Pemrograman Berorientasi Obyek

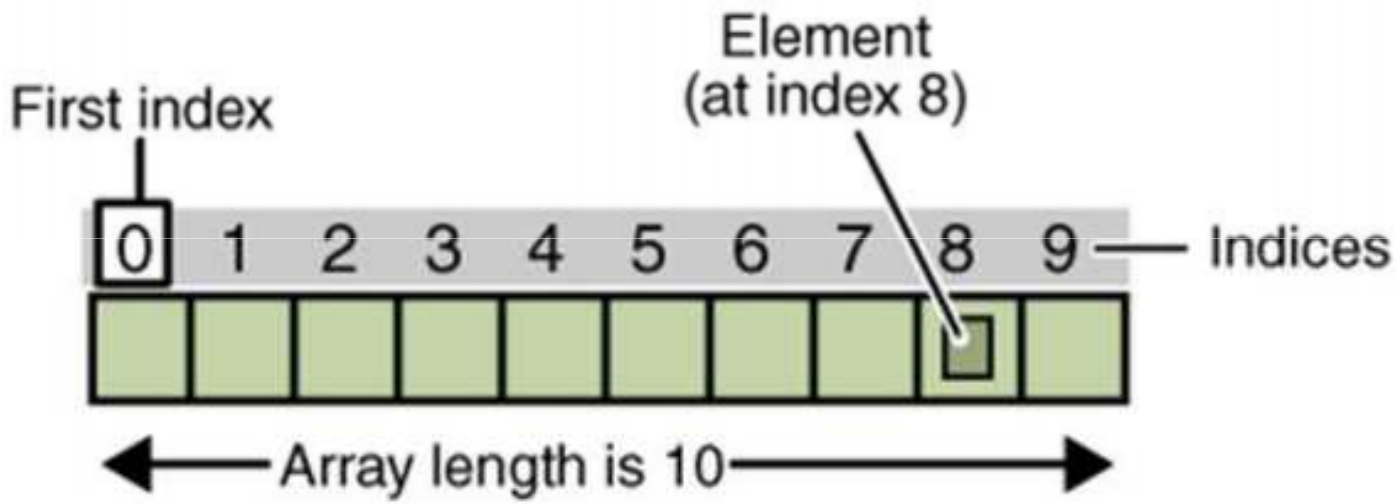
Array dan Collections

Part One: **Array**

Array

- Tipe data yang dapat menampung lebih dari satu nilai yang bertipe sama
- Menggunakan indeks untuk pengaksesannya
- Dapat diakses secara **random**, tidak harus sekuensial
- Array pada Java => bertipe Object / Reference
- Array bisa berisi:
  - Object atau tipe data primitif biasa

Ilustrasi Array



Array

- Static Memory Allocation = Fixed size
- Mendeklarasikan array
  - int[] data;
  - char[] alfabet;
  - int data[]; -> bentuk ini tidak dianjurkan !
  - String[] data;
  - Mobil[] mobilArray;

Array

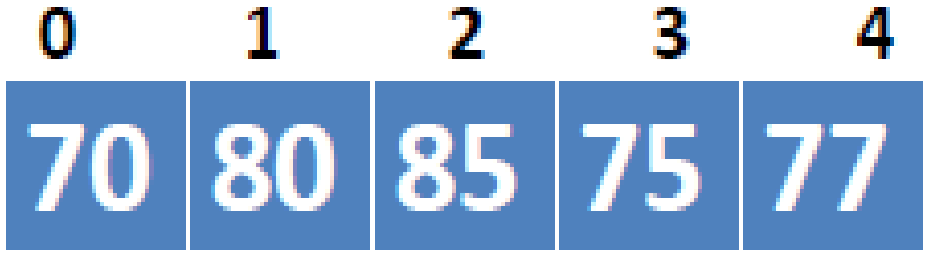
- Inisialisasi Array = menentukan ukuran (jumlah elemen)
- Inisialisasi Array
  - int[] data = new int[10];
  - String[] nama = new String[50];
  - char[] alfabet = new char[26];
- Ketiga hal diatas secara otomatis array akan berisi NULL

Array : Tipe data primitif

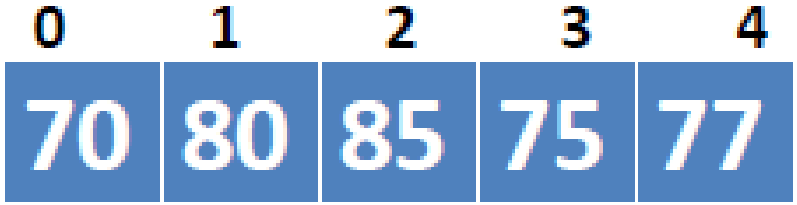
- int[] data = new int[5];

Pengisian

```
nilai[0] = 70;
nilai[1] = 80;
nilai[2] = 85;
nilai[3] = 75;
nilai[4] = 77;
```



Pengaksesan



- Mengakses elemen

```
for(int i = 0; i<5; i++) {
    System.out.println("Elemen " + i + ":" + nilai[i]);
}
```

Array object

- Inisialisasi
  - Dog[] dogArray = new Dog[5];
- Pengisian
  - dogArray[0] = new Dog(“waldo”);
  - dogArray[1] = new Dog(“froddo”);
  - dogArray[2] = new Dog(“rotty”);
  - dogArray[3] = new Dog(“percy”);
  - dogArray[4] = new Dog(“potty”);

Array object

- Mengakses Elemen

```
System.out.println(dogArray[3].getName());

for(int i=0; i<5; i++) {
   System.out.println(dogArray[i].getName());
}
```

Ukuran array

- Ukuran array dapat diambil dengan mengakses property length

```
int[] data = new int[100];
System.out.println(data.length);    100
```

Class Array

- java.util.Arrays (Helper)
- Terdapat static method:
  - Search & Sorting : binarySearch(), sort()
  - Comparison : equals()
  - Instantiation : fill();
  - Conversion : asList();

Pro dan Con

- **Kelebihan**
  - Type dari array sudah didefinisikan sejak awal (compile type checking)
  - Array mengetahui jumlah elemennya (length)
  - Array dapat menyimpan tipe data primitive secara langsung
- **Kekurangan**
  - Ukuran array tetap (fixed size)
  - Hanya dapat berisi satu type saja

Part Two: Collections

Collections

- Sebuah object yang **mengelompokkan** berbagai elemen ke dalam satu kesatuan (unit tunggal)
- Collection hanya berisi **object**
- Collection Framework : arsitektur yang merepresentasikan dan memanipulasi collections
- Ukurannya dapat bersifat **dinamis**
- Dapat menangani concurrent access (thread safe)

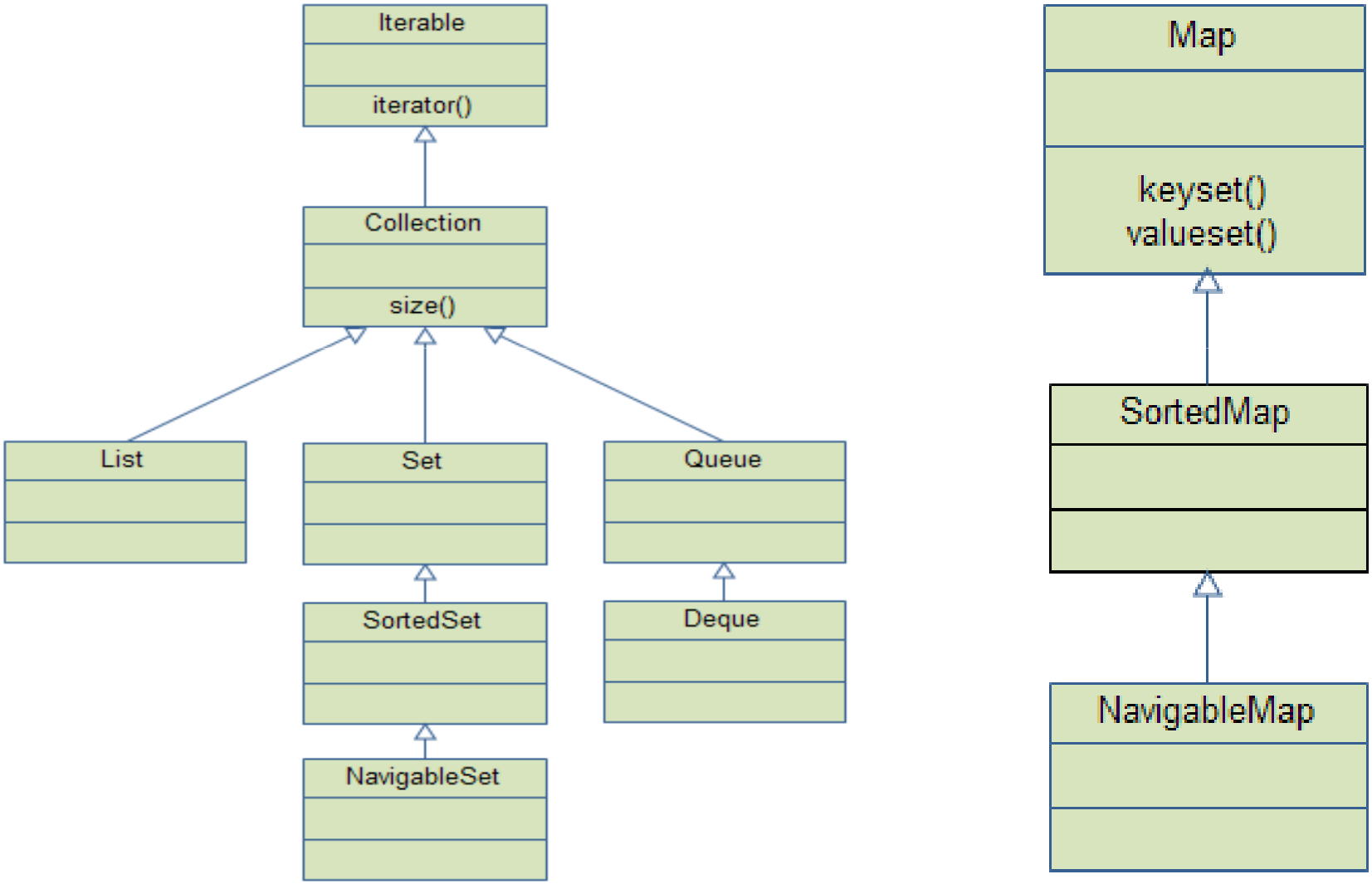
Collection framework

- Interface
  - Struktur dan karakteristik dasar
- Implementation
  - Implementasi program sesuai interface
- Algorithm
  - Algoritma program yang digunakan sesuai tujuannya

Collections interface

- Collection
  - Set
    - Extends collection tapi tidak mengizinkan duplikasi
  - List
    - Extends Collection, mengizinkan duplikasi dan penambahan fitur posisi (index)
  - Queue
    - Antrian
- Map
  - Pasangan key-value

Collection hirarki



Interface collections

```
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[]);
}
```

Interface set

- Tidak mengizinkan adanya duplikasi
- Extends dari Collection
- Implementasi berupa :
  - java.util.EnumSet
  - java.util.HashSet
  - java.util.LinkedHashSet
  - java.util.TreeSet

Interface set

- EnumSet
  - Untuk tipe data enumeration (definisi konstanta tertentu saja)
- HashSet dan LinkedHashSet
  - Implementasi menggunakan hash table
  - Tidak ada pengurutan elemen
  - add(), remove(), contains()
  - LinkedHashSet: it provides insertion-ordered iteration with linked list
- TreeSet
  - Implementasi dengan struktur pohon
  - Elemen akan selalu terurut
  - first(), last(), headSet(), and tailSet()

EnumSet Example

```
import java.util.EnumSet;

public final class EnumSetExample {
   private enum Weekday {
       SENIN, SELASA, RABU, KAMIS, JUMAT, SABTU, MINGGU;

       public static final EnumSet<Weekday> HARI_KERJA = EnumSet.range(SENIN, JUMAT);

       public final boolean isWorkday() {
           return HARI_KERJA.contains(this);
       }

       public static final EnumSet<Weekday> SEMINGGU = EnumSet.allOf(Weekday.class);
   }

   public static final void main(String[] args) {
       System.out.println("Jadwal Kerja:");
       for (final Weekday weekday : Weekday.SEMINGGU)
           System.out.println(String.format("%d. Pada hari %s kita harus " + (weekday.isWorkday() ? "bekerja" : "istirahat") + ".", weekday.ordinal() + 1, weekday)
       System.out.println("Apakah dalam satu minggu harus bekerja terus?");
       System.out.println(Weekday.HARI_KERJA.containsAll(Weekday.SEMINGGU) ? "Ya." : "Tidak");
       final EnumSet<Weekday> weekend = Weekday.SEMINGGU.clone();
       weekend.removeAll(Weekday.HARI_KERJA);
       System.out.println(String.format("Hari libur berjumlah %d hari.", weekend.size()));
   }
}
```

```
Jadwal Kerja:
1. Pada hari SENIN kita harus bekerja.
2. Pada hari SELASA kita harus bekerja.
3. Pada hari RABU kita harus bekerja.
4. Pada hari KAMIS kita harus bekerja.
5. Pada hari JUMAT kita harus bekerja.
6. Pada hari SABTU kita harus istirahat.
7. Pada hari MINGGU kita harus istirahat.
Apakah dalam satu minggu harus bekerja terus?
Tidak
Hari libur berjumlah 2 hari.
```

HashSet dan LinkedHashSet

```
import java.util.*;

public class HashSetExample {
        public static void main(String args[]) {

                HashSet HSet = new HashSet();
                LinkedHashSet LHSet = new LinkedHashSet();

                HSet.add("C");
                HSet.add("A");
                HSet.add("B");
                HSet.add("E");
                HSet.add("F");
                HSet.add("D");

                LHSet.add("X");
                LHSet.add("Z");
                LHSet.add("Y");

                System.out.println("The HashSet elements are: " + HSet);
                System.out.println("The LinkedHashSet elements are: " + LHSet);
        }
}
```

```
--------------------Configuration: <Default>---
The HashSet elements are: [D, E, F, A, B, C]
The LinkedHashSet elements are: [X, Z, Y]

Process completed.
```

TreeSet

```
import java.util.*;
public class TreeSetExample {

    public static void main (String[] args) {
        TreeSet ts = new TreeSet();
        ts.add("5");
        ts.add("2");
        ts.add("7");
        ts.add("6");
        ts.add("8");

        System.out.println ("Tree set : " + ts);
        System.out.println ("Tree first " + ts.first() + " and last " + ts.last());
        NavigableSet balik = ts.descendingSet();
        System.out.println ("Tree set descending : " + balik);
        ts.remove("6");
        System.out.println ("Tree set : " + ts);
    }

}
```

```
Tree set : [2, 5, 6, 7, 8]
Tree first 2 and last 8
Tree set descending : [8, 7, 6, 5, 2]
Tree set : [2, 5, 7, 8]

Process completed.
```

Interface List

- Elemen berada dalam **urutan** tertentu
- Mengizinkan **duplikasi**
- Elemen diakses menggunakan **index**
- Penambahan dilakukan di posisi **akhir**, penghapusan akan menghapus elemen pada posisi **awal**

Interface 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);
}
```

Interface List

- ArrayList
  - Implementasi seperti array, setiap elemen dapat diakses langsung (get(), set())
- LinkedList
  - Implementasi seperti double linked list
  - Performance lebih baik untuk operasi add(), remove()
- Vector
  - Seperti array dengan kemampuan subList()
- Stack
  - Dengan konsep Stack, memiliki method pop(), push()

ArrayList dan LinkedList

```
ArrayList<ArrayList<String>> listOlists = new ArrayList<ArrayList<String>>();
ArrayList<String> singleList = new ArrayList<String>();
singleList.add("hello");
singleList.add("world");
listOlists.add(singleList);

ArrayList<String> anotherList = new ArrayList<String>();
anotherList.add("this is another list");
listOlists.add(anotherList);
```

```
System.out.println (singleList);
System.out.println (listOlists);
```

```
[hello, world]
[[hello, world], [this is another list]]
```

```
LinkedList ls = new LinkedList();
ls.add("satu");
ls.add("dua");
ls.addFirst("nol");
ls.addLast("tiga");
System.out.println (ls);
```

```
[nol, satu, dua, tiga]
```

Vector dan Stack

```
Vector v = new Vector();
v.addElement("1");
v.addElement("2");
v.addElement("3");
v.addElement("4");
v.addElement("5");
List bagian = v.subList(1,3);
System.out.println (v);
System.out.println (bagian);

Stack s = new Stack();
s.push("A");
s.push("B");
s.push("C");
System.out.println (s);
s.pop();
System.out.println (s);
```

```
[1, 2, 3, 4, 5]
[2, 3]
[A, B, C]
[A, B]
```

Interface Map

- Memetakan kunci untuk nilai (keys to values)
- Associative array atau dictionary
- Operasi elemen
  - put(Object key, Object value)
  - remove(Object key);
  - get(Object key);

Interface Map

- HashMap
  - Implementasi menggunakan hash table
  - Pasangan key-value **tidak diurutkan**
- TreeMap
  - Implementasi berupa tree
  - Pasangan key-value selalu **terurut** berdasarkan key

HashMap dan TreeMap

```
import java.util.*;

public class MapExample {
    public static void main (String[] args) {
        HashMap tabungan = new HashMap();
        tabungan.put("anton",1000);
        tabungan.put("yuan", 1500);
        tabungan.put("mahas", 2000);

        System.out.println (tabungan);

        int uang = (int)tabungan.get("yuan");
        tabungan.put("yuan", uang+250);
        System.out.println (tabungan);

        TreeMap ipk = new TreeMap();
        ipk.put("anton",3.5);
        ipk.put("yuan", 3.1);
        ipk.put("mahas", 3.4);

        System.out.println (ipk);

        Double temp = (Double)ipk.get("yuan");
        ipk.put("yuan", temp+0.1);
        System.out.println (ipk);
    }
}
```

```
--------------------Configuration: <Defa
{mahas=2000, yuan=1500, anton=1000}
{mahas=2000, yuan=1750, anton=1000}
{anton=3.5, mahas=3.4, yuan=3.1}
{anton=3.5, mahas=3.4, yuan=3.2}

Process completed.
```

Interface Queue

- Interface Queue menggunakan prinsip antrian
- Untuk mengimplementasikan:
  - Queue q = new LinkedList();
  - Queue q2 = new PriorityQueue();
- Untuk menghapus elemen antrian: method poll()

QueueExample

```
import java.util.*;
public class QueueExample {

    public static void main (String[] args) {
        Queue q2 = new PriorityQueue();
        q2.add("a");
        q2.add("b");
        q2.add("c");
        System.out.println (q2);
        q2.poll();
        System.out.println (q2);
    }

}
```

```
[a, b, c]
[b, c]

Process completed.
```

Kelebihan dan Kekurangan Collection

- Kelebihan
  - Dapat diisi berbagai macam object
  - Ukurannya dinamis
- Kekurangan
  - Bukan compile type checking
  - Object yang diambil dari collection harus dicast terlebih dahulu

Next

- Class Diagram