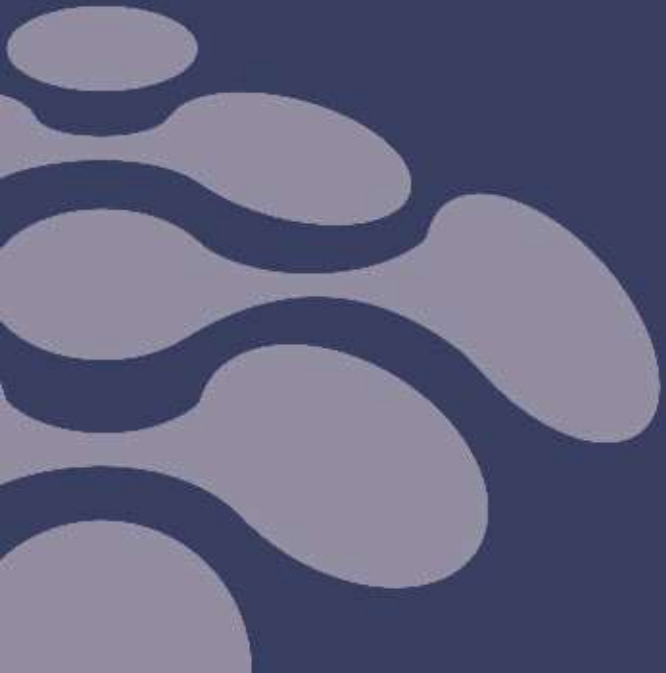
```
public class TestSort1{
    public static void main(String args[]){
        ArrayList<Student> al=new ArrayList<Student>();
        al.add(new Student(101,"Vijay",23));
        al.add(new Student(106,"Ajay",27));
        al.add(new Student(105,"Jai",21));

        Collections.sort(al);
        for(Student st:al){
            System.out.println(st.rollno+" "+st.name+" "+st.age);
        }
    }
}
```



bridge to the future

<http://www.eepis-its.edu>



Pemrograman Berorientasi Obyek

Collections API

Oleh Politeknik Elektronika Negeri Surabaya
2020



**Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer**

Konten

- Collections Framework: Java Collections API
- Interface Collections API:
 - Collection
 - List
 - Set
- Map
- Retrieve elements:
 - Iterator
 - ListIterator
 - Enumeration

Collections Framework

- Dikenalkan pada Java 2 SDK.
- Collection sudah ada sejak JDK 1.0
 - Hashtable
 - Vector

The Java Collections API

- Collection adalah suatu obyek yang bisa digunakan untuk menyimpan sekumpulan obyek
- Obyek yang ada dalam collection ini disebut sebagai **elemen**.
- Collection menyimpan elemen yang bertipe Object, sehingga berbagai tipe obyek bisa disimpan dalam collection.

Note:

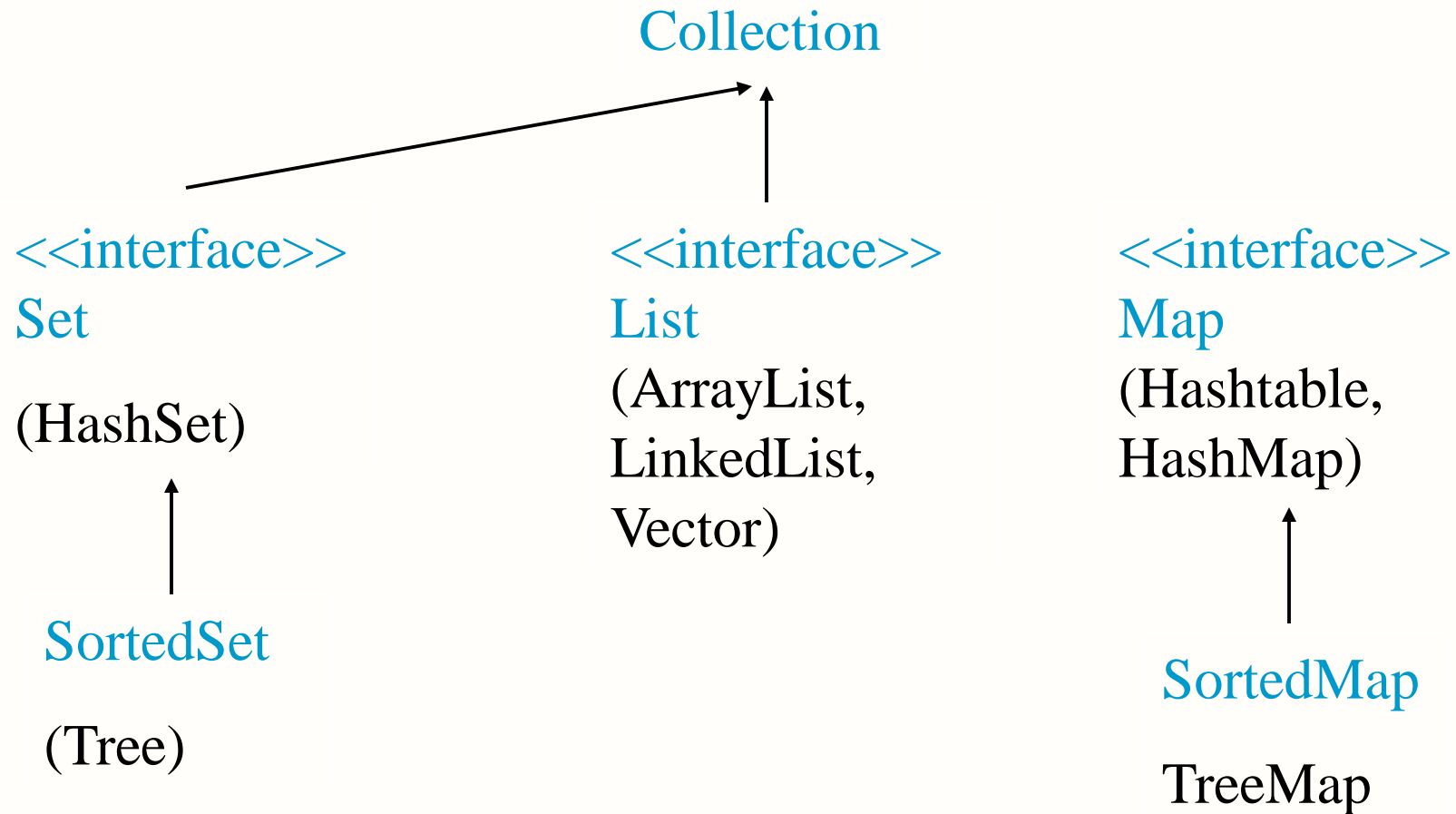
Jangan lupa!! Setelah mengambil obyek dari collection lakukan casting sesuai tipe data obyek yang baru diambil.



The Java Collections API

- Java Collections API terdiri dari interface:
 - **Collection** : sekumpulan obyek yang tidak mempunyai posisi yang tetap (no particular order) dan menerima duplikat.
 - **List**: sekumpulan obyek yangurut (ordered) dan menerima duplikat.
 - **Set**: sekumpulan obyek yang tidakurut (unordered) dan menolak duplikat.
 - **Map**: mendukung pencarian berdasarkan key, key ini harus unik. Has no particular order.

Interface Collection dan Hirarki Class



Method Interface Collection

- boolean add(Object element)

Menambahkan elemen pada collection, bila berhasil akan mengembalikan nilai true.

- boolean remove(Object element)

Menghapus elemen di collection, bila berhasil akan mengembalikan nilai true.

- int size()

Mengembalikan jumlah elemen yang terdapat pada collection.



Method Interface Collection

- boolean isEmpty()
Jika tidak terdapat elemen sama sekali dalam collection maka akan mengembalikan nilai true.
- boolean contains(Object elemen)
Akan mengembalikan nilai true jika elemen terdapat pada collection.
- boolean containsAll(Collection collection_A)
Akan mengembalikan nilai true jika semua elemen yang ada pada collection_A ada pada collection.



Method Interface Collection

- `boolean addAll(Collection collection)`
Akan mengembalikan nilai true jika semua elemen yang ada pada collectionA berhasil ditambahkan pada collection.
- `void clear()`
Menghapus semua elemen collection.
- `void removeAll(Collection collection_A)`
Menghapus semua elemen collection yang ada pada collectionA
- `void retainAll(Collection collection_A)`
Menghapus semua elemen Collection kecuali elemen yang ada pada Collection_A



Interface Collection

```
public interface Collection {  
    // Basic Operations  
    int size();  
    boolean isEmpty();  
    boolean contains(Object element);  
    boolean add(Object element);    // Optional  
    boolean remove(Object element); // Optional  
    Iterator iterator();  
  
    // Bulk Operations  
    boolean containsAll(Collection c);  
    boolean addAll(Collection c);    // Optional  
    boolean removeAll(Collection c); // Optional  
    boolean retainAll(Collection c); // Optional  
    void clear();                    // Optional  
  
    // Array Operations  
    Object[] toArray();  
    Object[] toArray(Object a[]);  
}
```



Set

- Elemen pada Set selalu unik.
- Set menolak duplikat.
- Elemen yang tersimpan tidakurut (unordered) dan unsorted.
- Interface Set merupakan sub interface dari interface Collection.
- Contoh class java yang mengimplementasikan interface Set:
 - HashSet
 - TreeSet.



```
public interface Set {  
    // Basic Operations  
    int size();  
    boolean isEmpty();  
    boolean contains(Object element);  
    boolean add(Object element);    // Optional  
    boolean remove(Object element); // Optional  
    Iterator iterator();  
  
    // Bulk Operations  
    boolean containsAll(Collection c);  
    boolean addAll(Collection c);    // Optional  
    boolean removeAll(Collection c); // Optional  
    boolean retainAll(Collection c); // Optional  
    void clear();                    // Optional  
  
    // Array Operations  
    Object[] toArray();  
    Object[] toArray(Object a[]);  
}
```



Set: HashSet

- Class ini menyimpan anggota dengan cara hashing
- Dibutuhkan waktu yang konstan untuk operasi tambah, hapus dan pengecekan keanggotaan.
- Jika jumlah anggota melebihi kapasitas maka secara otomatis akan menambah daya tampung.

Hashing

- Suatu metode menentukan posisi penyimpanan suatu anggota dalam map atau collection.
- Method hashCode() yang dipunyai oleh class Object dimiliki oleh setiap class yang ada.
- Kode hash ini yang digunakan oleh obyek map atau collection untuk menentukan posisi peletakan suatu anggota.
- Dengan kode hash ini, posisi suatu anggota dapat ditentukan tanpa harus melakukan iterasi satu persatu sehingga waktu pencarian anggota relatif sama.



Operasi Besar (Bulk operations) pada Set: HashSet

Merupakan operasi pada Himpunan

- `s1.containsAll(s2)`
mengembalikan nilai true jika s2 adalah subset s1. (Set s2 adalah subset s1 apabila s1 berisi semua anggota s2)
- `s1.addAll(s2)`
hasil dari s1 adalah gabungan (union) dari s1 dan s2
- `s1.retainAll(s2)`
hasil dari s1 adalah irisan (intersection) dari s1 dan s2.
- `s1.removeAll(s2)`
hasil dari s1 adalah selisih dari s1 dengan s2 ($s1 = s1 - s2$)
Selisih ($s1 - s2$) adalah set yang berisi semua elemen yang ada pada s1 tetapi tidak ada pada s2.



Set: HashSet

```
import java.util.*;

public class SetExample {
    public static void main(String[] args) {
        Set set = new HashSet();
        set.add("one");
        set.add("second");
        set.add("3rd");
        set.add(new Integer(4));
        set.add(new Float(5.0F));
        set.add("second"); // duplicate, not added
        set.add(new Integer(4)); // duplicate, not added
        System.out.println(set);
    }
}
```

Hasil: [one, second, 5.0, 3rd, 4]



Set: HashSet

```
import java.util.*;

public class FindDups {
    public static void main(String args[]) {
        Set s = new HashSet();
        for (int i=0; i<args.length; i++)
            if (!s.add(args[i]))
                System.out.println("Duplicate detected: "+args[i]);

        System.out.println(s.size()+" distinct words detected: "+s);
    }
}

% java FindDups i came i saw i left
```

```
Duplicate detected: i
Duplicate detected: i
4 distinct words detected: [came, left, saw, i]
```



Set: HashSet

```
import java.util.*;

public class FindDups2 {
    public static void main(String args[]) {
        Set uniques = new HashSet();
        Set dups = new HashSet();

        for (int i=0; i<args.length; i++)
            if (!uniques.add(args[i]))
                dups.add(args[i]);

        uniques.removeAll(dups); // Destructive set-difference

        System.out.println("Unique words: " + uniques);
        System.out.println("Duplicate words: " + dups);
    }
}

% java FindDups2 i came i saw i left

Unique words: [came, left, saw]
Duplicate words: [i]
```




```
import java.util.*;

public class Sets{
    public static void main(String args[]){
        Set s1 = new HashSet();
        Set s2 = new HashSet();

        for(int i=0;i<5;i++)
            s1.add(new Integer(i));

        for (int i=3; i<7; i++)
            s2.add(new Integer(i));
        System.out.println("s1 = " + s1);
        System.out.println("s2 = " + s2);

        System.out.println(s1.containsAll(s2));

        s1.retainAll(s2);
        System.out.println(s1);

        s1.removeAll(s2);
        System.out.println(s1);
    }
}
```

```
s1 = [2, 4, 1, 3, 0]
s2 = [4, 6, 3, 5]
false
[4, 3]
[]
```

SortedSet:TreeSet

- Aturan sama dengan interface Set → menolak duplikat.
- Ingat → SortedSet adalah subinterface Set.
- Beda: elemen tersimpan dalam urutan ascending → sorted.
- Contoh SortedSet: TreeSet.

SortedSet: TreeSet

```
import java.util.*;
class SortedSetTest{
    public static void main(String [] arg){
        SortedSet set = new TreeSet();
        set.add("Chess");
        set.add("Whist");
        set.add("Checkers");
        set.add("BlackJack");
        set.add(" Chess");
        System.out.println(set);
    }
}
```

Output: [BlackJack, Checkers, Chess, Whist]



List

- Elemen tersimpan terurut (ordered).
- Urut berdasarkan masukan.
- Menerima duplikat.
- Contoh List:
 - LinkedList : elemen dalam LinkedList masuk dari awal dan dihapus dari akhir.
 - Vector : a growable array of object.
 - ArrayList: mirip vector, bersifat unsynchronized (jika multiple threads mengakses object ArrayList, object ini harus synchronized secara eksternal)
 - Stack



List

- Sebagian besar algoritma(method) pada class Collections diaplikasikan ke List. Sehingga dengan algoritma ini memudahkan untuk memanipulasi data pada List.
- `sort(List)`
mengurutkan List dengan algoritma merge sort
- `shuffle(List)`
Permutasi secara random pada List
- `reverse(List)`
membalik urutan elemen pada List
- `fill(List, Object)`
Mengganti setiap elemen pada List dengan value yang ditentukan
- `copy(List dest, List src)`
Mengkopikan source List ke destination List.
- `binarySearch(List, Object)`
Mencari sebuah element pada List dengan algoritma binary Search

List

```
public interface List extends Collection {  
    // Positional Access  
    Object get(int index);  
    Object set(int index, Object element);           // Optional  
    void add(int index, Object element);           // Optional  
    Object remove(int index);                       // Optional  
    abstract boolean addAll(int index, Collection c); // Optional  
  
    // Search  
    int indexOf(Object o);  
    int lastIndexOf(Object o);  
  
    // Iteration  
    ListIterator listIterator();  
    ListIterator listIterator(int index);  
  
    // Range-view  
    List subList(int from, int to);  
}
```

```

import java.util.*;

public class TestVector{
    private static void swap(List a, int i, int j) {
        Object tmp = a.get(i);
        a.set(i, a.get(j));
        a.set(j, tmp);
    }
    public static void shuffle(List list, Random rnd) {
        for (int i=list.size(); i>1; i--)
            swap(list, i-1, rnd.nextInt(i));
    }
    public static void main(String args[]){
        Vector v = new Vector();

        for(int i=0;i<10;i++)
            v.add(new Integer(i));
        System.out.println(v);
        v.setElementAt("Andi",1);
        System.out.println(v);
        v.set(5,"Rita");
        System.out.println(v);

        swap(v,2,5) ;
        System.out.println(v);

        shuffle(v,new Random());
        System.out.println(v);
    }
}

```

```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, Andi, 2, 3, 4, 5, 6, 7, 8, 9]
[0, Andi, 2, 3, 4, Rita, 6, 7, 8, 9]
[0, Andi, Rita, 3, 4, 2, 6, 7, 8, 9]
[3, 7, 8, 9, Andi, 0, 2, 6, Rita, 4]

```



List : ArrayList

```
1  import java.util.*
2
3  public class ListExample {
4      public static void main(String[] args) {
5          List list = new ArrayList();
6          list.add("one");
7          list.add("second");
8          list.add("3rd");
9          list.add(new Integer(4));
10         list.add(new Float(5.0F));
11         list.add("second");           // duplicate, is added
12         list.add(new Integer(4));     // duplicate, is added
13         System.out.println(list);
14     }
15 }
```

[one, second, 3rd, 4, 5.0, second, 4]



List

```
import java.util.*;

public class TestList{
    public static void main(String args[]){
        List list = new ArrayList();
        list.add("Anis");
        list.add("Budi");
        list.add("Candra");
        list.add("Dewi");
        int i = list.indexOf("Candra");
        System.out.println(i);
    }
}
```

Output

2

List: Vector

```
import java.util.*;
class VectorTest{
    public static void main(String [] arg){
        Vector v = new Vector();
        v.add("Zak");
        v.add("Gordon");
        v.add(0 , "Duke");
        v.add("Lara");
        v.add("Zak");
        System.out.println(v);
        String name = (String) v.get(2);
        System.out.println(name);
    }
}
```

Output: [Duke, Zak, Gordon, Lara, Zak]

Gordon



Stack

- Merupakan class turunan dari Vector
- Dipakai untuk operasi stack
- Method-method yang ada:
 - boolean ← **empty()**
Tests if this stack is empty.
 - Object ← **peek()**
Looks at the object at the top of this stack without removing it from the stack.
 - Object ← **pop()**
Removes the object at the top of this stack and returns that object as the value of this function.
 - Object ← **push(Object item)**
Pushes an item onto the top of this stack.
 - int ← **search(Object o)**
Returns the 1-based position where an object is on this stack.



Contoh Stack

```
import java.util.*;  
  
public class MyStack {  
    public static void main(String args[]) {  
        Stack mystack=new Stack();  
        mystack.push(new Integer(5));  
        mystack.push(new Integer(6));  
        mystack.push(new Integer(7));  
        mystack.push(new Integer(8));  
  
        Integer x=(Integer)mystack.pop();  
        System.out.println(x.intValue());  
    }  
}
```

Hasil : 8

Map

- Menyimpan elemen dengan key unik.
- Satu key untuk satu elemen.
- Key disimpan dalam bentuk object.
- Map tidak bisa menyimpan duplicate key.
- Map bisa menyimpan duplicate element.
- Has no particular order.
- Cara mengakses elemen harus melalui key
- Contoh:
 - Hashtable
 - LinkedHashMap
 - TreeMap
 - HashMap
 - not synchronized for threads
 - permits null values to be stored



Map

```
public interface Map {
    // Basic Operations
    Object put(Object key, Object value);
    Object get(Object key);
    Object remove(Object key);
    boolean containsKey(Object key);
    boolean containsValue(Object value);
    int size();
    boolean isEmpty();

    // Bulk Operations
    void putAll(Map t);
    void clear();

    // Collection Views
    public Set keySet();
    public Collection values();
    public Set entrySet();

    // Interface for entrySet elements
    public interface Entry {
        Object getKey();
        Object getValue();
        Object setValue(Object value);
    }
}
```

Map: Hashtable

```
class CollectionTest{  
    public static void main(String [] arg){  
        Hashtable ht = new Hashtable();  
        ht.put("key1", new Integer(12));  
    }  
}
```



Map: HashMap

```
import java.util.*;
class HashMapTest{
    public static void main(String [] arg){
        HashMap hm = new HashMap();
        hm.put("Game1", "Hearts");
        hm.put(null, "Chess");
        hm.put("game3", "Checkers");
        hm.put("game3", "Whist");
        hm.put("game4", "Chess");
        System.out.println(hm);
    }
}
```

Output: {Game4=Chess, Game3=Whist, Game1=Hearts, null=Chess}




```
import java.util.*;

public class Freq {
    private static final Integer ONE = new Integer(1);

    public static void main(String args[]) {
        Map m = new HashMap();

        // Initialize frequency table from command line
        for (int i=0; i<args.length; i++) {
            Integer freq = (Integer) m.get(args[i]);
            m.put(args[i], (freq==null ? ONE :
                new Integer(freq.intValue() + 1)));
        }

        System.out.println(m.size()+" distinct words detected:");
        System.out.println(m);
    }
}

% java Freq if it is to be it is up to me to delegate
8 distinct words detected:
{to=3, me=1, delegate=1, it=2, is=2, if=1, be=1, up=1}
```

```
import java.util.*;
```

```
public class TestMap {
```

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

```
        Map m1 = new HashMap();
```

```
        m1.put(new Integer(1), "abc");
```

```
        m1.put(new Integer(2), "abc");
```

```
        m1.put(new Integer(3), "def");
```

```
        Map m2 = new HashMap();
```

```
        m2.put(new Integer(1), "klm");
```

```
        m2.put(new Integer(2), "abc");
```

```
        m2.put(new Integer(3), "def");
```

```
        m2.put(new Integer(4), "klm");
```

```
        m2.put(new Integer(5), "abc");
```

```
        m1.putAll(m2);
```

```
        System.out.println(m1);
```

```
        Set set = m1.keySet();
```

```
        System.out.println(set);
```

```
        System.out.println(m1.values());
```

```
        System.out.println(m1.entrySet());
```

```
        for (Iterator i=m1.entrySet().iterator(); i.hasNext(); ) {
```

```
            Map.Entry e = (Map.Entry) i.next();
```

```
            System.out.println(e.getKey() + ": " + e.getValue());
```

```
        }
```

```
    }
```

```
}
```

```
{2=abc, 4=klm, 1=klm, 3=def, 5=abc}
```

```
[2, 4, 1, 3, 5]
```

```
[abc, klm, klm, def, abc]
```

```
[2=abc, 4=klm, 1=klm, 3=def, 5=abc]
```

```
2: abc
```

```
4: klm
```

```
1: klm
```

```
3: def
```

```
5: abc
```



```
import java.util.*;

public class MultiMap {
    public static void main(String[] args) {
        Map m = new HashMap() ;

        String str[] ={"Andi","Ani","Anisa"};
        List l = Arrays.asList(str);

        m.put(new Integer(1),l);

        String str2[]= {"Budi","Badu","Bina"} ;
        l = Arrays.asList(str2);

        m.put(new Integer(2),l);

        System.out.println(m);
    }
}
```

```
{2=[Budi, Badu, Bina], 1=[Andi, Ani, Anisa]}
```

SortedMap: TreeMap

- Aturan mirip Map
- Beda: obyek tersimpan secara sorted berdasarkan key.
- No duplicate key.
- Elements may be duplicate.
- Key tidak boleh null value.

SortedMap: TreeMap

```
import java.util.*;
class TreeMapTest{
    public static void main(String [] args){
        SortedMap title = new TreeMap();
        title.put(new Integer(3), "Checkers");
        title.put(new Integer(1), "Euchre");
        title.put(new Integer(4), "Chess");
        title.put(new Integer(2), "Tic Tac Toe");
        System.out.println(title);
    }
}
```

Output: {1=Euchre, 2=Tic Tac Toe, 3=Checkers, 4=Chess}



Iterators

- Iterasi adalah proses mendapatkan kembali (retrieve) elemen yang terdapat dalam collection.
- **Iterator** merupakan interface yang bisa digunakan untuk meretrieve elemen collection.
- Iterator pada **Set** menghasilkan output yang non deterministic.
- Iterator pada **List** menghasilkan output secara forward.



Beberapa interface untuk iterator

- `java.lang.Iterable`
- `java.util.Iterator`
- `java.util.ListIterator`
- `java.util.Enumeration`

Iterator

- Digunakan untuk melakukan iterasi sequential (hanya satu arah dan berurutan dari awal hingga akhir)
- Semua obyek Collection mendukung penggunaan interface ini untuk melakukan iterasi terhadap anggota.

ListIterator

- **ListIterator** adalah subinterface dari **Iterator**.
- Iterasi bisa dilakukan dua arah yaitu backward dan forward.
- Dengan menggunakan ListIterator pada List, maka isi list bisa diubah dan mendapatkan posisi iterator pada list.
- Gunakan method **next** atau **previous** sebagai navigasi.

ListIterator

```
public interface ListIterator extends Iterator {
    boolean hasNext();
    Object next();

    boolean hasPrevious();
    Object previous();

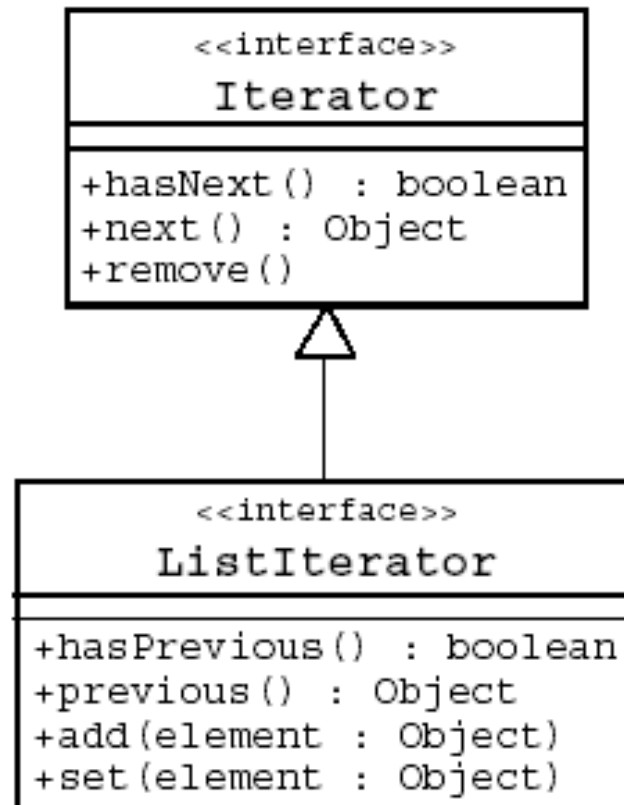
    int nextIndex();
    int previousIndex();

    void remove();           // Optional
    void set(Object o);     // Optional
    void add(Object o);     // Optional
}
```

Penggunaan Iterator

```
List list = new ArrayList();  
// add some elements  
Iterator elements = list.iterator();  
while ( elements.hasNext() ) {  
    System.out.println(elements.next());  
}
```

Hirarki Interface Iterator



Enumeration

- **Enumeration** adalah variasi dari Iterator.
- Penggunaan secara (sequential) hanya satu arah berurutan dari awal sampai akhir.
- Cara kerja Enumeration mirip dengan **Iterator**.
- Method **hasNext ()** → **hasMoreElements ()**



For example, to print all elements of a vector v :

```
for (Enumeration e = v.elements() ; e.hasMoreElements() ;) { System.out.println(e.nextElement());  
}
```

Summary of Collections

- Collections impose no order, nor restrictions, on content duplication.
- Lists maintain an order (possibly inherent in the data, possibly externally imposed).
- Sets reject duplicate entries.
- Maps use unique keys to facilitate lookup of their contents.



Summary of Collections

- For storage:
 - Using arrays makes insertion, deletion, and growing the store more difficult.
 - Using a linked list supports insertion, deletion, and growing the store, but makes indexed access slower.
 - Using a tree supports insertion, deletion, and growing the list. Indexed access is slow, but searching is faster.
 - Using hashing supports insertion, deletion, and growing the store. Indexed access is slow, but searching is particularly fast. However, hashing requires the use of unique keys for storing data elements.



Tugas

1. Buatlah resume 1 halaman mengenai Java Collection Framework dan pembagian kelompok Collection dan berikan penjelasannya.

1. Oracle Java Documentation, The Java™ Tutorials, <https://docs.oracle.com/javase/tutorial/>, Copyright © 1995, Oracle 2015.
2. Tita Karlita, Yuliana Setrowati, Rizky Yuniar Hakkun, Pemrograman Berorientasi Obyek, PENS-2012
3. Sun Java Programming, Sun Educational Services, Student Guide, Sun Microsystems, 2001.
bridge to the future
4. John R. Hubbard, Programming With Java, McGraw-Hill, ISBN: 0-07-142040-1, 2004.
5. Patrick Niemeyer, Jonathan Knudsen, Learning Java, O'reilly, CA, ISBN: 1565927184, 2000.
6. Philip Heller, Simon Roberts, Complete Java 2 Certification Study Guide, Third Edition, Sybex, San Francisco, London, ISBN: 0-7821-4419-5, 2002.
7. Herbert Schildt, The Complete Reference, Java™ Seventh Edition, Mc Graw Hill, Osborne, ISBN: 978-0-07-163177-8, 2007