MINGGU 9

Praktikum Pemrograman Berbasis Objek
oleh Asisten IF2210 2019/2020

OUTLINE

1. Generic
2. Java API

GENERIC

- Generic pada Java dapat dibatasi oleh suatu kelas atau interface
  <T extends Number>
  <E extends Cloneable & Serializable>

- Namun, Java tidak dapat menyatakan non-type parameter
  *ingat **template<class T, int N>** di C++?*

CONTOH METHOD GENERIC: MAX ELEMENT

```cpp
// element.cpp
template<class T>
T max_elmt(T* arr, int N)
{
    T max_result = arr[0];
    for (int i = 0; i < N; i++) {
        if (max_result < arr[i]) {
            max_result = arr[i];
        }
    }
    return max_result;
}
```

```java
// Element.java
public class Element {
   public static <T> T max_elmt(T[] arr, int N)
   {
       T max_result = arr[0];
       for (int i = 0; i < N; i++) {
           if (max_result < arr[i]) {
               max_result = arr[i];
           }
       }
       return max_result;
   }
}
```

Di java, operator binary hanya dapat digunakan pada tipe primitif

CONTOH METHOD GENERIC: MAX ELEMENT

```cpp
// element.cpp
template<class T>
T max_elmt(T* arr, int N)
{
    T max_result = arr[0];
    for (int i = 0; i < N; i++) {
        if (max_result < arr[i]) {
            max_result = arr[i];
        }
    }
    return max_result;
}
```

```java
// Element.java
public class Element {
   public static <T extends Number> T max_elmt(T[] arr, int N)
   {
       T max_result = arr[0];
       for (int i = 0; i < N; i++) {
           if (max_result.doubleValue() <
             arr[i].doubleValue()) {
               max_result = arr[i];
           }
       }
       return max_result;
   }
}
```

Method abstract doubleValue() dari Number dapat dimanfaatkan untuk mendapat nilai double

PEMANGGILAN GENERIC METHOD

```
public class Element {
...
   public static void main(String[] args) {
        Element e = new Element();
        Integer[] arr = new Integer[]{1,2,3};
        Integer max1 = e.<Integer>max_elmt(arr, 3); // OK
        Integer max2 = e.max_elmt(arr, 3); // OK
   }
}
```

CONTOH GENERIC CLASS: STACK

```cpp
// stack.h
#ifndef STACK_H
#define STACK_H

template<class T>
class Stack {
private:
    int size;
    int capacity;
    T* data;
public:
    Stack() {
       this->capacity = 10;
       this->size = 0;
       this->data = new T[this->capacity];
    }
...
};

#endif
```

```java
// Stack.java
public class Stack<T> {
   // size tidak dideklarasi karena
   // sudah tersedia dari ArrayList
   private ArrayList<T> data;

   public Stack() {
       this.capacity = 10;
       this.data = new ArrayList<T>();
   }
   ...
}
```

Di java, array tidak dapat berisi tipe generik[1]
Oleh karena itu, kita menggunakan ArrayList.
ArrayList akan dijelaskan di **Java API**.

[1] https://docs.oracle.com/javase/tutorial/java/generics/restrictions.html

SUBTYPING

- Relasi “is-a”: ketika parent class adalah “sejenis” dari child class sehingga semua karakteristik parent dimiliki oleh child.

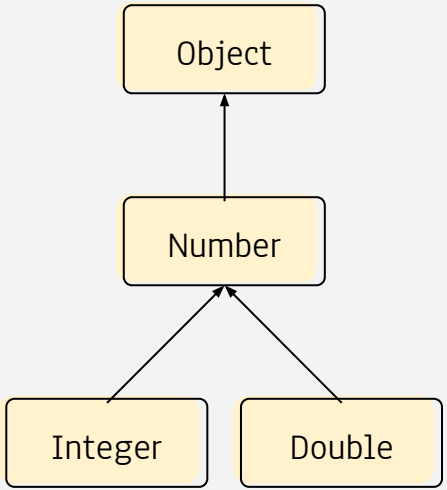
```
public void add(Number n1, Number n2) { /* ... */ }

add(new Integer(10), new Double(20.0));    // OK
```

WILDCARD

- Wildcard (?) merepresentasikan tipe unknown
- Tipe unknown diperlakukan seperti tipe Object
- Ada dua jenis wildcard:
  1. Unbounded **<?>**
  2. Bounded **<? extends Number>**
- Bounded wildcard sendiri memiliki dua jenis:
  1. Upper Bounded wildcard **<? extends Number>**
  2. Lower Bounded wildcard **<? super Integer>**

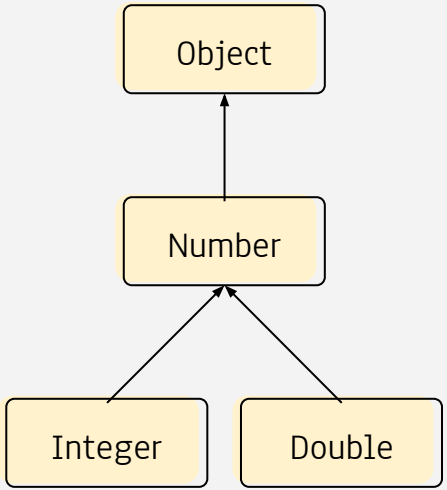
PENGGUNAAN UPPER BOUNDED WILDCARD

```
public class Element {
...
   public void printList(List<? extends Number> list) { /* implementasi */ }

   public static void main(String[] args) {
       Element elmt = new Element();

       ArrayList<Integer> l1 = new ArrayList<>();
       l1.add(new Integer(1));

       ArrayList<Object> l2 = new ArrayList<>();
       l2a.add(new Integer(2));

       ArrayList<Number> l3 = new ArrayList<>();
       l2.add(new Double(3.0));

       ArrayList<Double> l4 = new ArrayList<>();
       l3.add(new Double(4.0));

       elmt.printList(l1); // OK
       elmt.printList(l2); // Compile error, Object bukan subtype dari Number
       elmt.printList(l3); // OK
       elmt.printList(l4); // OK
   }
}
```



PENGGUNAAN LOWER BOUNDED WILDCARD

```
public class Element {
...
   public void printList(List<? super Integer> list) { /* implementasi */ }

   public static void main(String[] args) {
       Element elmt = new Element();

       ArrayList<Integer> l1 = new ArrayList<>();
       l1.add(new Integer(1));

       ArrayList<Object> l2 = new ArrayList<>();
       l2a.add(new Integer(2));

       ArrayList<Number> l3 = new ArrayList<>();
       l2.add(new Double(3.0));

       ArrayList<Double> l4 = new ArrayList<>();
       l3.add(new Double(4.0));

       elmt.printList(l1); // OK
       elmt.printList(l2); // OK
       elmt.printList(l3); // OK
       elmt.printList(l4); // Compile error, Double bukan super type dari Integer
   }
}
```



GENERIC TYPE VS WILDCARD

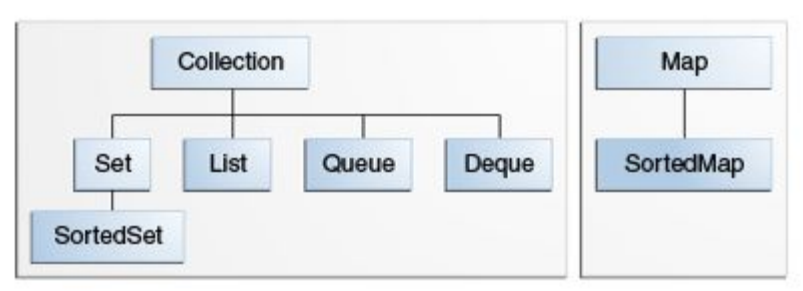
GENERIC TYPE	WILDCARD
Dapat memiliki >1 bound	Hanya dapat memiliki satu bound
Tidak ada lower bound, hanya upper bound (extends)	Ada lower bound (super) dan upper bound (extends)
Dapat digunakan ketika meng-enforce tipe yang berhubungan `public static <T extends Number> void copy(List<T> dest, List<T> src)`	Tidak dapat digunakan untuk meng-enforce tipe yang berhubungan `public static void copy(List<?> dest, List<?> src) // tipe isi List dest dan src dapat berbeda`

Tipe dapat digunakan di-refer kembali pada method body
 	
`public <T extends Number> Map<T, String> convertToMap(ArrayList<T> list) {`
`  Map<T, String> names = new HashMap<T, String>();`
`  ...`
`}`
	Tipe tidak dapat digunakan di-refer kembali pada method body

JAVA API

- Sebuah **Collection** adalah object yang mengumpulkan banyak elemen menjadi sebuah unit
- Java memiliki **Collection Framework**, yaitu arsitektur yang digunakan untuk merepresentasi sebuah collection

- Sebuah **Collection Framework** memiliki:
  - **Interface**: Abstract Data Type yang digunakan
  - **Implementation**: implementasi dari interface-nya
  - **Algorithm**: method that perform useful computation, such as sorting and searching



Core Collection interface, circa JDK 8, colorized

COLLECTION INTERFACE: SET

```
import java.util.*;

public class FindDups {
  public static void main(String[] args) {
    Set<String> s = new HashSet<String>();
    for (String a : args)
      s.add(a);
    System.out.println(s.size() + " distinct words: " + s);
   }
}
```

Set adalah Collection yang tidak bisa memiliki elemen duplikat.

Implementasi Set:

- HashSet
- TreeSet
- LinkedList

COLLECTION INTERFACE: LIST

```
import java.util.*;

public class Shuffle {
   public static void main(String[] args) {
      List<String> list = new ArrayList<String>();
      for (String a : args)
         list.add(a);
      Collections.shuffle(list, new Random());
      System.out.println(list);
   }
}
```

List adalah Collection yang urutan elemennya disesuaikan dengan urutan pemasukannya.

Implementasi List:

- ArrayList
- LinkedList

COLLECTION INTERFACE: QUEUE

```java
import java.util.*;

public class Countdown {
    public static void main(String[] args) throws InterruptedException {
        int time = Integer.parseInt(args[0]);
        Queue<Integer> queue = new LinkedList<Integer>();
        for (int i = time; i >= 0; i--)
            queue.add(i);
        while (!queue.isEmpty()) {
            System.out.println(queue.remove());
            Thread.sleep(1000);
        }
    }
}
```

Queue adalah Collection yang menyimpan elemen sebelum preprocessing, umumnya bersifat FIFO.

COLLECTION INTERFACE: DEQUE

```
import java.util.*;

public class DequeExample {
  public static void main(String[] args){
    Deque<String> deque = new
LinkedList<String>();

    deque.add("Element 1");         // Tail
    deque.addFirst("Element 2");    // Head
    deque.addLast("Element 3");     // Tail
    deque.push("Element 4");        // Head
    deque.offerFirst("Element 5"); // Head
    deque.offerLast("Element 6");  // Tail
```

```
    // Pop returns the head, and removes it
from the deque
    System.out.println("Pop " + deque.pop());
    System.out.println("After pop: " + deque);

    // We can remove the first / last element.
    deque.removeFirst();
    deque.removeLast();
    System.out.println("Deque after removing "
+ "first and last: " + deque);
    }
}
```

Deque (*double-ended queue*) adalah Collection yang elemennya dapat diakses dari HEAD dan TAIL.

COLLECTION INTERFACE: MAP

```
import java.util.*;

public class Freq {
    public static void main(String[] args) {
        Map<String, Integer> m = new HashMap<String, Integer>();
        // Initialize frequency table from command line
        for (String a : args) {
            Integer freq = m.get(a);
            m.put(a, (freq == null) ? 1 : freq + 1);
        }
        System.out.println(m.size() + " distinct words:");
        System.out.println(m);
    }
}
```

Map adalah Collection yang memetakan suatu *key* ke *value*.

Implementasi Map:

- HashMap
- TreeMap
- LinkedHashMap

JAVA STREAM API

- **Java Stream API** memungkinkan pengguna untuk melakukan operasi secara *functional-style*
- Secara umum terdapat empat proses:
  - filter
  - map
  - Reduce
  - forEach

CONTOH PEMAKAIAN JAVA STREAM API

```java
// Contoh penggunaan stream API reduce

import java.util.*;

class ReduceMain {
  public static void main(String[] args) {
      List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6);

      // Total semua elemen dalam list
      int result = numbers
      .stream()
      .reduce(0, (subtotal, element) -> subtotal + element);

      // Hasilnya 21
      System.out.println(result);
  }
}
```

https://docs.oracle.com/javase/tutorial/collections/streams/reduction.html

CONTOH PEMAKAIAN JAVA STREAM API

```java
// Contoh penggunaan stream API forEach

import java.util.*;

class ForEachMain {
  public static void main(String[] args) {
      List<String> names = Arrays.asList("Larry", "Steve", "James");

      // Mencetak setiap nama dalam names
      names.forEach(System.out::println);

      // Keluarannya:
      // Larry
      // Steve
      // James
  }
}
```

https://mkyong.com/java8/java-8-foreach-examples/