Bahasa Pemrograman Java
J a v a G e n e r i k

Saiful Akbar

Kontributor slide:
Achmad Imam Kistijantoro, Saiful Akbar, Yani Widyani, Hananto W.

Objectives

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

Ringkasan

- Mengenal Java Generik
- Tipe/Kelas Generik
- Method Generik
- Subtyping
- Bounded Type Parameters (Wildcards)
- Kelas/Interface Generik pada Java Collections API

Mengenal Java Generik

Problems


```
class StackItem {
     private Integer item = 0;
     private StackItem previous;
     public Integer getItem() { return item;}
     public void setItem(Integer x) { item = x; }
     ……
}
class Stack {
     private StackItem top;
     private int size;
     public Integer pop() {
          ……
     }
     public void push(Integer x) {
          ……
     }
     public int size() {……}
     public boolean empty() {……}
     public boolean full() {……}
     public Stack() {……}
}
```

Perhatikan representasi stack dalam kode ini agak berbeda dengan yang biasa dibahas pada kuliah

What about Stack of String, Stack of Float?

Solution 1

```
class StackItem {
    private Object item = 0;
    private StackItem previous;
    public Object getItem() { return item;}
    public void setItem(Object x) { item = x; }
    ……
}
class Stack {
    private StackItem top;
    private int size;
    public Object pop() {
        ……
    }
    public void push(Object x) {
        ……
    }
    public int size() {……}
    public boolean empty() {……}
    public boolean full() {……}
    public Stack() {……}
}
```

```
class StackApp {
  public static void main(String args[]) {
    int numberOfItems;
    numberOfItems = Integer.parseInt(args[0]);
    Stack sInt = new Stack();
    for (int i = 0; i < numberOfItems; i++)
      sInt.push(new Integer(i));
    Stack sStr = new Stack();
      String str1 = "Testing ";
      for (int i = 0; i < numberOfItems; i++)
        sStr.push(str1 + i);
  }
}
```

Solution 2, Run-time Type Identification (RTTI)

```
class StackApp {
  public static void main(String args[]) {
    int numberOfItems;
    numberOfItems = Integer.parseInt(args[0]);

    Stack s = new Stack();
    for (int i = 0; i < numberOfItems; i++)
      s.push(new Integer(i));

    String str1 = "Testing ";
    for (int i = 0; i < numberOfItems; i++)
      s.push(str1 + i);
  }
}
```

The stack can store not only Integer but also String. Often undesirable!

```
class InvalidTypeException extends Exception {}

class StackApp {
  private Stack s = new Stack();

  void pushInteger(Object o)
            throws InvalidTypeException {
    if (o instanceof Integer) s.push(o);
    else throw new InvalidTypeException();
  }

  public static void main(String args[]) {
    int numberOfItems;
    numberOfItems = Integer.parseInt(args[0]);
    StackApp sapp = new StackApp();
    try {
      for (int i = 0; i < numberOfItems; i++)
        sapp.pushInteger(new Integer(i));
        String str1 = "Testing ";
        sapp.pushInteger(str1 + 0);
    } catch (InvalidTypeException e) {
        System.out.println("WrongType pushed");
    }
  }
}
```

Clumsy, relies on programmers' diligence, not detectable during compile time

Solusi: Generic Type (preferable!)

```
class StackItem<E>{
  private E item = null;
  private StackItem<E> previous;
  public E getItem() { return item;}
  public void setItem(E x) { item = x; }
  public StackItem<E> getPrevious() { return previous; }
  public void setPrevious(StackItem<E> p) { previous = p; }
  StackItem() { previous = null; }
}
class Stack<E>{
  private StackItem<E> top, temp;
  private int size=10;
  public E pop() {
    E x = null;
    if (empty()) System.err.println("stack underflow");
    else {
      x = top.getItem();
      top = top.getPrevious();
      size = size - 1;
    }
    return x;
  }
  public void push(E x) {
    if (full()) System.err.println("stack overflow");
    else {
```

Stack <E>, Map <K,V>

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
- Keuntungan generik:
  - Meningkatkan expressive power
  - Meningkatkan type safety
  - Mengeksplisitkan parameter tipe dan mengimplisitkan type casting

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>();
```

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

Generic Type

Generic Type

```
public class Box<T> {  //T adalah formal type parameter
    // T stands for "Type"
    private T t;
    public void add(T t) {
       this.t = t;
    }
    public T get() {
       return t;
    }
}
```

```
 //generic type invocation, atau parameterized type
 Box<Integer> integerBox = new Box<Integer>();
//alternatif lain: diamond, JSE 7
Box<Integer> integerBox = new Box<>()

integerBox.add("10"); //error
```

Konvensi Penamaan Tipe

- E - Element (used extensively by the Java Collections Framework)
- K - Key
- N - Number
- T - Type
- V - Value
- S,U,V etc. - 2nd, 3rd, 4th types

Generic Methods

Method generik

sintaks:

```
Modifier <ParameterTipe> 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");
```

Generic Method


```
public class GenApp {
  private static <T> void printarray(T[] a){
    for (Object o : a)
      System.out.println (o);
  }
  public static void main(String args[]) {
    Integer iarr[] = new Integer[3];
    iarr[0] = new Integer(10);
    iarr[1] = new Integer(20);
    iarr[2] = new Integer(30);
    printarray(iarr);

    Float farr[] = new Float[3];
    farr[0] = new Float(48.0);
    farr[1] = new Float(59.0);
    farr[2] = new Float(67.0);
    printarray(farr);
  }
}
```

```
// "for-each"
for (Object o : a)
  System.out.println (o);

for (int i = 0; i < a.length; i++)
  System.out.println(a[i]);
```

Generic Methods – Contoh Lain

```
public static <U> void fillBoxes(U u,
          List<Box<U>> boxes) {
    for (Box<U> box : boxes) {
        box.add(u);
    }
}
```

```
 Crayon red = ...;
List<Box<Crayon>> crayonBoxes = ...;

Box.<Crayon>fillBoxes(red, crayonBoxes);
// alternatif, type inference
// compiler infers that U is Crayon
Box.fillBoxes(red, crayonBoxes);
```

Subtyping

Relasi Subtyping


```
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 karena reference ke student
          //(descendant) diacukan ke person (parent)

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

//ArrayList<Student> bukan subtipe dari ArrayList<Person>, meskipun
//Student adalah subtipe dari Person

Bounded Type Parameters (Wildcards)

Bounded Type Parameters?

- Membatasi tipe yang dapat di-*pass* ke sebuah parameter tipe
- Contoh:
  - Sebuah method yang melakukan operasi aritmatika, hanya dapat di pass dengan *Number* dan turunannya

```
 //the upper bound of a type wildcard
 public static <U extends Number> void Increment(U u) {
        ........
 }
//untuk memberikan spesifikasi tambahan interface
//myInterface
  public static <U extends Number & myInterface >
         void Increment(U u) { ... }
```

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>;
```

atau merupakan parent dari kelas tertentu

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

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 );
  }
}
```

Kelas/Interface Generik dalam Java Collection API

Contoh: Collection pada Java < JDK 5

```
public interface Collection {
    boolean     add(Object o)
    boolean     addAll(Collection c)
    void    clear()
    boolean     contains(Object o)
    boolean     containsAll(Collection c)
    boolean     equals(Object o)
    int     hashCode()
    boolean     isEmpty()
    Iterator    iterator()
    boolean     remove(Object o)
    boolean     removeAll(Collection c)
    boolean     retainAll(Collection c)
    int     size()
    Object[]    toArray()
    Object[]    toArray(Object[] a)

}
```

Contoh: List 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);
    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 >= 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 );
```

Contoh-contoh

Contoh ArrayList


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

ArrayList<String> ar = new
  ArrayList<String>();
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, karena add(E o)
             //berarti khusus untuk
             //objek berkelas "E"
             //atau turunan "E"

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
```

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() {  }
}
```