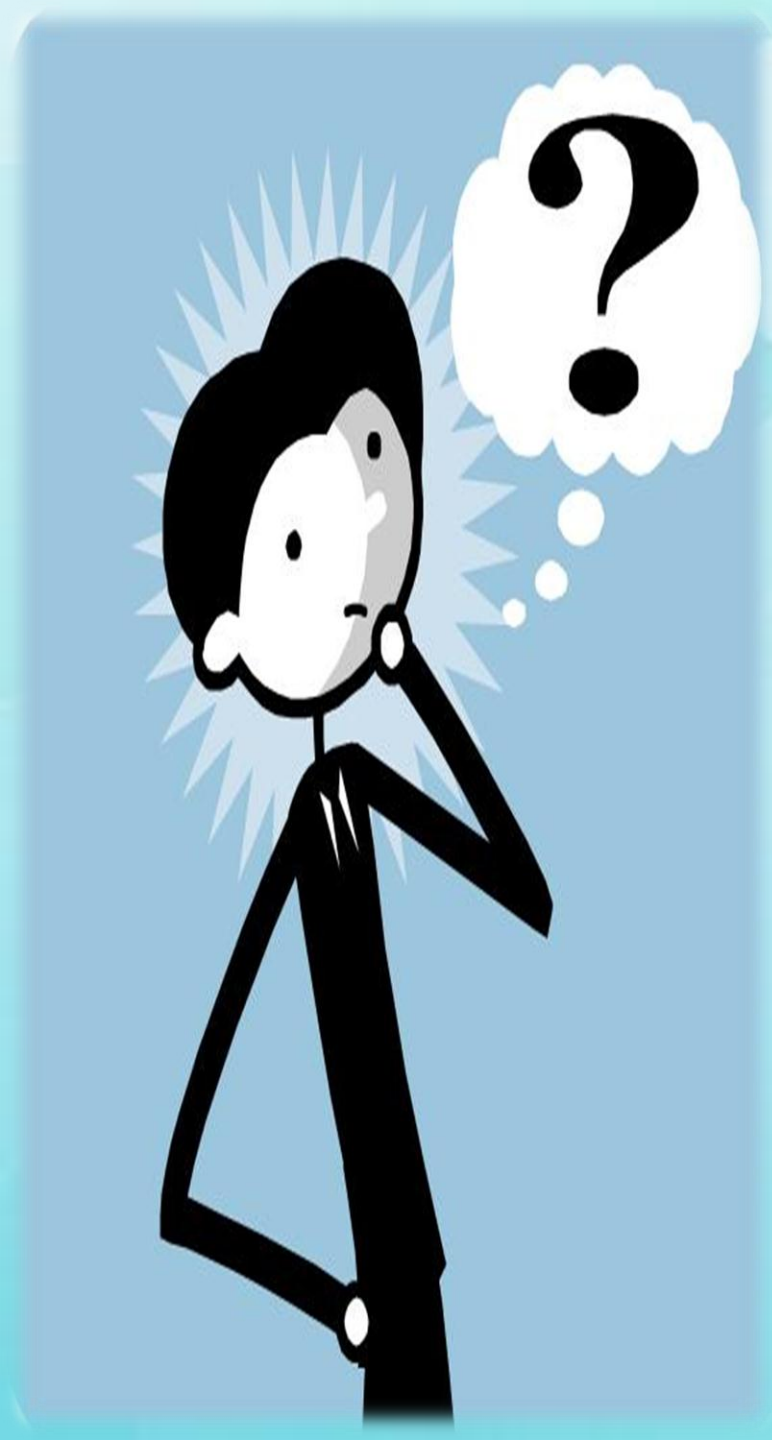
# Pemrograman Berorientasi Objek

## Collection dan Multithreading


Adam Mukharil Bachtiar

Teknik Informatika UNIKOM



# Generic Programming, Collection, dan Multithreading

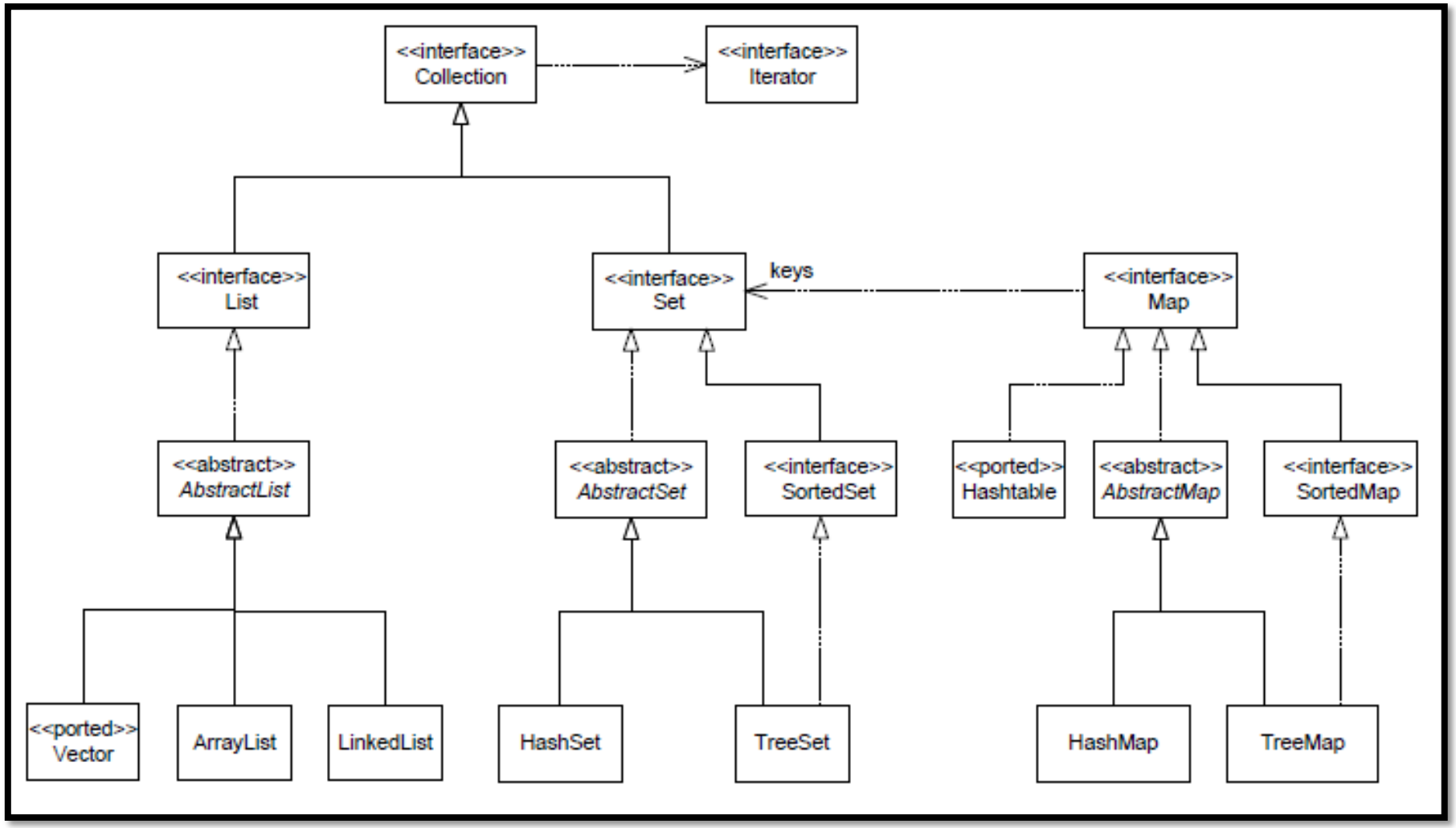
1. Definisi dan konsep Collection
2. Iterator
3. Pembahasan collection
4. Definisi dan konsep multithreading
5. Pembahasan multithreading

# COLLECTION

# Definisi Collection

Kumpulan data yang dimanipulasi sebagai single objek. Collection lebih dikenal sebagai struktur data.

# Jenis-Jenis Collection



# Operasi Umum Collection

- Adding
- Removing
- Searching
- Sorting
- Iterating

# Kegunaan Collection

- Program yang menggunakan konsep collection tidak terikat pada implementasi.
- Penggunaan generic dan Object memungkinkan penggunaan tipe data yang bebas.

# Format Dasar 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[]);
}
```

# Definisi Iterator

Class yang digunakan untuk menyeleksi elemen dari sebuah collection. Tujuanny adalah menyembuyikan collection agar tidak diakses secara sembarangan.

# Konsep Iterator

# Format Umum Iterator

```
// the interface definition
Interface Iterator {
   boolean hasNext();
   Object next();           // note "one-way" traffic
   void remove();
}



// an example
public static void main (String[] args){
   ArrayList cars = new ArrayList();
   for (int i = 0; i < 12; i++)
      cars.add (new Car());

   Iterator it = cats.iterator();
   while (it.hasNext())
      System.out.println ((Car)it.next());
}
```

# LIST

# Definisi List

List merupakan interface yang berkoresponden terhadap kumpulan objek. Duplikasi diperbolehkan pada list. Penerapan yang sering digunakan adalah ArrayList dan LinkedList.

# Format Interface List

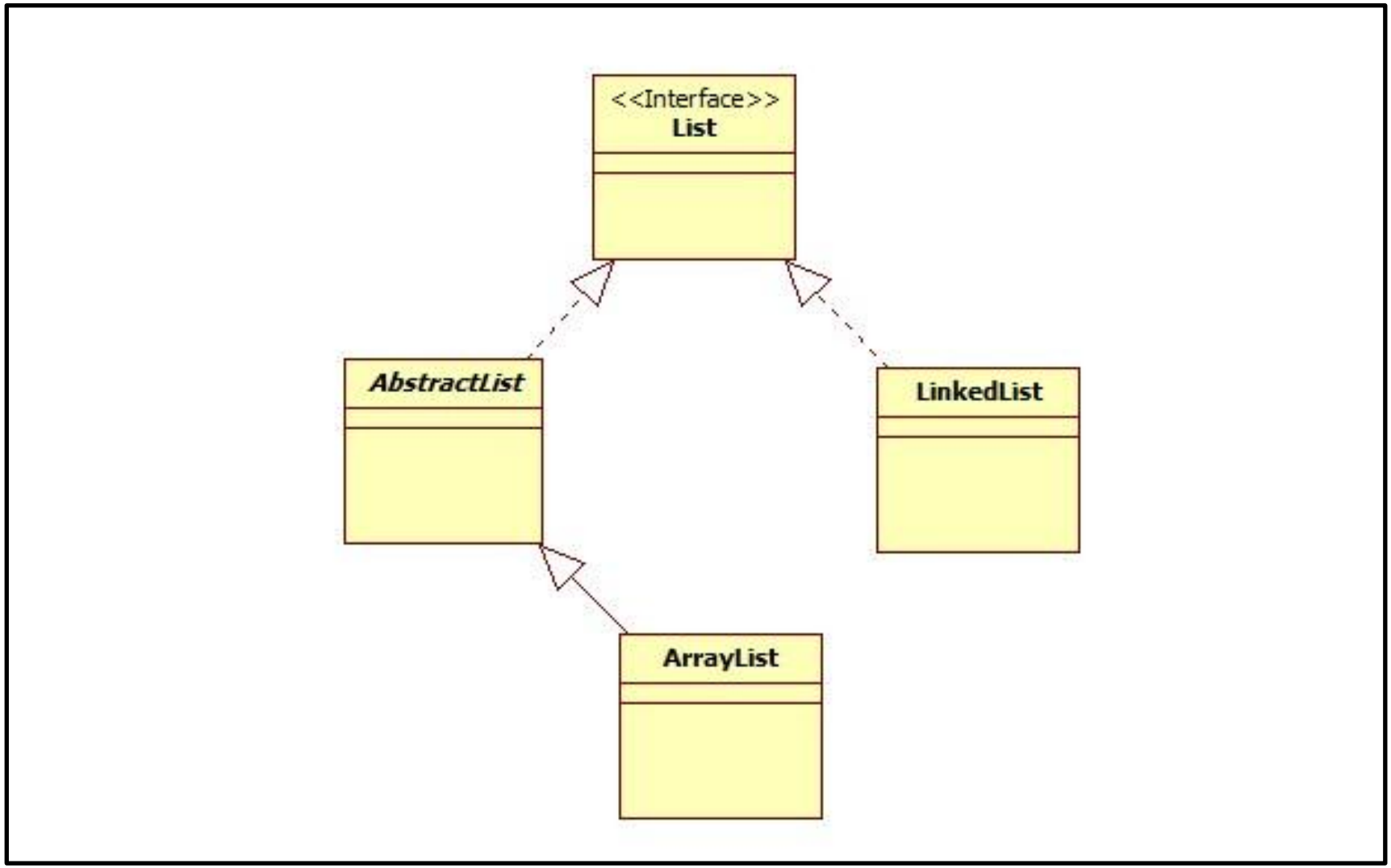
```
public interface List extends Collection {
    // Positional Access
    Object get(int index);
    Object set(int index, Object element); // Optional
    void add(int index, Object element);   // Optional
    Object remove(int index);              // Optional
    abstract boolean addAll(int index, 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);
}
```

# Struktur List



# Definisi ArrayList

Collection yang memiliki sifat yang memiliki sifat dan bentuk yang mirip array tapi tidak terikat pada suatu ukuran yang spesifik. Biasa disebut array dinamis.

# Operasi ArrayList

- add
- clear
- remove
- size
- equal
- get
- set
- isEmpty

# Contoh Penggunaan ArrayList

```
public class ArrayListTester {
     public static void main(String[] args) {
         ArrayList<String> nama=new ArrayList<String>();
         String nama1="Adam";
         String nama2="Mira";

         nama.add(nama1);
         nama.add(nama2);

         //for each
         for(String name:nama){
             System.out.println(name);
         }
     }
}
```

# Definisi LinkedList

Collection yang memiliki kemampuan untuk add dan remove yang lebih baik dibandingkan list pada umumnya. Bisa dikembangkan menjadi stack maupun queue.

# Operasi LinkedList

- addFirst
- addLast
- getFirst
- getLast
- removeFirst
- removeLast
- push
- pop

# Contoh Penggunaan LinkedList

```
import java.util.*;
public class MyStack {
    private LinkedList list = new LinkedList();
    public void push(Object o){
        list.addFirst(o);
    }
    public Object top(){
        return list.getFirst();
    }
    public Object pop(){
        return list.removeFirst();
    }

    public static void main(String args[]) {
        Car myCar;
        MyStack s = new MyStack();
        s.push (new Car());
        myCar = (Car)s.pop();
    }
}
```

# Contoh Iterator List

```
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
}
```

# MAP

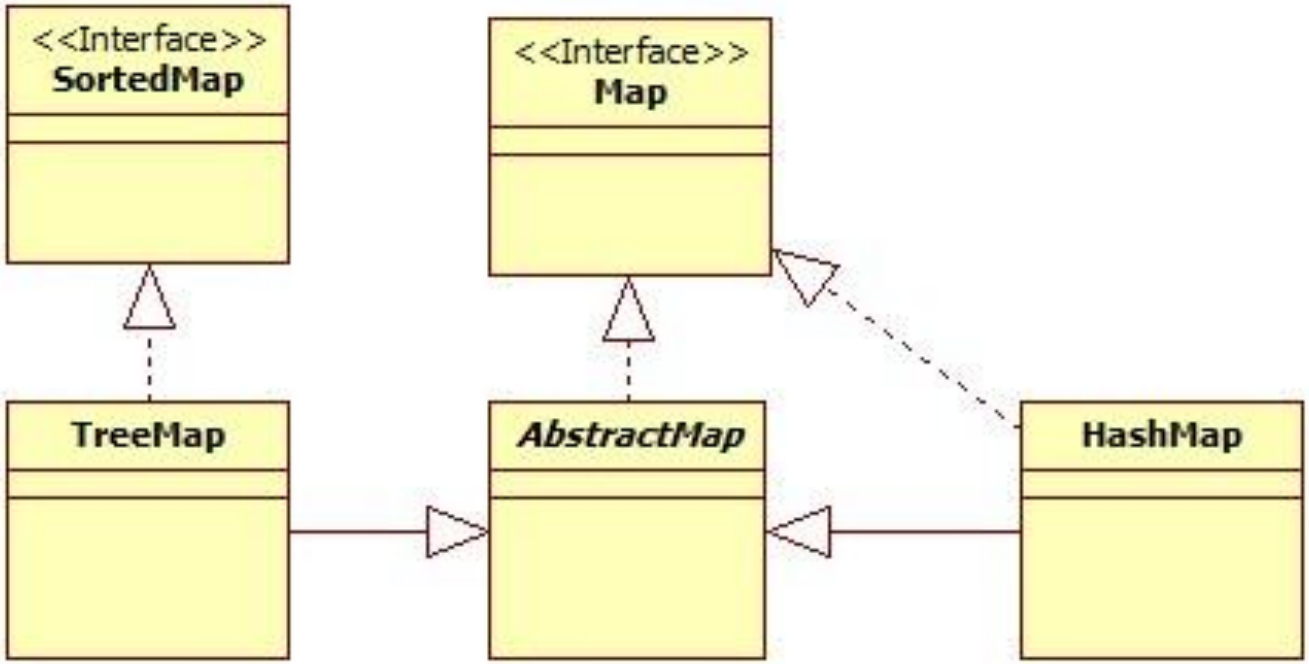
# Definisi Map

Map merupakan objek yang memetakan kunci terhadap suatu value. Sering disebut sebagai associative array atau kamus. Penerapan yang sering digunakan adalah HashMap dan TreeMap.

# Format Interface 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);
    }
}
```

# Struktur Umum Map



# Definisi HashMap

- Implementasi HashMap berdasarkan pada sebuah hash table.
- Nilai diakses melalui kunci.
- Apabila ada value yang memiliki kunci yang sama maka akan terjadi penimpaan value.

# Format Class HashMap

```
public class HashMap extends AbstractMap
{
  public void       clear();
  public boolean    containsKey( Object key );
  public boolean    containsVlaue( Object value );
  public Set        entrySet();
  public Object     get( Object key );
  public boolean    isEmpty();
  public Set        keySet();
  public void       put( Object key, Object value );
  public Object     remove( Object key );
  public int        size();
  public Collection values();
}
```

# Contoh Penggunaan HashMap

```
public class ContohIteratorTester {
     public static void main(String[] args) {
         HashMap alat=new HashMap();
         alat.put(1,"Palu");
         alat.put(2,"Bor");

         System.out.println(alat.get(1));
         System.out.println(alat.get(2));

         alat.put(1,"Paku");
         System.out.println(alat.get(1));
     }
}
```

# Definisi TreeMap

- Pasangan kunci disimpan dalam sebuah pohon yang berurutan.
- Pengurutan pohon didasarkan pada kunci.
- Kunci yang digunakan harus merupakan turunan dari kelas yang bisa dibandingkan (interface comparable dan comparator).

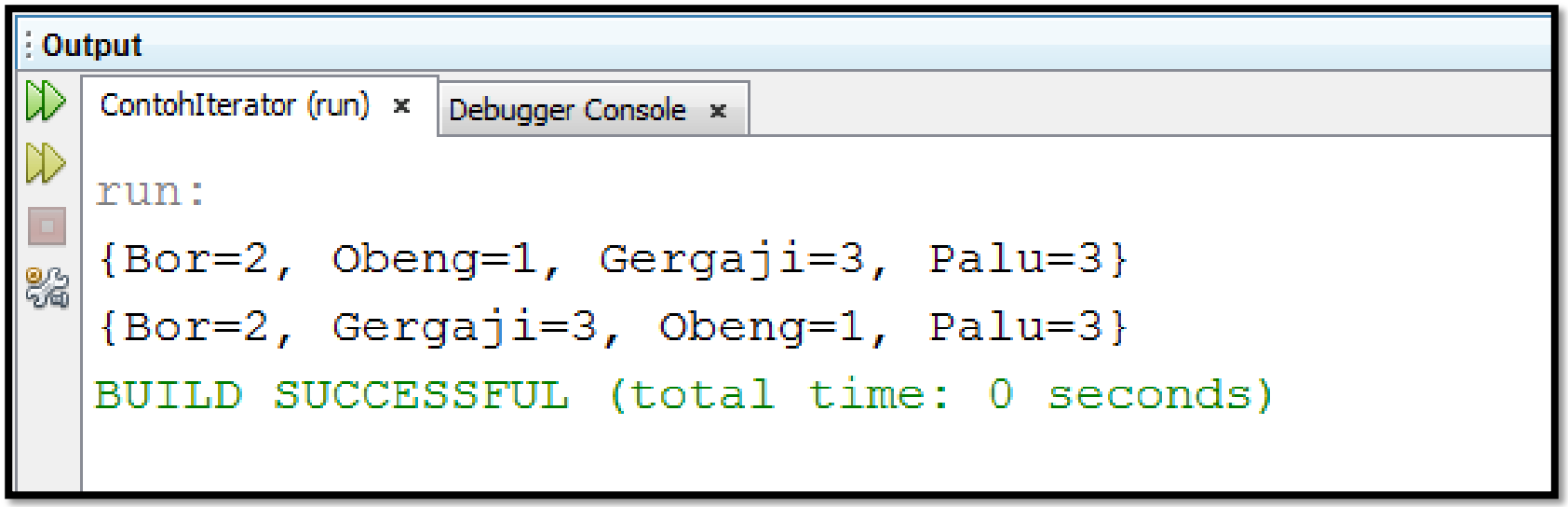
# Perbedaan HashMap dan TreeMap

```
public class HashMapTreeMapTester {
     public static void main(String[] args) {
        Map<String,Integer> alat=new HashMap();
        alat.put("Palu",3);
        alat.put("Bor",2);
        alat.put("Gergaji",3);
        alat.put("Obeng",1);

        System.out.println(alat);

        TreeMap perkakas=new TreeMap(alat);
        System.out.println(perkakas);
    }
}
```

# Perbedaan HashMap dan TreeMap

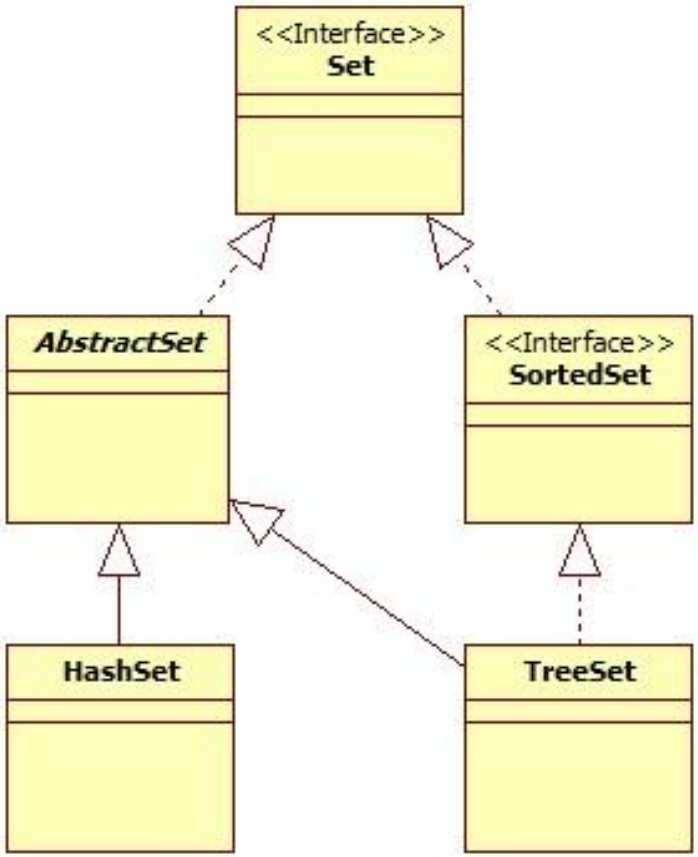


# SET

# Definisi Set

Set merupakan collection yang tidak mengizinkan ada elemen yang duplikat. Dua penerapan yang sering digunakan adalah HashSet dan TreeSet

# Struktur Umum Set



# Operasi Set

- add
- remove
- addAll
- equal

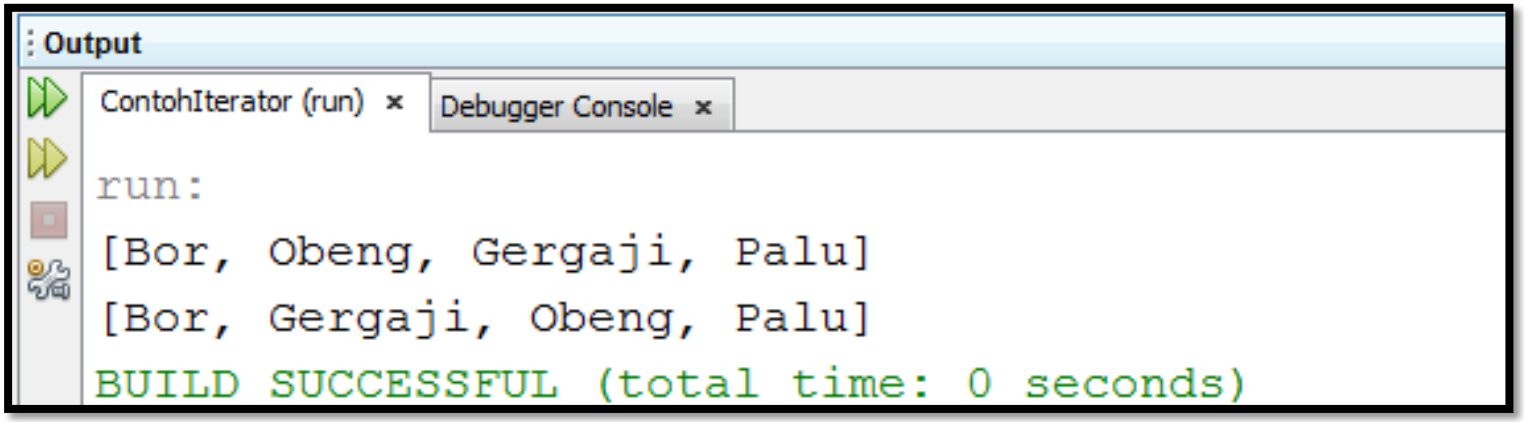
- set1 ∪ set2
  - `set1.addAll(set2)`
- set1 ∩ set2
  - `set1.retainAll(set2)`
- set1 – set2
  - `set1.removeAll(set2)`

# Perbedaan HashSet dan TreeSet

```
public class HashSetTreeSetTester {
     public static void main(String[] args) {
         Set alat=new HashSet();
         alat.add("Palu");
         alat.add("Bor");
         alat.add("Gergaji");
         alat.add("Obeng");
         alat.add("Bor");
         System.out.println(alat);

         TreeSet perkakas=new TreeSet(alat);
         System.out.println(perkakas);
     }
}
```

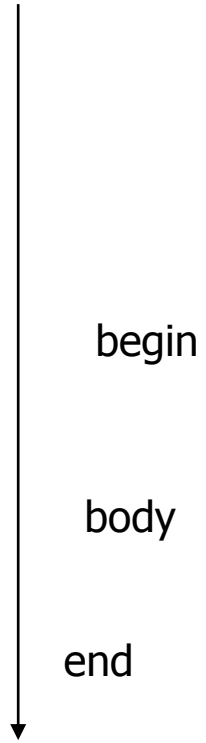
# Perbedaan HashSet dan TreeSet



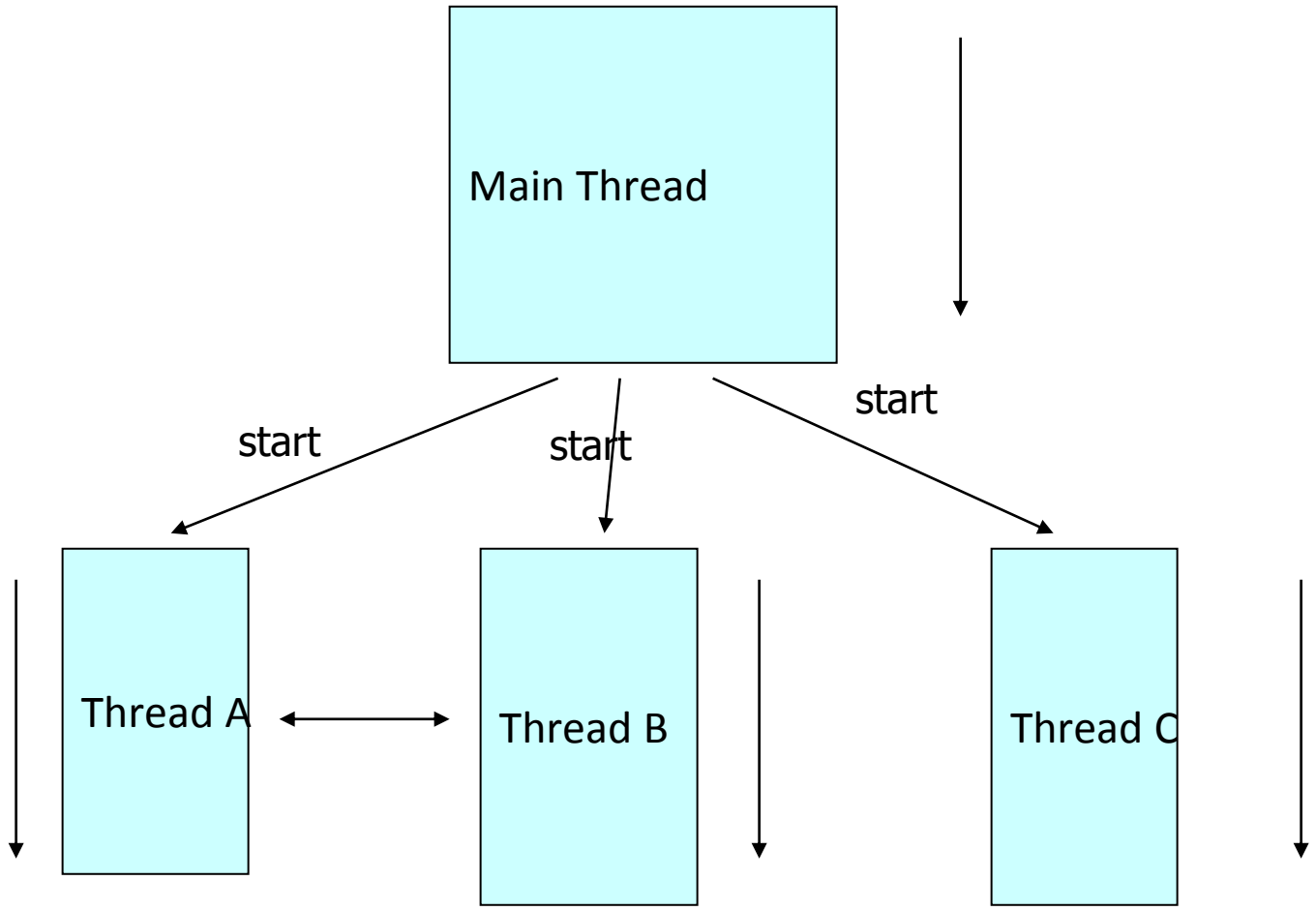
# MULTITHREAD

# Single Thread Program

```
class ABC
{
....
    public void main(..)
    {
    ...
    ..
    }
}
```



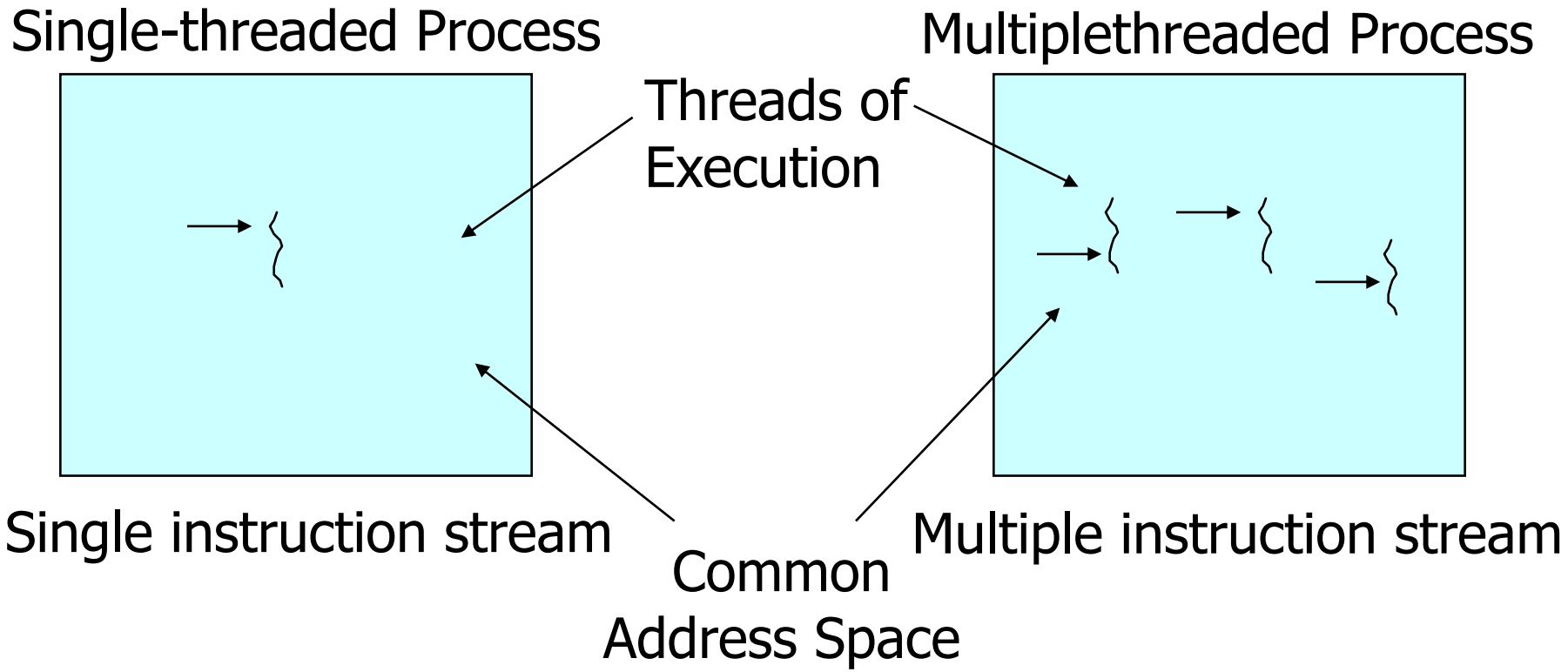
# Multithreaded Program




**Threads memungkinkan pertukaran atau pergantian data/hasil**

# Single Thread VS. Multithread

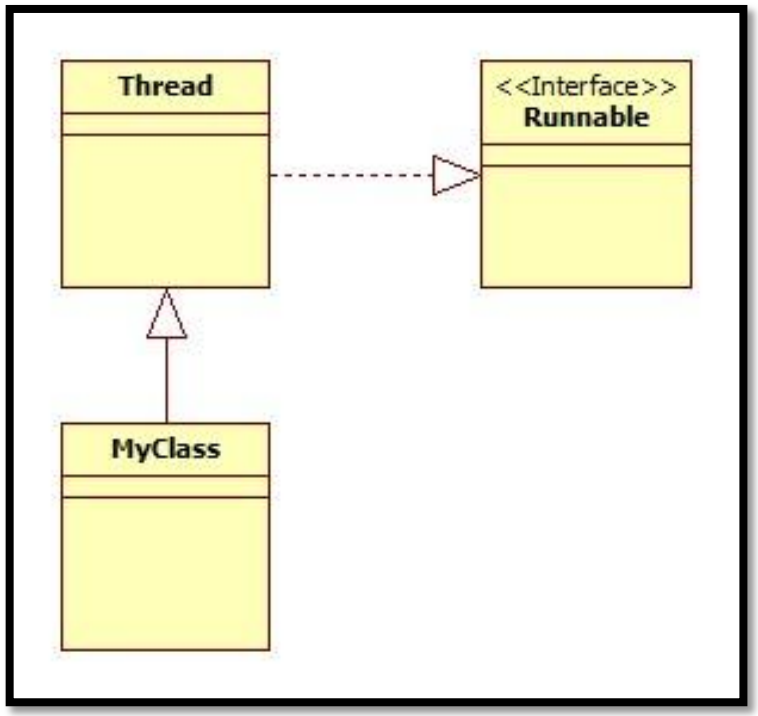
**Thread adalah proses yang ringan di dalam sebuah proses**



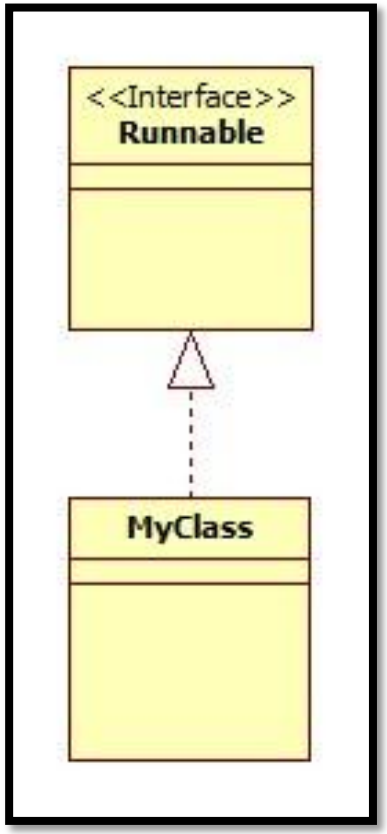
# Definisi Thread

- Kode program yang berjalan secara konkuren dengan kode program lain.
- Penggunaan thread memungkinkan lebih dari satu kode program dijalankan secara bersamaan.

# Cara Multithread di JAVA




Meng-extend class Thread




Mengimplementasi interface Runnable

# Meng-extend Class Thread

- Buat sebuah kelas yang meng-extend class Thread dan override method **run()**.
- Buat sebuah thread (membuat **objek** dari kelas tersebut).
- Eksekusi thread dengan memanggil method **start()**.
- Create and Execute: **new MyClass().start();**

# Contoh Penggunaan

```
public class Something extends Thread{
    @Override
    public void run(){
        System.out.println("Thread berjalan di sini");
    }
}


public class Thread1 {
     public static void main(String[] args) {
        Something thread=new Something();
        thread.start();
    }
}
```

# Mengimplementasi Interface Runnable

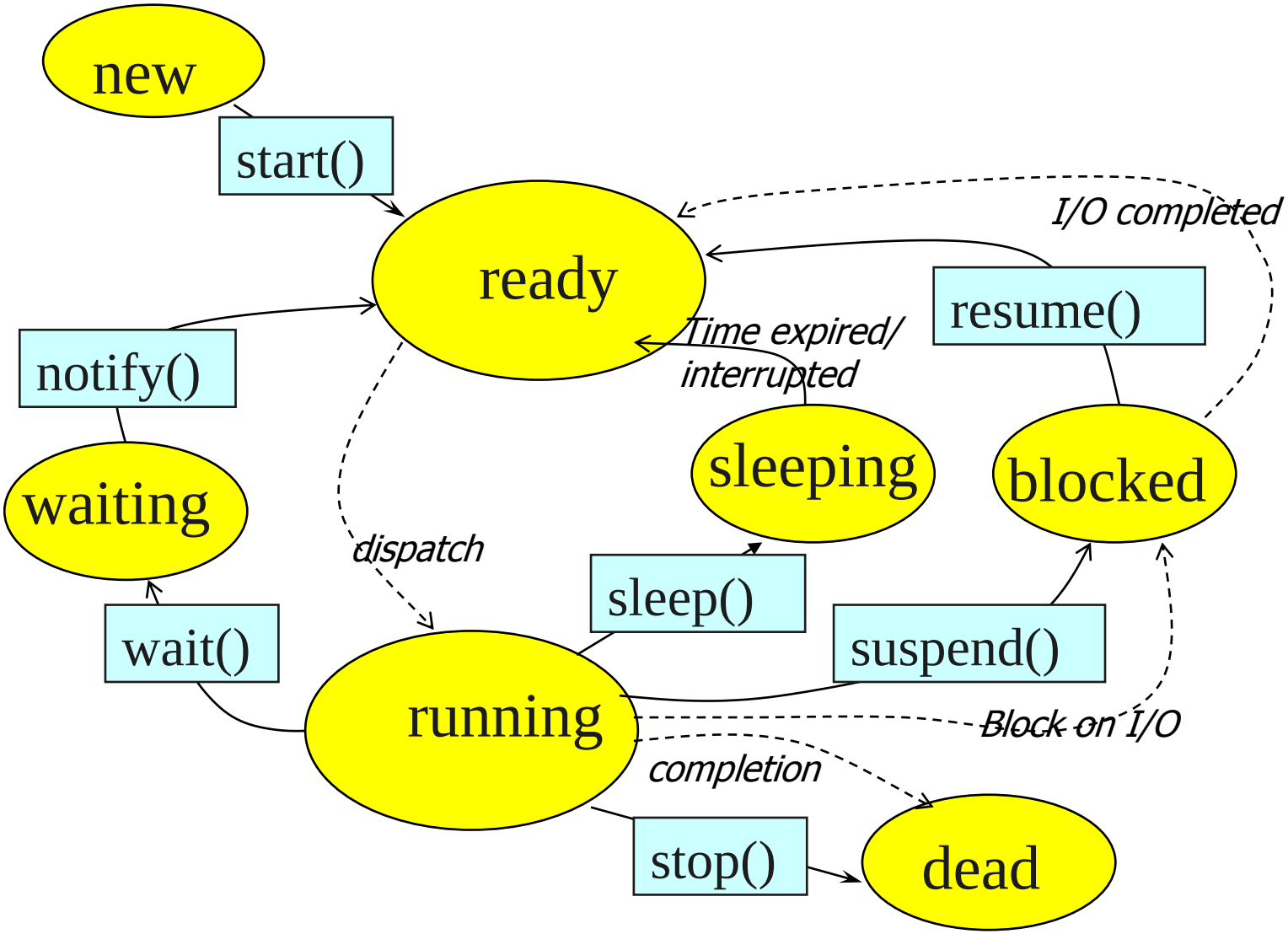
- Buat sebuah class yang mengimplementasi **interface Runnable** dan override method **run()**.
- Buat objek dari kelas tersebut (misal: **obj**).
- Buat objek thread: Thread thr1=new Thread(obj);
- Mulai eksekusi thread: **thr1.start();**

# Contoh Penggunaan

```
public class Something implements Runnable{
    @Override
    public void run(){
        System.out.println("Thread berjalan di sini");
    }
}

public class Thread1 {
     public static void main(String[] args) {
        Something sm=new Something();
        Thread thr1=new Thread(sm);
        thr1.start();
    }
}
```

# Siklus Hidup Thread



# Contoh Three Thread

```
public class A extends Thread{
    @Override
    public void run(){
        for(int i=1;i<=5;i++){
            System.out.println("Thread A : "+i);
        }
        System.out.println("Selesai dari thread A");
    }
}
public class B extends Thread{
    @Override
    public void run(){
        for(int j=1;j<=5;j++){
            System.out.println("Thread B : "+j);
        }
        System.out.println("Selesai dari thread B");
    }
}
```

# Contoh Three Thread

```
public class C extends Thread{
    @Override
    public void run(){
        for(int k=1;k<=5;k++){
            System.out.println("Thread C : "+k);
        }
        System.out.println("Selesai dari thread C");
    }
}

public class ThreeThread {
    public static void main(String[] args) {
        new A().start();
        new B().start();
        new C().start();
    }
}
```

# Prioritas Thread

- Sebuah thread dapat ditentukan prioritasnya.
- Prioritas akan menentukan penjadwalan sebuah thread.
- Panggil method setPriority(int number); :

1. MIN_PRIORITY = 1
2. NORM_PRIORITY = 5 → **DEFAULT**
3. MAX_PRIORITY = 10

# Contoh Prioritas Thread

```
public class A extends Thread{
    @Override
    public void run(){
        for(int i=1;i<=5;i++){
            System.out.println("Thread A : "+i);
        }
        System.out.println("Selesai dari thread A");
    }
}
public class B extends Thread{
    @Override
    public void run(){
        for(int j=1;j<=5;j++){
            System.out.println("Thread B : "+j);
        }
        System.out.println("Selesai dari thread B");
    }
}
```

# Contoh Prioritas Thread

```
public class C extends Thread{
    @Override
    public void run(){
        for(int k=1;k<=5;k++){
            System.out.println("Thread C : "+k);
        }
        System.out.println("Selesai dari thread C");
    }
}
```

# Contoh Prioritas Thread

```
public class ThreeThread {
     public static void main(String[] args) {
        // TODO code application logic here
        A a=new A();
        B b=new B();
        C c=new C();
        c.setPriority(Thread.MAX_PRIORITY);
        b.setPriority(a.getPriority()+1);
        a.setPriority(Thread.MIN_PRIORITY);
        System.out.println("Thread A dimulai");
        a.start();
        System.out.println("Thread B dimulai");
        b.start();
        System.out.println("Thread C dimulai");
        c.start();
        System.out.println("Akhir dari thread");
    }
}
```

# THE END