PRAKTIKUM 6

GENERIC 2

A. TUJUAN PEMBELAJARAN

1. Memahami mengenai konsep generic.
2. Memahami penggunaan tipe parameter yang dibatasi (Bounded Type Parameter).
3. Memahami penggunaan wildcard.

B. DASAR TEORI

Generics merupakan cara Java dalam melakukan generalisasi terhadap tipe data tanpa mengurangi kemampuan Java dalam menjaga keamanan penggunaan tipe data.

```
public class NonGen {
  Object ob;

  NonGen(Object o) {
    ob = o;
  }

  Object getob() {
    return ob;
  }
  void showType() {
    System.out.println("Type of ob is " +
                       ob.getClass().getName());
  }
}
```

Pada object Box, kita bisa memasukkan sembarang object karena parameter pada method `add()` adalah Class Object, tapi pada saat mengambil object tersebut harus diubah sesuai dengan tipe dari object tersebut.

```
public class NonGenDemo {
  public static void main(String args[]) {
    NonGen integerObject;
    integerObject = new NonGen(88);

    integerObject.showType();
```

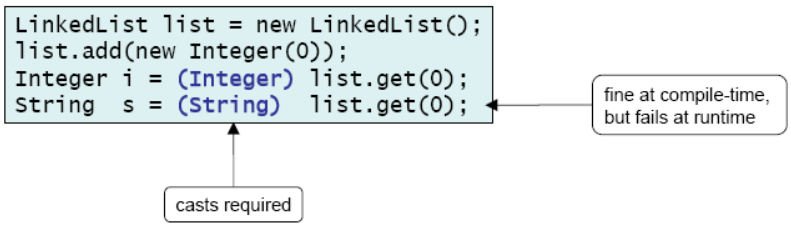
```
    int v = (Integer) integerObject.getob();
    System.out.println("value: " + v);


    NonGen strOb = new NonGen("10");
    strOb.showType();

    String str = (String) strOb.getob();
    System.out.println("value: " + str);

    //menyebabkan exception
    Integer i = (Integer) strOb.getob();
  }
}
```

Terjadi exception karena pada object strOb dimasukkan object 10 tapi dengan tipe String, tapi pada saat mengambil object, diubah menjadi tipe Integer. Tipe data tidak sesuai sehingga menyebabkan terjadinya exception.

Output :

```
Type of ob is java.lang.Integer
value: 88
Exception in thread "main" java.lang.ClassCastException:
java.lang.String cannot be cast to java.lang.Integer
Type of ob is java.lang.String
value: Non-Generics Test
        at com.NonGenDemo.main(NonGenDemo.java:30)
Java Result: 1
```

Permasalahan yang muncul dengan penyimpanan objek yang non homogeneous adalah memerlukan banyak casting. Tidak ada pengecekan pada saat kompile, kesalahan baru bisa terdeteksi pada saat runtime.



Cara mendeklarasikan Class Generics

Ubah class Box Non Generics menjadi class Box Generic. Pendeklarasian type generics dengan mengubah public class Box() menjadi public class Box <T>

– T biasanya disebut parameter type formal (formal type parameter)

– T adalah type parameter yang akan diganti dengan tipe sebenarnya (Type dari T bisa berupa class, interface atau tipe variabel lainnya).
– T adalah nama dari type parameter.

```
public class Gen<T> {
  T ob; // declare an object of type T

  Gen(T o) {
    ob = o;
  }

  T getob() {
    return ob;
  }

  void showType() {
    System.out.println("Type of T is " + ob.getClass().getName());
  }
}
```

```
public class GenDemo {
  public static void main(String args[]) {
    Gen<Integer> iOb;
    iOb = new Gen<Integer>(88);

    iOb.showType();

    int v = iOb.getob();
    System.out.println("value: " + v);

    System.out.println();

    Gen<String> strOb = new Gen<String>("Generics Test");

    String v2 = strOb.getob();
  }
}
```

Objek `iOb` adalah objek dari class Generic, menggunakan tipe data Integer, sehingga pada saat mengambil objek menggunakan method get(), tidak perlu proses casting.

Aturan Penamaan Type Parameter

Nama type parameter biasanya satu huruf dan huruf besar. Jenis nama tipe parameter yang sering digunakan :

– E - Element (biasanya digunakan untuk Collection Framework)

– K – Key
– N – Number
– T - Type
– V - Value
– S,U,V dll. - 2nd, 3rd, 4th types

Generics pada Method

Pada contoh sebelumnya, kita mendefinisikan type parameter pada level class.

Sebenarnya tipe variable ini juga dapat didefinisikan pada level method.

```
public class GenericMethodTest {
    public static<E> void printArray(E[] inputArray){
        for(E element : inputArray)
            System.out.printf("%s",element);
        System.out.println("");
    }
    public static void main(String[] args) {
        Integer[] intArray = {1,2,3,4,5} ;
        Double[] doubleArray = {1.1,2.2,3.3,4.4,5.5};
        Character[] charArray = {'J','A','V','A'};

        printArray(intArray);
        printArray(doubleArray);
        printArray(charArray);
    }
}
```

Type parameter yang dibatasi

Jika kita ingin memberikan batasan type yang diperbolehkan untuk dilewatkan ke type parameter. Contoh method dengan parameter number, hanya menerima object dari class Number dan subclass. Hal ini yang disebut *bounded type parameter*.

- Cara

  <U extends Number>
- Jika terdapat interface yang harus diimplementasikan gunakan &

  <U extends Number & MyInterface>

Menggunakan ? Wildcard

Jika kita mendeklarasikan sebuah List<aType>, isi List adalah object dengan tipe aType, maka kita bisa menyimpan di List object dengan tipe :

– Object dari aType
– Object dari subclass aType, jika aType adalah class
– Object dari class yang mengimplementasikan aType, jika aType adalah interface

Sedangkan List<?> berarti semua tipe data bisa masuk, terlalu luas bentuk lain :

```
List<? extends Number>
List<? extends T>
List<? super Integer>
```

C. TUGAS PENDAHULUAN

Buatlah review mengenai :

- permasalahan pada class non generic
- merubah bentuk class non generic menjadi class generic
- manfaat dari class generics
- generic pada collection.

D. PERCOBAAN

Percobaan 1 : 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 BoundsDemo {
```

```
  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);
  }
}
```

Percobaan 2 : Generic pada konstruktor, dengan type parameter dibatasi.

```
class GenCons {
  private double val;

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

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

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

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

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

Percobaan 3 : Menggunakan Interface Comparable Generic

```
interface MinMax<T extends Comparable<T>> {
  T min();
  T max();
}

class MyClass<T extends Comparable<T>> implements MinMax<T> {
  T[] vals;

  MyClass(T[] o) { vals = o; }
```

```
  public T min() {
    T v = vals[0];
    for(int i=1; i < vals.length; i++)
      if(vals[i].compareTo(v) < 0) v = vals[i];

    return v;
  }

  public T max() {
    T v = vals[0];

    for(int i=1; i < vals.length; i++)
      if(vals[i].compareTo(v) > 0) v = vals[i];

    return v;
  }
}

public class GenIFDemo {
  public static void main(String args[]) {
    Integer inums[] = {3, 6, 2, 8, 6 };
    Character chs[] = {'b', 'r', 'p', 'w' };

    MyClass<Integer> iob = new MyClass<Integer>(inums);
    MyClass<Character> cob = new MyClass<Character>(chs);

    System.out.println("Max value in inums: " + iob.max());
    System.out.println("Min value in inums: " + iob.min());

    System.out.println("Max value in chs: " + cob.max());
    System.out.println("Min value in chs: " + cob.min());
  }
}
```

Percobaan 4 : Menggunakan Unbounded Wildcard(1)

```
public class TestGeneric {

    public static void printList(List<?> list) {
        for (Object elem : list) {
            System.out.print(elem + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        List<Integer> li = Arrays.asList(1, 2, 3);
        List<String> ls = Arrays.asList("one", "two", "three");
        printList(li);
        printList(ls);
    }
}
```

Percobaan 5 : Menggunakan Unbounded Wildcard(2)

```
class Stats<T extends Number> {
  T[] nums; // array of Number or subclass

  // Pass the constructor a reference to
  // an array of type Number or subclass.
  Stats(T[] o) {
    nums = o;
  }

  // Return type double in all cases.
  double average() {
    double sum = 0.0;

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

    return sum / nums.length;
  }

  // Determine if two averages are the same.
  // Notice the use of the wildcard.
  boolean sameAvg(Stats<?> ob) {
    if(average() == ob.average())
      return true;

    return false;
  }
}

// Demonstrate wildcard.
public class WildCardDemo {
  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);

    Float fnums[] = { 1.0F, 2.0F, 3.0F, 4.0F, 5.0F };
    Stats<Float> fob = new Stats<Float>(fnums);
    double x = fob.average();
    System.out.println("fob average is " + x);

    // See which arrays have same average.
    System.out.print("Averages of iob and dob ");
    if(iob.sameAvg(dob))
      System.out.println("are the same.");
    else
```

```
      System.out.println("differ.");

    System.out.print("Averages of iob and fob ");
    if(iob.sameAvg(fob))
      System.out.println("are the same.");
    else
      System.out.println("differ.");
  }
}
```

Percobaan 6 : Menggunakan UpperBound Wildcard(2)

```
import java.util.Arrays;
import java.util.List;

public class TestGeneric2 {

    public static double sumOfList(List<? extends Number> list) {
        double s = 0.0;
        for (Number n : list) {
            s += n.doubleValue();
        }
        return s;
    }

    public static void main(String[] args) {
        List<Integer> li = Arrays.asList(1, 2, 3);
        System.out.println("sum = " + sumOfList(li));
        List<Double> ld = Arrays.asList(1.2, 2.3, 3.5);
        System.out.println("sum = " + sumOfList(ld));
    }
}
```

Percobaan 7 : Menggunakan UpperBound Wildcard(2)

```
class TwoD {
  int x, y;

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

```
// Three-dimensional coordinates.
class ThreeD extends TwoD {
  int z;

  ThreeD(int a, int b, int c) {
    super(a, b);
    z = c;
```

```
  }
}
```

```
// Four-dimensional coordinates.
class FourD extends ThreeD {
  int t;

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

```
// This class holds an array of coordinate objects.
class Coords<T extends TwoD> {
  T[] coords;

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

```
// Demonstrate a bounded wildcard.
public class BoundedWildcard {
  static void showXY(Coords<?> 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 showXYZ(Coords<? extends ThreeD> 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(Coords<? extends FourD> 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();
  }

  public static void main(String args[]) {
    TwoD td[] = {
```

```
      new TwoD(0, 0),
      new TwoD(7, 9),
      new TwoD(18, 4),
      new TwoD(-1, -23)
    };

    Coords<TwoD> tdlocs = new Coords<TwoD>(td);

    System.out.println("Contents of tdlocs.");
    showXY(tdlocs); // OK, is a TwoD
//  showXYZ(tdlocs); // Error, not a ThreeD
//  showAll(tdlocs); // Erorr, not a FourD

    // Now, create some FourD objects.
    FourD fd[] = {
      new FourD(1, 2, 3, 4),
      new FourD(6, 8, 14, 8),
      new FourD(22, 9, 4, 9),
      new FourD(3, -2, -23, 17)
    };

    Coords<FourD> fdlocs = new Coords<FourD>(fd);

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

Percobaan 8 : Menggunakan LowerBound Wildcard

```
public class BoxImpl<T> implements Box<T> {

    private T element;

    public T get() {
        return element;
    }

    public void put(T element) {
        this.element = element;
    }

}
```

```
public class TestBoxKu {
    public static void main(String[] args) {
        BoxKu<Integer> bIn = new BoxKu<Integer>(7);
        BoxKu<Number> bNum = new BoxKu<Number>(10);
```

```
        System.out.println(bIn.toString());
        System.out.println(bNum.toString());

        bIn.copyTo(bNum);

        System.out.println(bIn.toString());
        System.out.println(bNum.toString());
    }
}
```

E. LATIHAN

1. Apakah program di bawah ini dapat dikompile ? Jika tidak, jelaskan !

```
public static void print(List<? extends Number> list) {
    for (Number n : list)
        System.out.print(n + " ");
    System.out.println();
}
```

2. Buatlah method generic untuk mencari nilai maksimul pada range [begin, end] dari list.

```
public static <T extends Object & Comparable<? super T>>
        T max(List<? extends T> list, int begin, int end) {...}
```

3. Buatlah interface Box seperti dibawah ini! Dan buatlah class BoxImpl yang mengimplementasikan interface Box.

```
public interface Box<T> {
    public T get();
    public void put(T element);
    public void put(Box<T> box);
}
```

```
public class BoxImpl<T> implements Box<T> {
    private T element ;
    public T get() {
        //isilah
    }

    public void put(T element) {
        //isilah
    }

    public void put(Box<T> box) {
        //isilah
    }
```

```
}
```

Jelaskan mengapa program di bawah ini error !

```
public class TestBoxImpl {
    public static void main(String[] args) {
        Box<Number> nBox = new BoxImpl<Number>();
        Box<Integer> iBox = new BoxImpl<Integer>();

        nBox.put(iBox);     // ERROR
    }
}
```

Perbaiki interface Box seperti di bawah ini ! Jelaskan mengapa dengan perubahan seperti ini program yang sebelumnya error menjadi tidak error ?

```
public interface Box<T> {
    public T get();
    public void put(T element);
    public void put(Box<? extends T> box);
}
```

Tambahkan pada interface Box method **`containsSame`** dan interface **`EqualityComparator`**

```
public interface Box<T> {
    public T get();
    public void put(T element);
    public void put(Box<? extends T> box);

    boolean containsSame(Box<? extends T> other,
                         EqualityComparator<T> comparator);

    public interface EqualityComparator<T> {
        public boolean compare(T first, T second);
    }
}
```

Tambahkan pada class TestBoxImpl. Jelaskan mengapa EqualityComparator error ?

```
public class TestBoxImpl {
    public static EqualityComparator<T> sameObject = new
EqualityComparator<T>() {
        public boolean compare(T o1, T o2) {
            return o1 == o2;
        }
    }
    public static void main(String[] args) {
        Box<Number> nBox = new BoxImpl<Number>();
```

```
        Box<Integer> iBox = new BoxImpl<Integer>();

        //sebelumnya error menjadi OK
        nBox.put(iBox);

        boolean b = nBox.containsSame(iBox, sameObject);
    }
}
```

Ubahlah menjadi bentuk seperti di bawah ini ! Jelaskan mengapa method containsSame() masih error ?

```
public class TestBoxImpl {
    public static EqualityComparator<Object> sameObject = new
EqualityComparator<Object>() {

        public boolean compare(Object o1, Object o2) {
            return o1 == o2;
        }
    };
    public static void main(String[] args) {
        Box<Number> nBox = new BoxImpl<Number>();
        Box<Integer> iBox = new BoxImpl<Integer>();

        //sebelumnya error menjadi OK
        nBox.put(iBox);

        boolean b = nBox.containsSame(iBox, sameObject);
    }
}
```

Ubahlah interface Box menjadi seperti di bawah ini ! Ubah pula pada class BoxImpl

```
public interface Box<T> {
    public T get();
    public void put(T element);
    public void put(Box<? extends T> box);

    boolean containsSame(Box<? extends T> other,
                         EqualityComparator<? super T> comparator);

    public interface EqualityComparator<T> {
        public boolean compare(T first, T second);
    }
}
```

Lakukan cek pada method containsSame() apakah error ? Jelaskan !

F. LAPORAN RESMI

Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.