



## **JOBSHEET 10**

### **POLIMORFISME**

#### **2.1 Tujuan Praktikum**

Setelah melakukan materi praktikum ini, mahasiswa mampu:

1. Memahami dan mengimplementasikan object casting, baik upcast maupun down cast
2. Memahami dan mengimplementasikan heterogenous collection
3. Memahami dan mengimplementasikan polymorphic arguments

#### **2.2 Polimorfisme**

Polimorfisme merupakan kemampuan suatu objek untuk memiliki banyak bentuk. Penggunaan polimorfisme yang paling umum dalam OOP terjadi ketika ada suatu object yang diinstansiasi dari suatu class tapi dikenali/diidentifikasi/diperlakukan sebagai object bertipe class lainnya. Konsep polimorfisme dapat diterapkan pada class-class yang memiliki relasi inheritance.

#### **2.3 Heterogenous Collection**

Umumnya collection dalam bahasa pemrograman Java (seperti Array, ArrayList, Stack, Queue, dll) hanya dapat menyimpan elemen bertipe data yang sama.

Contoh:

```
int[] daftarNilai = new int[10];
```

daftarNilai merupakan array of int, berarti elemen yang boleh ditambahkan ke array tersebut hanya elemen bertipe int

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

students merupakan ArrayList of Students, berarti elemen yang boleh ditambahkan dalam ke array list tersebut hanya elemen bertipe Student

Konsep polimorfisme memungkinkan collection bersifat heterogen, artinya collection dapat menyimpan elemen-elemen dengan tipe data yang berbeda, asalkan berada dalam hirarki inheritance.



1. Buatlah folder baru dengan nama Praktikum10. Di dalamnya, buat class Pegawai

```
public class Pegawai {  
    public String nip;  
    public String nama;  
  
    public Pegawai(){  
  
    }  
  
    public Pegawai(String nip, String nama) {  
        this.nip = nip;  
        this.nama = nama;  
    }  
  
    public void displayInfo()  
    {  
        System.out.println("NIP: " + nip);  
        System.out.println("Nama: " + nama);  
    }  
}
```

2. Tambahkan class Dosen

```
public class Dosen extends Pegawai {  
    public String nidn;  
  
    public Dosen() {  
    }  
  
    public Dosen(String nip, String nama, String nidn) {  
        super(nip, nama);  
        this.nidn = nidn;  
    }  
  
    public void displayInfo(){  
        super.displayInfo();  
        System.out.println("NIDN: " + nidn);  
    }  
  
    public void mengajar(){  
        System.out.println("Membuat rencana pembelajaran");  
        System.out.println("Menyusun materi");  
        System.out.println("Melaksanakan PBM");  
        System.out.println("Melakukan evaluasi");  
    }  
}
```



### 3. Tambahkan class TenagaKependidikan

```
public class TenagaKependidikan extends Pegawai {
    public String kategori;

    public TenagaKependidikan(){

    }

    public TenagaKependidikan(String nip, String nama, String kategori){
        super(nip, nama);
        this.kategori = kategori;
    }

    public void displayInfo(){
        super.displayInfo();
        System.out.println("Kategori: " + kategori);
    }
}
```

### 4. Buatlah class Demo beserta fungsi main(). Di dalam fungsi main(), instansiasi beberapa object dosen dan tenaga kependidikan sebagai berikut

```
Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");
Dosen dosen2 = new Dosen("19700105", "Muhammad, S.T, M.T", "197001");
TenagaKependidikan tendik1 = new TenagaKependidikan("19750301", "Aida, A.Md.", "Tenaga Administrasi");
TenagaKependidikan tendik2 = new TenagaKependidikan("19650304", "Rika, S.T.", "Tenaga Laboratorium");
```

### 5. Buatlah arrayList daftarPegawai bertipe ArrayList of Pegawai.

```
ArrayList<Pegawai> daftarPegawai = new ArrayList<Pegawai>();
```

### 6. Konsep polimorfisme mengizinkan object dosen1, dosen2, tendik1, dan tendik2 untuk ditambahkan ke Array List daftarPegawai meskipun tidak secara eksplisit bertipe Pegawai.

```
daftarPegawai.add(dosen1);
daftarPegawai.add(dosen2);
daftarPegawai.add(tendik1);
daftarPegawai.add(tendik2);

System.out.println("Jumlah Pegawai: " + daftarPegawai.size());
```

### 7. Compile dan run program untuk memastikan bahwa heterogenous collection dapat dibuat.

## 2.4 Object Casting

Typecasting merupakan proses konversi variable dari suatu tipe data menjadi tipe data lainnya. Object casting adalah typecasting yang dilakukan terhadap object. Konsep polimorfisme berkaitan erat dengan object casting. Terdapat 2 jenis object casting:

#### ➤ Upcasting

- Upcasting dilakukan untuk mengubah child object menjadi parent object
- Dapat dilakukan secara implisit (nama parent class tidak perlu dituliskan)



➤ Downcasting

- Upcasting dilakukan untuk mengubah parent object menjadi child object
- Harus dilakukan secara eksplisit (nama child class dituliskan) karena bisa jadi parent class-nya memiliki lebih dari 1 child class

1. Pada langkah sebelumnya, Anda telah membuat object dosen1 yang diinstansiasi dari class Dosen. Object dosen1 bertipe Dosen. Dengan kata lain, object tersebut akan dikenali/diperlakukan sebagai object bertipe Dosen. Oleh karena itu, dosen1 memiliki atribut NIDN dan dapat memanggil method mengajar()
2. Modifikasi fungsi main() sebagai berikut

```
Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");

System.out.println(dosen1.nip);
System.out.println(dosen1.nama);
System.out.println(dosen1.nidn);
dosen1.mengajar();
```

3. Run dan compile kode program. Amati hasilnya.

```
19940201
Widia, S.Kom. M.Kom
199402
Membuat rencana pembelajaran
Menyusun materi
Melaksanakan PBM
Melakukan evaluasi
```

4. Lakukan upcasting object dosen1 menjadi object dari parent class nya, yaitu Pegawai. Object pegawai1 merupakan hasil instansiasi dari class Dosen, tetapi proses upcasting ini membuat pegawai1 dikenali dan diperlakukan sebagai object bertipe Pegawai.

```
Pegawai pegawai1 = dosen1;
```

5. Modifikasi kode program sebagai berikut

```
Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");

Pegawai pegawai1 = dosen1;

System.out.println(pegawai1.nip);
System.out.println(pegawai1.nama);
System.out.println(pegawai1.nidn);
pegawai1.mengajar();
```



6. Tidak ada compile error pada baris kode upcasting. Error muncul saat mengakses atribut NIDN dan memanggil method mengajar() karena object1 dikenali sebagai object bertipe Pegawai, sementara class Pegawai tidak memiliki atribut NIDN dan method mengajar()
7. Modifikasi fungsi main sebagai berikut

```
Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");

Pegawai pegawai1 = dosen1;

System.out.println(pegawai1.nip);
System.out.println(pegawai1.nama);
pegawai1.displayInfo();
```

8. Run dan compile kode program, amati hasilnya

```
19940201
Widia, S.Kom. M.Kom
NIP: 19940201
Nama: Widia, S.Kom. M.Kom
NIDN: 199402
```

9. Perhatikan bahwa method displayInfo() dapat dipanggil oleh object pegawai1 karena terdapat method displayInfo() pada class Pegawai sehingga tidak muncul compile error. Tetapi saat program di-run, yang dieksekusi adalah method displayInfo() pada class Dosen, karena adanya overriding
10. Cobalah lakukan downcasting object pegawai1 ke class TenagaKependidikan. Perhatikan bahwa downcasting harus dilakukan secara eksplisit dengan menyebutkan nama subclass nya.

```
Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");

Pegawai pegawai1 = dosen1;

System.out.println(pegawai1.nip);
System.out.println(pegawai1.nama);
pegawai1.displayInfo();

TenagaKependidikan test = (TenagaKependidikan) pegawai1;
```

11. Tidak terdapat warning pada kode program karena tidak ada compile error sebab TenagaKependidikan merupakan subclass dari class Pegawai.
12. Run program dan amati bahawa terdapat runtime error java.lang.ClassCastException kerana object tersebut bukan instance dari class TenagaKependidikan.



13. Cobalah lakukan downcasting object pegawai1 kembali ke class Dosen.

```
Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");

Pegawai pegawai1 = dosen1;

System.out.println(pegawai1.nip);
System.out.println(pegawai1.nama);
pegawai1.displayInfo();

Dosen newDosen = (Dosen) pegawai1;

System.out.println(newDosen.nama);
System.out.println(newDosen.nidn);
newDosen.mengajar();
```

Object newDosen sekarang sudah dikenali kembali sebagai object bertipe Dosen. Oleh karena itu, atribut NIDN dapat diakses dan method mengajar() juga dapat dipanggil

14. Run dan compile kode program kemudian amati hasilnya

## 2.5 Polymorphic Arguments & instanceof

Konsep polimorfisme juga memungkinkan parameter dari suatu method menerima argument dengan berbagai bentuk object asalkan berada dalam hirarki inheritance.

1. Misalnya pada class Demo terdapat method train() yang bertujuan untuk memberikan pelatihan bagi pegawai baru.

```
public static void train(Pegawai pegawai){
    System.out.println("Memberikan pelatihan untuk pegawai");
    pegawai.displayInfo();
}
```

2. Dengan konsep polimorfisme, method train() tidak hanya dapat dipanggil dengan argument bertipe Pegawai, tetapi juga subclass Pegawai, yaitu Dosen dan TenagaKependidikan.
3. Modifikasi kode program sebagai berikut

```
public class Demo {
    Run | Debug
    public static void main(String[] args) {
        Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");
        TenagaKependidikan tendik1 = new TenagaKependidikan("19750301", "Aida, A.Md.", "Tenaga Administrasi");

        train(dosen1);
        train(tendik1);
    }

    public static void train(Pegawai pegawai){
        System.out.println("Memberikan pelatihan untuk pegawai");
        pegawai.displayInfo();
    }
}
```



4. Perhatikan bahwa terdapat proses upcasting (**secara implisit**) dalam polymorphic argument, artinya di dalam method `train()` object pegawai akan dikenali sebagai object bertipe `Pegawai`, sehingga atribut `NIDN` dan kategori tidak dapat diakses. Di samping itu, method `mengajar()` juga tidak dapat dipanggil.

```
public class Demo {
    Run | Debug
    public static void main(String[] args) {
        Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");
        TenagaKependidikan tendik1 = new TenagaKependidikan("19750301", "Aida, A.Md.", "Tenaga Administrasi");

        train(dosen1);
        train(tendik1);
    }

    public static void train(Pegawai pegawai){
        System.out.println("Memberikan pelatihan untuk pegawai");
        pegawai.displayInfo();

        //hanya test
        System.out.println(pegawai.nidn);
        System.out.println(pegawai.kategori);
        pegawai.mengajar();
    }
}
```

5. Jika object perlu dikenali sebagai class asalnya, lakukan proses downcasting seperti percobaan sebelumnya.
6. Misalnya method `train()` memiliki proses yang sedikit berbeda untuk dosen dan tenaga kependidikan. Keyword `instanceof` dapat digunakan untuk mengetahui dari class mana suatu object diinstansiasi untuk memastikan tidak ada runtime error saat proses downcasting.

```
public class Demo {
    Run | Debug
    public static void main(String[] args) {
        Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");
        TenagaKependidikan tendik1 = new TenagaKependidikan("19750301", "Aida, A.Md.", "Tenaga Administrasi");

        train(dosen1);
        train(tendik1);
    }

    public static void train(Pegawai pegawai){
        pegawai.displayInfo();
        System.out.println("Mengenalkan lingkungan kampus");
        System.out.println("Menginfokan SOP/Juknis");

        if (pegawai instanceof Dosen) {
            System.out.println("Memberikan pelatihan pedagogik");
        }
    }
}
```

7. Run program kemudian amati hasilnya



## 2.6 Pertanyaan

1. Apakah upcasting dan downcasting dapat dilakukan dari suatu class terhadap class lain yang tidak memiliki relasi inheritance?
2. Dari 2 baris kode program berikut, manakah proses upcasting yang tepat? Jelaskan  
`Pegawai pegawai1 = new Dosen();`  
`Pegawai pegawai1 = (Pegawai) new Dosen();`
3. Apa fungsi dari keyword instanceof?
4. Apa yang dimaksud heterogenous collection?
5. Sebuah object diinstansiasi dari class Pegawai. Kemudian dilakukan downcasting menjadi object bertipe Dosen. Apakah hal ini dapat dilakukan? Lakukan percobaan untuk membuktikannya.

## 2.7 Heterogenous Collection & Polymorphic Arguments dengan Interface

Selain class-class yang berada dalam hirarki pewarisan, konsep polimorfisme juga dapat diterapkan pada class-class yang meng-implement interface yang sama.

1. Buatlah interface Printable

```
public interface Printable {  
    void print();  
}
```

2. Buatlah class yang meng-implements interface Printable yaitu class Document

```
public class Document implements Printable {  
    public String fileName;  
    public String content;  
  
    @Override  
    public void print() {  
        System.out.println("Mencetak dokumen: " + content);  
    }  
}
```





3. Buatlah class lain yang meng-implements interface Printable yaitu Webpage

```
public class Webpage implements Printable {
    public String url;
    public String title;
    public String content;

    @Override
    public void print() {
        System.out.println("URL: " + url);
        System.out.println("Title: " + title);

        String plainText = content.replaceAll("<[^>]*>", "");
        System.out.println(plainText);
    }
}
```

4. Tambahkan class NewDemo.java beserta fungsi main() lalu instansiasi object dari class Document dan Webpage. Selanjutnya instansiasi ArrayList of Printables. Konsep polimorfisme mengizinkan ArrayList ini menampung object-object dari class yang berbeda asal class tersebut mengimplements interface Printables.

```
public class NewDemo {
    Run | Debug
    public static void main(String[] args) {
        Webpage webpage1 = new Webpage();
        webpage1.title = "Jurusan Teknologi Informasi Polinema";
        webpage1.url = "https://jti.polinema.ac.id/";
        webpage1.content = "<h1>Ini contoh heading</h1><br/>";

        Document doc1 = new Document();
        doc1.fileName = "PBO 2024.docx";
        doc1.content = "Contoh content Pemrograman Berorientasi Objek";

        ArrayList<Printable> printables = new ArrayList<>();
        printables.add(webpage1);
        printables.add(doc1);
    }
}
```

5. Buatlah class Printer dengan method cetak() yang memiliki parameter bertipe Printable

```
public class Printer {
    public void cetak(Printable printable) {
        printable.print();
    }
}
```



6. Pada NewDemo, instansiasi object dari class Printer kemudian panggil method cetak() dengan argumen webpage1 bertipe Webpage dan doc1 bertipe Document.

```
Printer printer1 = new Printer();
printer1.cetak(webpage1);
printer1.cetak(doc1);
```

Konsep polymorphic arguments juga diperbolehkan tidak hanya untuk class-class dalam hirarki inheritance, tetapi juga class-class yang meng-implements interface yang sama. Pada contoh di atas, method cetak() dapat menerima argumen bertipe class apapun asal class tersebut meng-implement interface Printable.

## 2.8 Polymorphic Arguments dengan Built-In Interface

Built-in interfaces merujuk pada interface yang sudah disediakan oleh Java API dan dapat digunakan langsung dalam program. Interface ini mendefinisikan kontrak/panduan mengenai method yang harus diimplementasikan oleh sebuah class untuk memenuhi kapabilitas tertentu. Salah satu interface yang sering digunakan adalah Comparable.

1. Buatlah class baru bernama ComparableDemo.java beserta fungsi main() nya
2. Pada fungsi main() buatlah ArrayList of Dosen bernama daftarDosen kemudian tambahkan beberapa object bertipe Dosen ke dalam ArrayList daftarDosen.

```
public class ComparableDemo {
    Run | Debug
    public static void main(String[] args) {
        ArrayList<Dosen> daftarDosen = new ArrayList<>();
        daftarDosen.add(new Dosen("321", "Bella", "0091"));
        daftarDosen.add(new Dosen("123", "Adila", "0081"));
        daftarDosen.add(new Dosen("231", "Petrus", "0071"));
    }
}
```

3. Java telah menyediakan static method sort() pada class Collections yang dapat digunakan untuk melakukan sorting elemen-elemen di dalam list.

```
Collections.sort(daftarDosen);
```

4. Method sort() dapat menerima polymorphic arguments, artinya method ini dapat menerima argumen list of objects dengan berbagai tipe data yang berbeda. Tetapi dapat dilihat bahwa pada langkah sebelumnya, masih terdapat error sebab meskipun bersifat polymorphic, class dari elemen di dalam list harus meng-implement interface Comparable.



The method `sort(List<T>)` in the type `Collections` is not applicable for the arguments `(ArrayList<Dosen>)` Java(67108979)

```
<T> void java.util.Collections.sort(List<T> list)
```

Sorts the specified list into ascending order, according to the [natural ordering](#) of its elements. All elements in the list must implement the `Comparable` interface. Furthermore, all elements in the list must be *mutually comparable* (that is, `e1.compareTo(e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the list).

This sort is guaranteed to be *stable*: equal elements will not be reordered as a result of the sort.

The specified list must be modifiable, but need not be resizable.

- **Type Parameters:**
  - `T` the class of the objects in the list

```
sort(daftarDosen);
```

5. Oleh karena itu, class `Dosen` harus mengimplements interface `Comparable`

```
public class Dosen extends Pegawai implements Comparable<Dosen> {
```

6. Interface `Comparable` akan “memaksa” class `Dosen` untuk mengimplements method `compareTo()`. Gunakan kode program berikut jika Anda ingin mengurutkan object `Dosen` berdasarkan atribut nama.

```
@Override
public int compareTo(Dosen otherDosen) {
    return this.nama.compareTo(otherDosen.nama);
}
```

Catatan: implementasi method `compareTo()` sesuai tipe data atribut yang dipilih. Anda juga dapat menggunakan static method `Double.compare()` atau `Integer.compare()`.

7. Untuk mengecek hasil sorting, tambahkan kode program untuk menampilkan `daftarDosen`

```
public static void main(String[] args) {
    ArrayList<Dosen> daftarDosen = new ArrayList<>();
    daftarDosen.add(new Dosen("321", "Bella", "0091"));
    daftarDosen.add(new Dosen("123", "Adila", "0081"));
    daftarDosen.add(new Dosen("231", "Petrus", "0071"));

    Collections.sort(daftarDosen);

    for (Dosen dosen : daftarDosen) {
        dosen.displayInfo();
    }
}
```

8. Note: jika tidak diimport oleh VS Code secara otomatis, lakukan import

```
import java.util.ArrayList;
import java.util.Collections;
```