Praktikum Generics

Yuliana Setiowati
Politeknik Elektronika Negeri Surabaya

Praktikum 1

- Bagaimana ouput di bawah ini ?
- Tambahkan data lagi pada object integerBox dengan data object Integer, dan tambahkan data lagi pada object stringBox dengan data String. Bagaimana outputnya ?
- Tambahkan data lagi pada object integerBox dengan data **selain** object Integer, dan tambahkan data lagi pada object stringBox dengan data **selain** String. Bagaimana outputnya ? Jelaskan !

```
public class Box<T> {

  private T t;

  public void add(T t) {
    this.t = t;
  }

  public T get() {
    return t;
  }

  public static void main(String[] args) {
     Box<Integer> integerBox = new Box<Integer>();
     Box<String> stringBox = new Box<String>();

     integerBox.add(new Integer(10));
     stringBox.add(new String("Hello World"));

     System.out.printf("Integer Value :%d\n\n", integerBox.get());
     System.out.printf("String Value :%s\n", stringBox.get());
  }
}
```

Praktikum 2 : Class Generic dengan Dua Type Parameter

```
class TwoGen<T, V> {
  T ob1;

  V ob2;

  TwoGen(T o1, V o2) {
    ob1 = o1;
    ob2 = o2;
  }

  void showTypes() {
    System.out.println("Type of T is " + ob1.getClass().getName());

    System.out.println("Type of V is " + ob2.getClass().getName());
  }

  T getob1() {
    return ob1;
  }

  V getob2() {
    return ob2;
  }
}
```

Bagaimana output program dibawah ini? Jelaskan mengenai penggunaan generics di aplikasi ini !

```
public class MainClass {
  public static void main(String args[]) {
    TwoGen<Integer, String> tgObj = new TwoGen<Integer, String>(88, "Generics");
    tgObj.showTypes();

    int v = tgObj.getob1();
    System.out.println("value: " + v);

    String str = tgObj.getob2();
    System.out.println("value: " + str);
  }
}
```

Praktikum 3 - **Generic Methods**

- Bagaimana output di program ?
- Buatlah program yang sama dengan program disamping tapi tidak menggunakan generics! Jelaskan !

```
3
4  public class GenericMethodTest
5  {
6     // generic method printArray
7     public static < E > void printArray( E[] inputArray )
8     {
9        // display array elements
10       for ( E element : inputArray )
11          System.out.printf( "%s ", element );
12
13       System.out.println();
14    } // end method printArray
15
16    public static void main( String args[] )
17    {
18       // create arrays of Integer, Double and Character
19       Integer[] intArray = { 1, 2, 3, 4, 5 };
20       Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
21       Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };
22
23       System.out.println( "Array integerArray contains:" );
24       printArray( integerArray ); // pass an Integer array
25       System.out.println( "\nArray doubleArray contains:" );
26       printArray( doubleArray ); // pass a Double array
27       System.out.println( "\nArray characterArray contains:" );
28       printArray( characterArray ); // pass a Character array
29    } // end main
30 } // end class GenericMethodTest
```

Praktikum 4 : List

- Bagaimana output program dibawah ini?
- Jelaskan mengenai penggunaan generics di aplikasi ini !

```
1  // Fig. 19.3: CollectionTest.java
2  // Using the Collection interface.
3  import java.util.List;
4  import java.util.ArrayList;
5  import java.util.Collection;
6  import java.util.Iterator;
7
8  public class CollectionTest
9  {
10    private static final String[] colors =
11       { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
12    private static final String[] removeColors =
13       { "RED", "WHITE", "BLUE" };
14
15    // create ArrayList, add Colors to it and manipulate it
16    public CollectionTest()
17    {
18       List< String > list = new ArrayList< String >();
19       List< String > removeList = new ArrayList< String >();
20
21       // add elements in colors array to list
22       for ( String color : colors )
23          list.add( color );
24
25       // add elements in removeColors to removeList
26       for ( String color : removeColors )
27          removeList.add( color );
28
29       System.out.println( "ArrayList: " );
30
```

```
31          // output list contents
32          for ( int count = 0; count < list.size(); count++ )
33             System.out.printf( "%s ", list.get( count ) );
34
35          // remove colors contained in removeList
36          removeColors( list, removeList );
37
38          System.out.println( "\n\nArrayList after calling removeColors: " );
39
40          // output list contents
41          for ( String color : list )
42             System.out.printf( "%s ", color );
43       } // end CollectionTest constructor
44
45       // remove colors specified in collection2 from collection1
46       private void removeColors(
47          Collection< String > collection1, Collection< String > collection2 )
48       {
49          // get iterator
50          Iterator< String > iterator = collection1.iterator();
51
52          // loop while collection has items
53          while ( iterator.hasNext() )
54
55             if ( collection2.contains( iterator.next() ) )
56                iterator.remove(); // remove current Color
57       } // end method removeColors
58
59       public static void main( String args[] )
60       {
61          new CollectionTest();
62       } // end main
63    } // end class CollectionTest
```

Praktikum 4 : List

- Bagaimana output program dibawah ini?
- Jelaskan mengenai penggunaan generics di aplikasi ini !

```
1  // Fig. 19.4: ListTest.java
2  // Using LinkLists.
3  import java.util.List;
4  import java.util.LinkedList;
5  import java.util.ListIterator;
6
7  public class ListTest
8  {
9     private static final String colors[] = { "black", "yellow",
10       "green", "blue", "violet", "silver" };
11    private static final String colors2[] = { "gold", "white",
12       "brown", "blue", "gray", "silver" };
13
14    // set up and manipulate LinkedList objects
15    public ListTest()
16    {
17       List< String > list1 = new LinkedList< String >();
18       List< String > list2 = new LinkedList< String >();
19
20       // add elements to list link
21       for ( String color : colors )
22          list1.add( color );
23
24       // add elements to list link2
25       for ( String color : colors2 )
26          list2.add( color );
27
28       list1.addAll( list2 ); // concatenate lists
29       list2 = null; // release resources
30       printList( list1 ); // print list1 elements
31
32       convertToUppercaseStrings( list1 ); // convert to upper case string
33       printList( list1 ); // print list1 elements
34
35       System.out.print( "\nDeleting elements 4 to 6..." );
36       removeItems( list1, 4, 7 ); // remove items 4-7 from list
37       printList( list1 ); // print list1 elements
38       printReversedList( list1 ); // print list in reverse order
39    } // end ListTest constructor
40
```

```
41     // output List contents
42     public void printList( List< String > list )
43     {
44        System.out.println( "\nlist: " );
45
46        for ( String color : list )
47           System.out.printf( "%s ", color );
48
49        System.out.println();
50     } // end method printList
51
52     // locate String objects and convert to uppercase
53     private void convertToUppercaseStrings( List< String > list )
54     {
55        ListIterator< String > iterator = list.listIterator();
56
57        while ( iterator.hasNext() )
58        {
59           String color = iterator.next();  // get item
60           iterator.set( color.toUpperCase() ); // convert to upper case
61        } // end while
62     } // end method convertToUppercaseStrings
63
64     // obtain sublist and use clear method to delete sublist items
65     private void removeItems( List< String > list, int start, int end )
66     {
67        list.subList( start, end ).clear();  // remove items
68     } // end method removeItems
69
70     // print reversed list
71     private void printReversedList( List< String > list )
72     {
73        ListIterator< String > iterator = list.listIterator( list.size() );
74
75        System.out.println( "\nReversed List:" );
76
77        // print list in reverse order
78        while ( iterator.hasPrevious() )
79           System.out.printf( "%s ", iterator.previous() );
80     } // end method printReversedList
81
82     public static void main( String args[] )
83     {
84        new ListTest();
85     } // end main
86  } // end class ListTest
```

Praktikum 5 : Raw Types

```
// Demonstrate a raw type.
class Gen<T> {
  T ob;

  Gen(T o) {
    ob = o;
  }

  T getob() {
    return ob;
  }
}
```

```
public class MainClass {
  public static void main(String args[]) {
    Gen<Integer> iOb = new Gen<Integer>(88);
    Gen<String> strOb = new Gen<String>("Generics Test");

    Gen raw = new Gen(new Double(98.6));

    // Cast here is necessary because type is unknown.
    double d = (Double) raw.getob();
    System.out.println("value: " + d);

    strOb = raw; // OK, but potentially wrong
    String str = strOb.getob();

  // This assignment also overrides type safety.
  raw = iOb; // OK, but potentially wrong
  d = (Double) raw.getob();

  }
}
```

Praktikum 6 : Generics and Collections: ArrayList

```
import java.util.ArrayList;
import java.util.Iterator;

public class MainClass {
  public static void main(String args[]) {
    ArrayList<String> list = new ArrayList<String>();
    list.add("one");
    list.add("two");
    list.add("three");
    list.add("four");

    Iterator<String> itr = list.iterator();

    while(itr.hasNext()) {
      String str = itr.next();

      System.out.println(str + " is " + str.length() + " chars long.");
    }
  }
}
```

```
one is 3 chars long.
two is 3 chars long.
three is 5 chars long.
four is 4 chars long.
```

Praktikum 7 : Array

```
class Person {
  private String lastName;
  private String firstName;

  private int age;

  public Person(String last, String first, int a) {
    lastName = last;
    firstName = first;
    age = a;
  }

  public String toString() {
    return "Last name: " + lastName + " First name: " + firstName + " Age: " + age;
  }

  public String getLast() {
    return lastName;
  }
}
```

```
public class MainClass {
  public static void main(String[] args) {
    Person[] persons = new Person[10];

    for(int i=0;i<persons.length;i++){
      persons[i] = new Person("A","B",10);
    }

    for(Person p: persons){
      System.out.println(p.getLast());

    }
  }
}
```

Menggunakan Interface Generic Comparable

```
import java.util.Arrays;

class Person implements Comparable<Person> {
  public Person(String firstName, String surname) {
    this.firstName = firstName;
    this.surname = surname;
  }

  public String toString() {
    return firstName + " " + surname;
  }

  public int compareTo(Person person) {
    int result = surname.compareTo(person.surname);
    return result == 0 ? firstName.compareTo(((Person) person).firstName) : result;
  }

  private String firstName;

  private String surname;
}
```

```
public class MainClass {
  public static void main(String[] args) {
    Person[] authors = {
        new Person("D", "S"),
        new Person("J", "G"),
        new Person("T", "C"),
        new Person("C", "S"),
        new Person("P", "C"),
        new Person("B", "B") };

    Arrays.sort(authors); // Sort using Comparable method

    System.out.println("\nOrder after sorting into ascending sequence:");
    for (Person author : authors) {
      System.out.println(author);
    }

    Person[] people = {
        new Person("C", "S"),
        new Person("N", "K"),
        new Person("T", "C"),
        new Person("C", "D") };
    int index = 0;
    System.out.println("\nIn search of authors:");

    for (Person person : people) {
      index = Arrays.binarySearch(authors, person);
      if (index >= 0) {
        System.out.println(person + " was found at index position " + index);
      } else {
        System.out.println(person + "was not found. Return value is " + index);
      }
    }
  }
```

Praktikum 8:Type parameter yang dibatasi

```
public class Box<T> {

    private T t;

    public void add(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }

    public <U extends Number> void inspect(U u){
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    }

    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.add(new Integer(10));
        integerBox.inspect("some text"); // error: this is still String!
    }
}
```

```
Box.java:21: <U>inspect(U) in Box<java.lang.Integer> cannot
  be applied to (java.lang.String)
                        integerBox.inspect("10");
                                   ^
1 error
```

Praktikum 9 : Generic Method

```
public class MainClass {
  static <T, V extends T> boolean isIn(T x, V[] y) {
    for (int i = 0; i < y.length; i++){
      if (x.equals(y[i])){
        return true;
      }
    }
    return false;
  }

  public static void main(String args[]) {
    Integer nums[] = { 1, 2, 3, 4, 5 };

    if (isIn(2, nums)){
      System.out.println("2 is in nums");
    }
    if (!isIn(7, nums)){
      System.out.println("7 is not in nums");
    }

    // Use isIn() on Strings.
    String strs[] = { "one", "two", "three", "four", "five" };

    if (isIn("two", strs))
      System.out.println("two is in strs");

    if (!isIn("seven", strs))
      System.out.println("seven is not in strs");

  }
}
```

Praktikum 10 : Generic Constructor

```
class GenericClass {
  private double val;

  <T extends Number> GenericClass(T arg) {
    val = arg.doubleValue();
  }

  void showValue() {
    System.out.println("val: " + val);
  }
}

public class MainClass {
  public static void main(String args[]) {

    GenericClass test = new GenericClass(100);
    GenericClass test2 = new GenericClass(123.5F);

    test.showValue();
    test2.showValue();
  }
}
```

```java
public class MainClass {
  // generic method printArray
  public static <E> void printArray(E[] inputArray) {
    // display array elements
    for (E element : inputArray)
      System.out.printf("%s ", element);

    System.out.println();
  }

  public static void main(String args[]) {
    // create arrays of Integer, Double and Character
    Integer[] integerArray = { 1, 2, 3, 4, 5, 6 };
    Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
    Character[] characterArray = { 'H', 'E', 'L', 'L', 'O' };

    System.out.println("Array integerArray contains:");
    printArray(integerArray); // pass an Integer array
    System.out.println("\nArray doubleArray contains:");
    printArray(doubleArray); // pass a Double array
    System.out.println("\nArray characterArray contains:");
    printArray(characterArray); // pass a Character array
  } // end main
}
```

Praktikum 11 :Type parameter yang dibatasi

```
class Stats<T extends Number> {
  T[] nums;

  Stats(T[] o) {
    nums = o;
  }

  double average() {
    double sum = 0.0;

    for(int i=0; i < nums.length; i++)
      sum += nums[i].doubleValue();

    return sum / nums.length;
  }
}

public class MainClass {
  public static void main(String args[]) {

    Integer inums[] = { 1, 2, 3, 4, 5 };
    Stats<Integer> iob = new Stats<Integer>(inums);
    double v = iob.average();
    System.out.println("iob average is " + v);

    Double dnums[] = { 1.1, 2.2, 3.3, 4.4, 5.5 };
    Stats<Double> dob = new Stats<Double>(dnums);
    double w = dob.average();
    System.out.println("dob average is " + w);
  }
}
```

Praktikum 12 : Menggunakan ? Wildcard

- Bagaimana penyelesaiannya ?
- Menggunakan ? Wildcard. List<?> berarti list dengan object tipe sembarang.

```
public static void doIt(List<?> l) {
}
```

```
package com.brainysoftware.jdk5.app16;
import java.util.ArrayList;
import java.util.List;

public class WildCardTest {

  public static void printList(List<?> list) {
    for (Object element : list) {
      System.out.println(element);
    }
  }
  public static void main(String[] args) {
    List<String> list1 = new ArrayList<String>();
    list1.add("Hello");
    list1.add("World");
    printList(list1);

    List<Integer> list2 = new ArrayList<Integer>();
    list2.add(100);
    list2.add(200);
    printList(list2);
  }
}
```

Praktikum 13 : Menggunakan ? Wildcard

```
// Use a wildcard.
class GenericStats<T extends Number> {
  T[] nums;

  GenericStats(T[] o) {
    nums = o;
  }

  double average() {
    double sum = 0.0;
     for(int i=0; i < nums.length; i++){
      sum += nums[i].doubleValue();
     }
    return sum / nums.length;
  }

  boolean sameAvg(GenericStats<?> ob) {
    if(average() == ob.average())
      return true;

    return false;
  }
}
```

Praktikum 14 : Menggunakan Bounded Wildcard dalam Method

```
package com.brainysoftware.jdk5.app16;
import java.util.ArrayList;
import java.util.List;
public class BoundedwildcardTest {
  public static double getAverage(List<? extends Number> numberList)
  {
    double total = 0.0;
    for (Number number : numberList)
      total += number.doublevalue();
    return total/numberList.size();
  }

  public static void main(String[] args) {
    List<Integer> integerList = new ArrayList<Integer>();
    integerList.add(3);
    integerList.add(30);
    integerList.add(300);
    System.out.println(getAverage(integerList)); // 111.0
    List<Double> doubleList = new ArrayList<Double>();
    doubleList.add(3.0);
    doubleList.add(33.0);
    System.out.println(getAverage(doubleList)); // 18.0
  }
}
```

Praktikum 15 : Bounded Wildcards

```
class Two {
  int x, y;

  Two(int a, int b) {
    x = a;
    y = b;
  }
}

class Three extends Two {
  int z;

  Three(int a, int b, int c) {
    super(a, b);
    z = c;
  }
}

class Four extends Three {
  int t;

  Four(int a, int b, int c, int d) {
    super(a, b, c);
    t = d;
  }
}
```

```
class Gen<T extends Two> {
  T[] coords;

  Gen(T[] o) {
    coords = o;
  }
}
```

Bounded Wildcards

```
public class MainClass {
  static void showTwo(Gen<?> c) {
    System.out.println("X Y Coordinates:");
    for (int i = 0; i < c.coords.length; i++)
      System.out.println(c.coords[i].x + " " + c.coords[i].y);
    System.out.println();
  }

  static void showThree(Gen<? extends Three> c) {
    System.out.println("X Y Z Coordinates:");
    for (int i = 0; i < c.coords.length; i++)
      System.out.println(c.coords[i].x + " " + c.coords[i].y + " " + c.coords[i].z);
    System.out.println();
  }

  static void showAll(Gen<? extends Four> c) {
    System.out.println("X Y Z T Coordinates:");
    for (int i = 0; i < c.coords.length; i++)
      System.out.println(c.coords[i].x + " " + c.coords[i].y + " " + c.coords[i].z + " "
          + c.coords[i].t);
    System.out.println();
  }
```

Bounded Wildcards

```
  public static void main(String args[]) {
    Two td[] = { new Two(0, 0), new Two(7, 9), new Two(18, 4), new Two(-1, -23) };

    Gen<Two> tdlocs = new Gen<Two>(td);

    System.out.println("Contents of tdlocs.");
    showTwo(tdlocs); // OK, is a TwoD

    Four fd[] = { new Four(1, 2, 3, 4), new Four(6, 8, 14, 8), new Four(22, 9, 4, 9),
        new Four(3, -2, -23, 17) };

    Gen<Four> fdlocs = new Gen<Four>(fd);

    System.out.println("Contents of fdlocs.");
    // These are all OK.
    showTwo(fdlocs);
    showThree(fdlocs);
    showAll(fdlocs);
  }
}
```