



Tutorial 5

Praktikum Pemrograman Berbasis Objek

Asisten IF2210 2022/2023

Outline

1. Review Praktikum 4
2. Java API
3. Generic

Generics

Generics in Java

- Generic pada Java dapat dibatasi oleh suatu kelas atau interface, misal:
 - **<T extends Number>**
 - **<E extends Cloneable & Serializable>**
- Namun, Java tidak dapat menyatakan non-type parameter seperti pada C++.

Ingat **template<class T, int N>** di C++?

Generic Method : Max Element of Array

```
// 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;
}
```

```
// 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 saja (int, float, dll.)

Generic Method : Max Element of Array

```
// 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;
}
```

```
// 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 java abstract class `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, specify manual  
        Integer max2 = e.max_elmt(arr, 3); // OK, automatically infer  
    }  
}
```

Generic Class: Stack

```
// 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
```

```
// Stack.java
public class Stack<T> {
    // size tidak dideklarasikan karena
    // sudah tersedia dari ArrayList
    // lewat method size()
    private ArrayList<T> data;

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


Use Case?

- Generic pada Java dapat digunakan untuk membuat relasi “is-a”
 - Cat **is-an** Animal
 - Apple **is-a** Fruit
 - T **is-a** Number

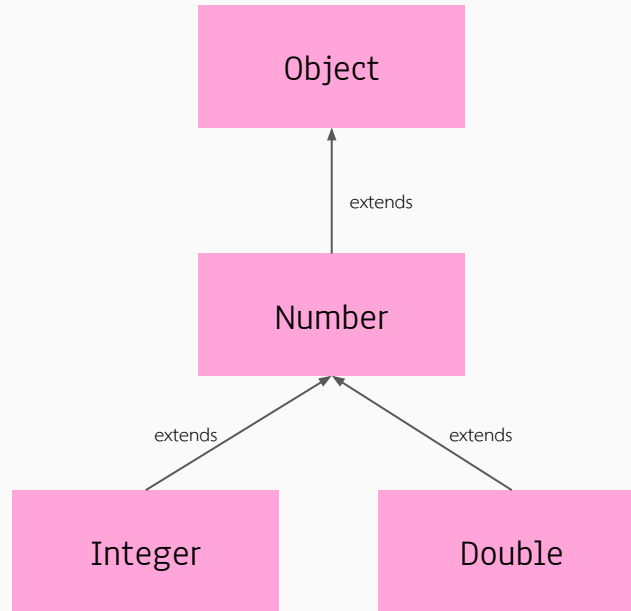
```
public void add(Number n1, Number n2) { /* ... */ }  
add(new Integer(10), new Double(20.0)); // OK, Integer & Double is-a Number  
add(new String("michel"), new String("faaaaaaaaang")); // String is NOT a Number
```

Wildcard



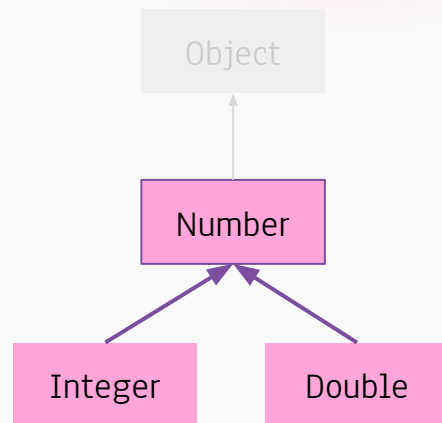
- Wildcard adalah representasi suatu object yang *unknown* (tidak diketahui kelasnya)
- Wildcard dalam Java direpresentasikan dalam “?”
- Kenapa tidak pakai <Object>? Cuman <Object> yang bisa masuk ke <Object>, tapi kalau <?> semua bisa masuk <Object> <Number> <ClassA> etc.
- Wildcard tidak bisa berdiri sendiri, melainkan hanya menjadi simbol untuk membuat batasan.
- Terdapat dua jenis batasan:
 - Unbounded (<? >)
 - Bounded
 - Upper-bounded (<? extends Number>)
 - Lower-bounded (<? super Integer>)

Hierarki Class Number



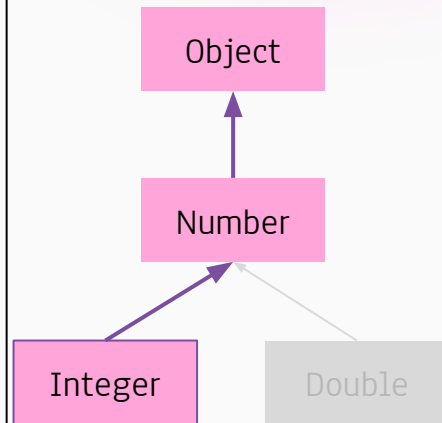
Upper Bounded Wildcard

```
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<>();  
        l2.add(new Integer(2));  
  
        ArrayList<Number> l3 = new ArrayList<>();  
        l3.add(new Double(3.0));  
  
        ArrayList<Double> l4 = new ArrayList<>();  
        l4.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  
    }  
}
```



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<>();  
        l2.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  
    }  
}
```

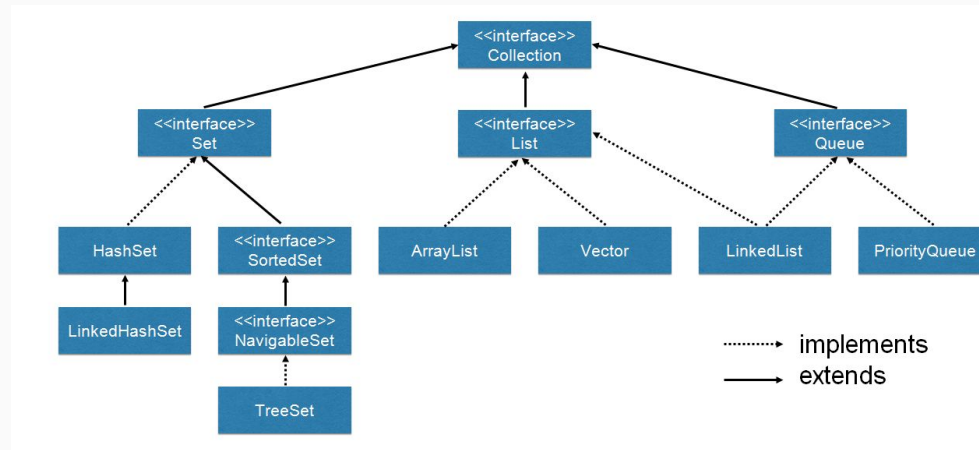


GENERIC TYPE	WILDCARD
Dapat memiliki >1 bound	Hanya dapat memiliki satu bound
Tidak ada lower bound, hanya upper bound (extends)	Ada lower bound (super) dan upper bound (extends)
<p>Dapat digunakan ketika meng-enforce tipe yang berhubungan</p> <pre>public static <T extends Number> void copy(List<T> dest, List<T> src)</pre>	<p>Tidak dapat digunakan untuk meng-enforce tipe yang berhubungan</p> <pre>public static void copy(List<?> dest, List<?> src) // tipe isi List dest dan src dapat berbeda</pre>
<p>Tipe dapat digunakan di-refer kembali pada method body</p> <pre>public <T extends Number> Map<T, String> convertToMap(ArrayList<T> list) { Map<T, String> names = new HashMap<T, String>(); ... }</pre>	<p>Tipe tidak dapat digunakan di-refer kembali pada method body</p>

Java API

Collection

- Terdapat Collection interface dalam Java API yang dapat digunakan untuk mengimplementasikan sebuah kumpulan data.



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

Queue

```
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.

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.

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

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

Apa itu Java Stream API

- Java Stream API memungkinkan pengguna untuk melakukan operasi secara *functional-style*.
- Pandanglah setiap item dalam collection sebagai sebagai aliran/*stream* data yang ingin diproses!
- Secara umum terdapat empat proses:
 - **filter()**
 - **map()**
 - **reduce()**
 - **forEach()**
- Dapat memanfaatkan Java Lambda Expression.

`(x, y) -> x + y`

Reduce

```
// 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);
    }
}
```


For Each

```
// 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
    }
}
```

Use Case Lainnya?

Misalkan kalian memiliki `List<String>` berisi beberapa string, bagaimana cara mendapatkan:

1. `List<Boolean>` yang akan bernilai `true` bila panjang string $> N$?
2. `List<String>` yang berisi string dengan panjang string $> N$?
3. Sebuah variabel yang bernilai jumlah string dengan panjang string $> N$?



Sekian.

Ditunggu praktikum dan tutorial berikutnya.
