



IF2032

Pemrograman Berorientasi Objek Java Generic

Yani Widayani (yani@stei.itb.ac.id)

Achmad Imam Kistijantoro (imam@informatika.org)

April 2009

Informatika – STEI – ITB

Sumber: *Slide Hananto W. Informatika – ITB 2006*



Objectives

- Di akhir session, peserta diharapkan mampu untuk:
 - Memahami konsep generic
 - Menggunakan kelas-kelas generik dalam java Collections API
 - Membuat implementasi kelas generik



Generic Java

- Generik dimasukkan pada Java untuk meningkatkan performansi pada kelas yang bersifat umum, misalnya kelas untuk menangani koleksi
 - menghindari keharusan melakukan casting tipe objek



Contoh: Pada Java < JDK 5

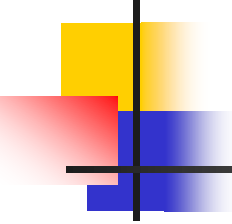
```
public interface List extends Collection {  
    Object get(int index);  
    Object set(int index, Object element);  
    boolean add(Object element);  
    void add(int index, Object element);  
    Object remove(int index);  
    boolean addAll(int index, Collection c);  
    int indexOf(Object o);  
    int lastIndexOf(Object o);  
    Iteration ListIterator listIterator();  
}
```



Contoh

```
List myList = new ArrayList();  
myList.add( new Person("amir") );  
Person p = (Person) myList.get( 0 );
```

- Downcast harus dilakukan saat runtime
 - performance cost
 - maintenance cost: pemeriksaan tidak dapat dilakukan oleh kompilator



Implementasi List Generic pada Java \geq JDK 5

```
public interface List<E> extends Collection<E> {  
    E get(int index);  
    E set(int index, E element);  
    boolean add(E element);  
    void add(int index, E element);  
    E remove(int index);  
    boolean addAll(int index,  
                   Collection<? extends E> c);  
    int indexOf(Object o);  
    int lastIndexOf(Object o);  
    ListIterator<E> listIterator();  
}
```



Contoh penggunaan

```
List<Person> myList = new ArrayList<Person>();  
myList.add( new Person("amir" ) );  
Person p = myList.get( 0 );
```



Generic pada Java

- Kelas generik pada java diimplementasikan sebagai parameter tipe
- hasil kompilasi kode kelas generik tetap hanya satu kelas, dengan parameter tipe yang diganti dengan tipe riil pada saat runtime
- pada C++, kompilasi menghasilkan kelas yang berbeda untuk setiap tipe generik



Sintaks Generik

- Mendefinisikan kelas & interface:
 - `class>NamaKelas<TipeGenerik> { ... }`
 - `interface>NamaInterface<TipeGenerik> { ... }`
 - `class>NamaKelas<TipeGenerik1, TipeGenerik2> { ... }`
 - `interface>NamaInterface<TipeGenerik1, TipeGenerik2> { ... }`
- Nama untuk tipe generik sebaiknya menggunakan sebuah karakter huruf besar, misalnya E atau T
- Kita dapat mendefinisikan tipe generik lebih dari satu



Sintaks Generik

- Kelas yang dapat digunakan untuk membuat kelas baru dengan menggunakan template kelas generik.

```
public class MyList<E> {  
    MyList() { ... }  
    public void add(E o) { ... }  
    public E get(int idx) { ... }  
}
```

```
MyList<String> ls = new MyList<String>();
```



Contoh ArrayList

```
public class ArrayList<E> {  
    public ArrayList() { ...}  
    boolean add(E o) { ... }  
    boolean addAll(Collection<? extends E> c) {  
        ... }  
}
```

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

HW/Java Thread

ITTA-7

```
Ar.add("satu");
```



Contoh ArrayList

```
class Student extends Person;  
ArrayList<Person> a1 = new ArrayList<Person>();  
ArrayList<Student> a2 = new ArrayList<Student>();  
Person p = new Person();  
Student s = new Student();  
a1.add( s );  
a2.add( p ); // tidak boleh  
  
a1 = new ArrayList<Student>(); // ??
```



Contoh ArrayList

- boolean addAll(Collection<? extends E> c)

class Student extends Person ...

```
ArrayList<Person> a1 = new ArrayList<Person>();
```

```
ArrayList<Person> a2 = new ArrayList<Person>()
```

```
ArrayList<Student> a3 = new ArrayList<Student>();
```

```
a1.addAll(a2);
```

```
a1.addAll(a3); boleh
```

```
a3.addAll(a1); // tidak boleh
```



Method generik

- sintaks:

`Modifier <ParameterType> ReturnType MethodName(..)`

- contoh:

```
class C {  
    public <T> List<T> sebuahMethod(T o) { ... }  
}
```

- Method generik digunakan:

- untuk menyatakan hubungan tipe antar parameter sebuah method
- untuk menyatakan tipe generik yang tidak berkaitan dengan parameter tipe yang dimiliki kelas

- Saat pemanggilan, tanda generik dapat dihilangkan, karena tipe generik dapat disimpulkan dari parameter yang digunakan:

```
C c = new C();  
c.<String>sebuahMethod("string");  
c.sebuahMethod("string");
```



Implementasi interface generik

```
public interface Comparator<T> {  
    int compare(T o1, T o2);  
    boolean equals(Object obj);  
}
```

```
Class MyComparator implements  
    Comparator<Person> {  
    int compare(Person p1, Person p2){ ...}  
    boolean equals(Object o) {... }  
}
```



contoh interface generic

- Interface Generik dideklarasikan seperti kelas Generik:

```
public interface List<E> {  
    void add(E x);  
    Iterator<E> iterator();  
}
```

- Dan diimplementasikan seperti ini:

```
public class MyList<E> implements List<E> {  
    void add(E x) { /*implementasi Add*/}  
    Iterator<E> iterator() { }  
}
```




Relasi subtype antar kelas Generik

- Contoh:

```
class Person { String nama; }  
class Student extends Person { String nim; }  
...  
ArrayList<Person> a1 = new ArrayList<Person>();  
ArrayList<Student> a2 = new ArrayList<Student>();  
Person p = new Person();  
Student s = new Student();  
a1.add( s );  
a2.add( p ); // tidak boleh
```

```
a1 = new ArrayList<Student>(); // tidak boleh
```

- `ArrayList<Student>` bukan subtype dari `ArrayList<Person>`, meskipun `Student` adalah subtype dari `Person`



wildcard

- wildcard digunakan untuk menyatakan sebuah variabel yang dapat menerima tipe generik apa saja
- contoh:

```
ArrayList<?> a3 = new  
ArrayList<Student>;
```

- wildcard dapat dibatasi, dengan memberi syarat bahwa tipe yang digunakan harus diturunkan dari kelas tertentu

```
ArrayList<? extends Person> a3 =  
    new ArrayList<Student>;
```



contoh

```
public void tulisNama(List<?> list) {  
    for( Object o : list) {  
        Person p = (Person) o;  
        System.out.println(  p.nama );  
    }  
}
```

```
public void tulisNama(List<? extends Person>  
    list) {  
    for( Person p : list) {  
        System.out.println(  p.nama );  
    }  
}
```