VERSION 1.0

SEPTEMBER 9, 2021

[STRUKTUR DATA]

MODUL 1, GENERICS

DISUSUN OLEH: - BELLA DWI MARDIANA
- LIDYA FANKKY OKTAVIA P.

DIAUDIT OLEH: - IR. GITA INDAH MARTHASARI, ST., M.KOM.
- DIDIH RIZKI CHANDRANEGARA, S.KOM., M.KOM.

PRESENTED BY: TIM LAB-IT

UNIVERSITAS MUHAMMADIYAH MALANG

[STRUKTUR DATA]

PERSIAPAN MATERI

Mahasiswa diharapkan mempelajari materi sebelum mengerjakan tugas, materi yang tercakup antara lain:

1. Class dan Object
2. Enkapsulasi
3. Array

TUJUAN

Mahasiswa mampu menguasai & menjelaskan konsep dari Struktur Data Generics

TARGET MODUL

Mahasiswa mampu memahami:

1. Menerapkan Array of Object
2. Generics Class
3. Generics Method
4. Wildcard
5. Enumerated Types

PERSIAPAN SOFTWARE/APLIKASI

1. Java Development Kit
2. Java Runtime Environment
3. IDE (Intellij IDEA, Eclipse, Netbeans, dll)

REFERENSI MATERI

Oracle iLearning Java Programming section 6-1 Generics

MATERI POKOK

Pengertian:

Generics merupakan sebuah cara yang digunakan untuk membuat tipe bersifat umum. Generics juga dilakukan untuk menambah stabilitas pada kode program sehingga dapat membantu mempermudah deteksi bug pada saat kompilasi.

Kenapa Menggunakan Generics?

Generics memberikan keamanan tipe pada saat kompilasi yang memungkinkan pemrogram mengetahui tipe yang tidak valid. Generics memungkinkan tipe (class dan interface) menjadi parameter ketika mendefinisikan suatu class, interface, dan method. Sama seperti parameter formal yang lebih akrab digunakan dalam deklarasi method, parameter memberikan cara untuk menggunakan kembali kode yang sama dengan input yang berbeda. Perbedaannya terdapat pada input yang dilakukan ke parameter formal adalah suatu nilai, sedangkan input untuk parameter adalah suatu tipe.

Contoh Perbedaan Dalam Pemanfaatan Generics:

Berikut adalah kode program yang dibuat tanpa menggunakan metode generics (metode yang dipakai adalah metode casting yang digunakan untuk menerima value dari variable):

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

Jika menggunakan metode generics:

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

Dengan metode generics, pemrogram dapat menerapkan algoritma generics yang bekerja pada koleksi dari berbagai jenis, dapat disesuaikan, aman dan lebih mudah dibaca. Tipe generics dapat dideklarasikan menggunakan angled brackets (tanda <>) dengan didalamnya diisi tipe pengembalian yang ingin didapatkan.

Contoh Deklarasi Class Menggunakan Metode Generics:

Penggunaan pada class ditandai dengan angled brackets (tanda <>) lalu dituliskan di dalamnya seperti class Cell berikut:

```
public class Cell<T> {
    private T t;
    public void set(T celldata){
        t = celldata;
    }
    public T get(){
        return t;
    }
}
```

Untuk pembuatan object dan menerima object yang menggunakan class dapat menggunakan cara sebagai berikut:

```java
public class Main {
    public static void main(String[] args) {
        Cell<Integer> integerCell = new Cell<Integer>();
        Cell<String> stringCell =new Cell<>();
        integerCell.set(1);
        stringCell.set("Test");
    }
}
```

Deklarasi generics dapat dilakukan dengan cara yang lebih ringkas seperti yang dapat dilihat pada class Main diatas, yaitu pada baris ke-3 new Cell<Integer> dan baris ke-4 diringkas menjadi new Cell<>, kedua penulisan tersebut benar semua.

Konvensi Penamaan Parameter Pada Generics:

Penamaan parameter dapat dibuat sesuai keinginan pemrogram, berikut penamaan parameter yang paling umum digunakan:

- E – Element (digunakan secara luas oleh Java Collections Framework)
- K – Key
- N – Number
- T – Type
- V – Value

Generics Method:

Generics method adalah metode yang digunakan dengan memanfaatkan tipe parameter mereka sendiri. Cara ini mirip dengan mendeklarasikan tipe generics, tetapi cakupan tipe parameter terbatas pada metode yang dideklarasikan. Method static, non-static, dan constructor class generics dapat menggunakan metode ini. Berikut adalah contoh penggunaan generics method:

```java
class Util {
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey()) &&
               p1.getValue().equals(p2.getValue());
    }
}

public class Pair<K, V> {

    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public void setKey(K key) { this.key = key; }
    public void setValue(V value) { this.value = value; }
    public K getKey()   { return key; }
    public V getValue() { return value; }
}
```

Pemanggilan dan pembuatan object dapat dilakukan pada class main seperti berikut:

```
public static void main(String[] args) {
    Pair<Integer, String> p1 = new Pair<>(1, "apple");
    Pair<Integer, String> p2 = new Pair<>(2, "pear");
    boolean same = Util.compare(p1, p2);
}
```

Wildcards:

Wildcard dituliskan dalam tanda tanya (?) yang memiliki arti tipe data belum diketahui. Wildcard dapat digunakan dalam berbagai situasi, seperti jenis parameter, field, variabel local atau tipe pengembalian. Dalam wildcard terdapat istilah yang disebut dengan upper bound wildcards, yaitu digunakan untuk memberi batasan pada variabel. Berikut adalah contoh penggunaannya:

```
public class Wildcard {
    public static void printList(List<?> list) {
        for (Object elm : list) {
            System.out.print(elm + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        List<Integer> dataInt = Arrays.asList(7, 9, 10);
        List<String> dataString = Arrays.asList("tujuh", "sembilan", "sepuluh");
        printList(dataInt);
        printList(dataString);
    }
}
```

Enumerated Types:

Enumerated types adalah tipe data khusus yang memungkinkan variabel menjadi kumpulan konstanta yang telah ditentukan. Variabel harus sama dengan salah satu nilai yang telah ditentukan sebelumnya. Enumerated types ini bisa digunakan untuk arah kompas (NORTH, SOUTH, EAST, WEST) dan lainnya. Berikut adalah contoh Enumerated types untuk membuat deskripsi nama hari:

```
public enum  Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
```

```java
public class EnumType {
    Day day;

    public EnumType(Day day){
        this.day = day;
    }

    public void aboutThisDay(){
        switch (day){
            case MONDAY:
                System.out.println("Mondays are bad");
                break;
            case FRIDAY:
                System.out.println("Fridays are better");
                break;
            case SATURDAY: case SUNDAY:
                System.out.println("Weekend are best");
                break;
            default:
                System.out.println("Midweek are :(");
                break;
        }
    }
    public static void main(String[] args) {
        EnumType firstDay = new EnumType(Day.MONDAY);
        firstDay.aboutThisDay();
        EnumType thirdDay = new EnumType(Day.WEDNESDAY);
        thirdDay.aboutThisDay();
        EnumType fifthDay = new EnumType(Day.FRIDAY);
        fifthDay.aboutThisDay();
        EnumType sixthDay = new EnumType(Day.SATURDAY);
        sixthDay.aboutThisDay();
    }
}
```

LATIHAN PRAKTIKUM

LATIHAN 1

Berikut adalah kode program dengan pemanfaatan generics pada class. Lakukan percobaan latihan dengan mengetik kode program pada IDE anda untuk mengetahui outputnya.

```java
public class Kota<E> {
    private E element;

    public Kota(E kota){
        element = kota;
    }

    public E getElement(){
        return element;
    }

    public static void main(String[] args) {
        Kota<Integer> jumlahKota = new Kota<Integer>(9);
        Kota<String> namaKota = new Kota<String>("Malang");
        System.out.println("Jumlah Kota di Jawa Timur : "+ jumlahKota.getElement() + " kota");
        System.out.println("Salah Satu Kota di Jawa Timur : Kota "+ namaKota.getElement());
    }
}
```

LATIHAN 2

Pemanfaatan generics juga bisa terdapat pada array atau bisa disebut array of object. Cobalah untuk mengikuti kode program dibawah agar mengetahui alur proses sehingga menambah pemahaman Anda.

1. Buat class generics untuk konsumsi

```java
public class Konsumsi<M, I> {
    private M m;
    private I i;

    public M getM(){
        return m;
    }

    public I getI(){
        return i;
    }

    public void setKonsumsi (M makanan, I minuman){
        this.m = makanan;
        this.i = minuman;
    }
```

2. Buat class hidangan

```
public class Hidangan {
    protected String namaHidangan;

    public String getNamaHidangan(){
        return namaHidangan;
    }

    public void setNamaHidangan(String namaHidangan){
        this.namaHidangan = namaHidangan;
    }

    public String disantap() {
        return "Makanan Dihidangkan";
    }
```

3. Pemanfaatan inheritance pada class minuman

```
public class Minuman extends Hidangan{
    public String disantap(){
        return this.getNamaHidangan() + " diminum";
    }
```

4. Pemanfaatan inheritance pada class makanan

```
public class Makanan extends Hidangan{
    public String disantap(){
        return this.getNamaHidangan() + " dimakan";
    }
```

5. Class main

```
public class Main {
    public static void main(String[] args) {
        ArrayList<Konsumsi> listKonsumsi = new ArrayList<>();
        Konsumsi<Makanan, Minuman> breakfast = new Konsumsi<>();
        Konsumsi<Makanan, Minuman> lunch = new Konsumsi<>();

        Makanan roti = new Makanan();
        roti.setNamaHidangan("Roti Tawar");
        Minuman susu = new Minuman();
        susu.setNamaHidangan("Susu Sapi");
        breakfast.setKonsumsi(roti, susu);
        listKonsumsi.add(breakfast);

        Makanan nasi = new Makanan();
        nasi.setNamaHidangan("Nasi Putih");
        Minuman air = new Minuman();
        air.setNamaHidangan("Air Putih");
```

```
        lunch.setKonsumsi(nasi, air);
        listKonsumsi.add(lunch);

        System.out.println("Menu Konsumsi");
        for (Konsumsi<Makanan,Minuman> konsumsi: listKonsumsi){
            Makanan makanan = konsumsi.getM();
            Minuman minuman = konsumsi.getI();

            System.out.println(makanan.disantap());
            System.out.println(minuman.disantap());
        }
    }
}
```

TUGAS PRAKTIKUM

KEGIATAN 1

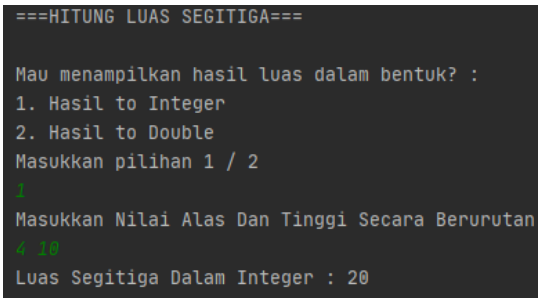
Buat class generic bernama “Segitiga” yang akan digunakan untuk menyimpan nilai alas dan tinggi dengan syarat hanya menerima jenis (Numbers). Tugas ini dibuat dengan tujuan menghitung nilai inputan dan mendapatkan hasil luas dari Segitiga dalam 2 tipe. Berikut adalah syarat yang harus dipenuhi dalam menyelesaikan tugas ini:

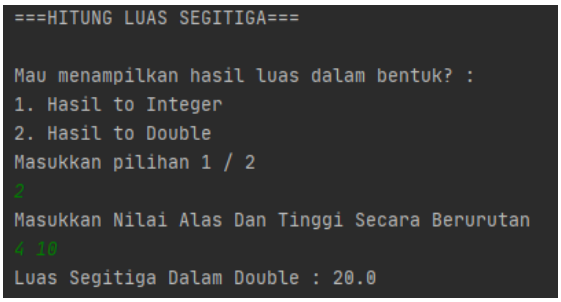
1. Fungsi dalam class generic:

Tipe	**Nama Method**	**Deskripsi**
Constructor	Segitiga(T alas, T tinggi)	Inisialisasi variabel alas dan tinggi.
Int	getResultAsInt()	Mendapatkan hasil perhitungan luas segitiga dalam bentuk tipe integer.
Double	getResultAsDouble()	Mendapatkan hasil perhitungan luas segitiga dalam bentuk tipe double.

2. Memanfaatkan fungsi Scanner untuk menerima input nilai alas dan tinggi dari user
3. Contoh inisialisasi object pada main dapat dilakukan seperti ini:
   Segitiga<Integer> luasTipeInt = new Segitiga<>();
   Segitiga<Double> luasTipeDouble = new Segitiga<>();

Berikut adalah contoh output program perhitungan luas segitiga:





Mahasiswa diberbolehkan melakukan improvisasi dari tugas yang diberikan dengan syarat tidak mengurangi ketentuan yang ada.

CATATAN

Aturan umum penulisan JAVA agar mudah dikoreksi oleh asisten:

1. Untuk nama class, enum, dan yang lainnya biasakan menggunakan gaya CamelCase (diawali dengan huruf besar pada tiap kata untuk mengganti spasi) seperti: Kursi, JalanRaya, ParkiranGedung, dan lain seterusnya.
2. Untuk penulisan nama method dan attribute diawali dengan huruf kecil di awal kata dan menggunakan huruf besar untuk kata setelahnya, seperti: getNamaJalan, namaJalan, harga, setNamaJalan, dan lain seterusnya.
3. Jika menggunakan IDE Intellij jangan lupa untuk memformat kode agar terlihat rapi menggunakan menu code -> show reformat file dialog -> centang semua field dan klik ok.

DETAIL PENILAIAN TUGAS

Kriteria	Nilai
Semua Ketentuan Terpenuhi dan Program Tanpa Error	40
Tidak Ada Plagiasi	25
Pemahaman Materi dan Coding Yang Dibuat	35