JOBSHEET 10
POLIMORFISME

SAFRIZAL RAHMAN

https://github.com/safrizalrahman46/PBO_SAFRIZ_THEVIGILANTE

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

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

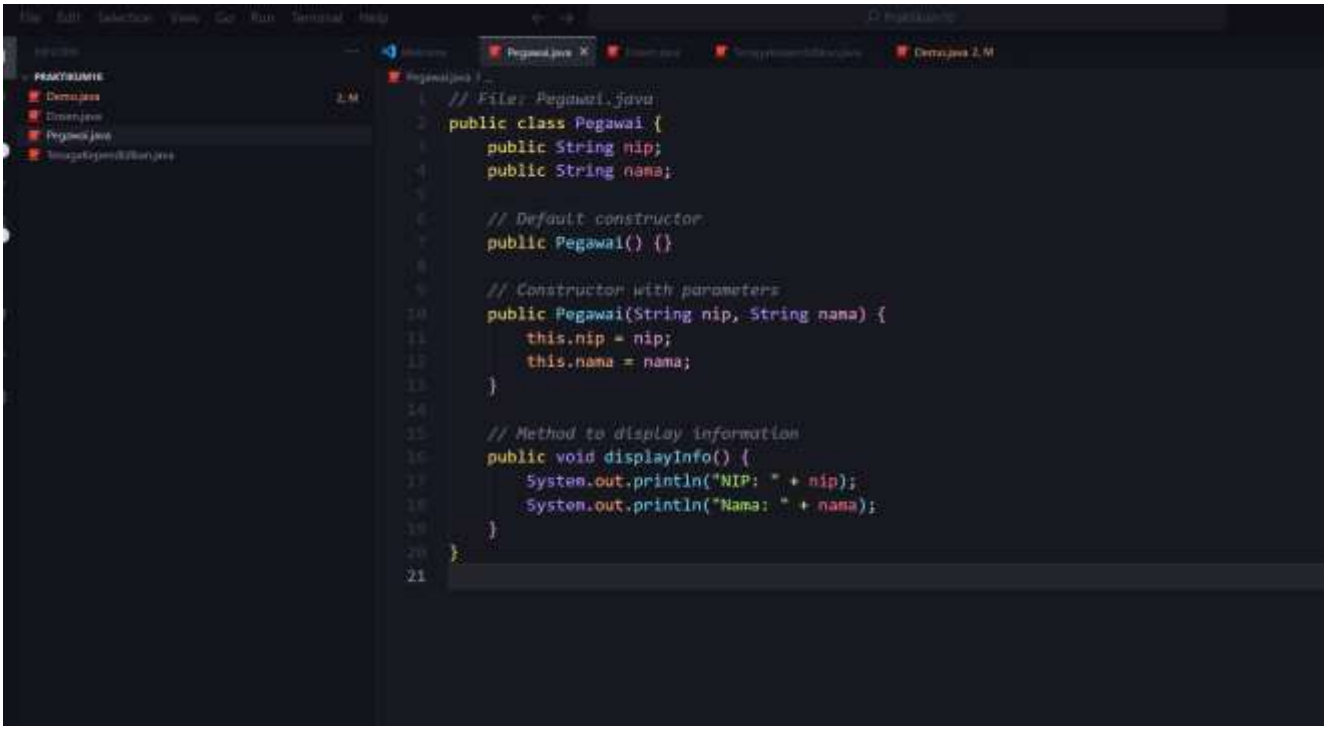
```
ArrayList<Student> students = new ArrayList<Student>();
```

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

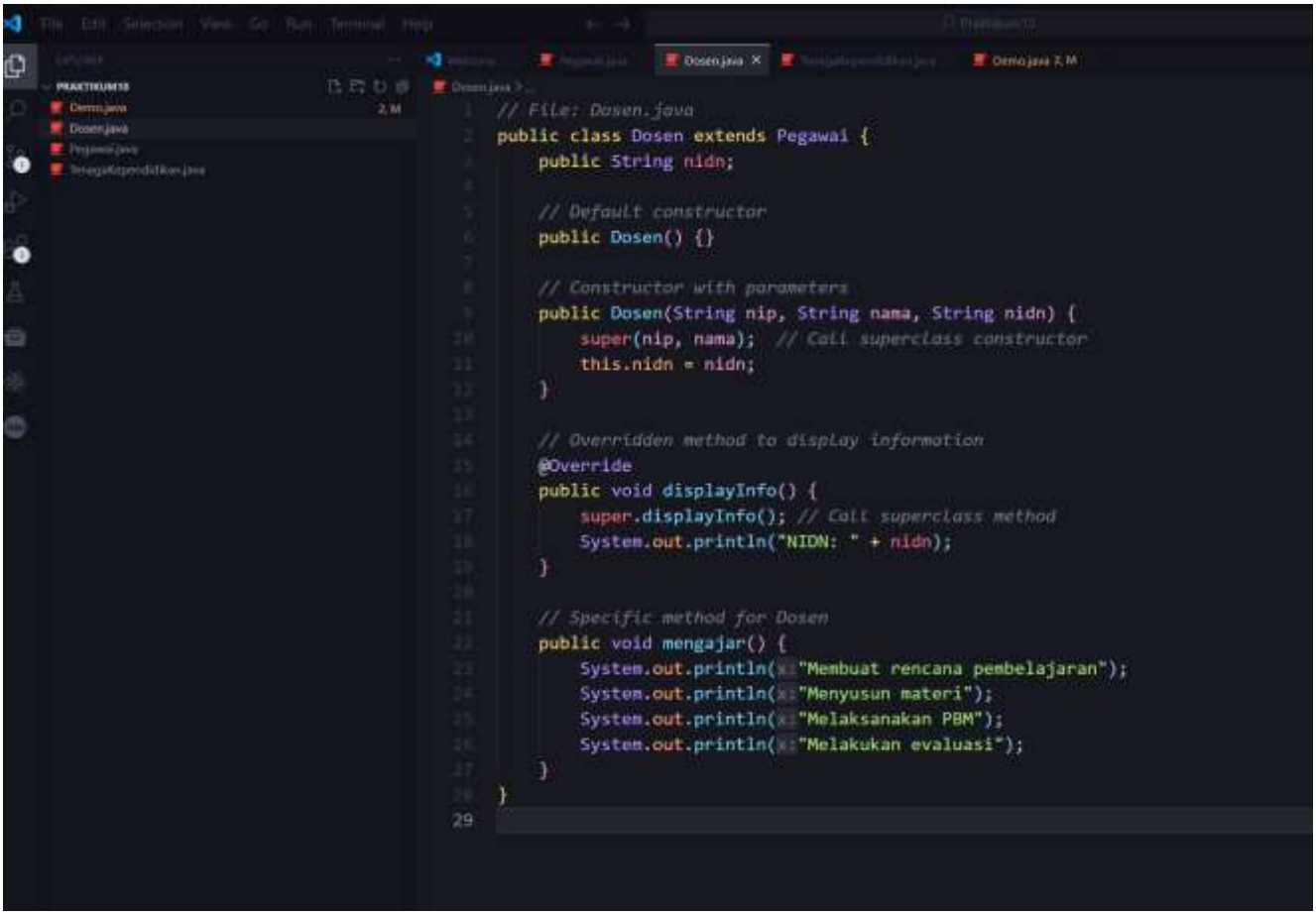
```
public class Pegawai {
    public String nip;
    public String nama;

    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);
    }
}
```



```java
// File: Pegawai.java
public class Pegawai {
    public String nip;
    public String nama;

    // Default constructor
    public Pegawai() {}

    // Constructor with parameters
    public Pegawai(String nip, String nama) {
        this.nip = nip;
        this.nama = nama;
    }

    // Method to display information
    public void displayInfo() {
        System.out.println("NIP: " + nip);
        System.out.println("Nama: " + nama);
    }
}
```

2. Tambahkan class Dosen

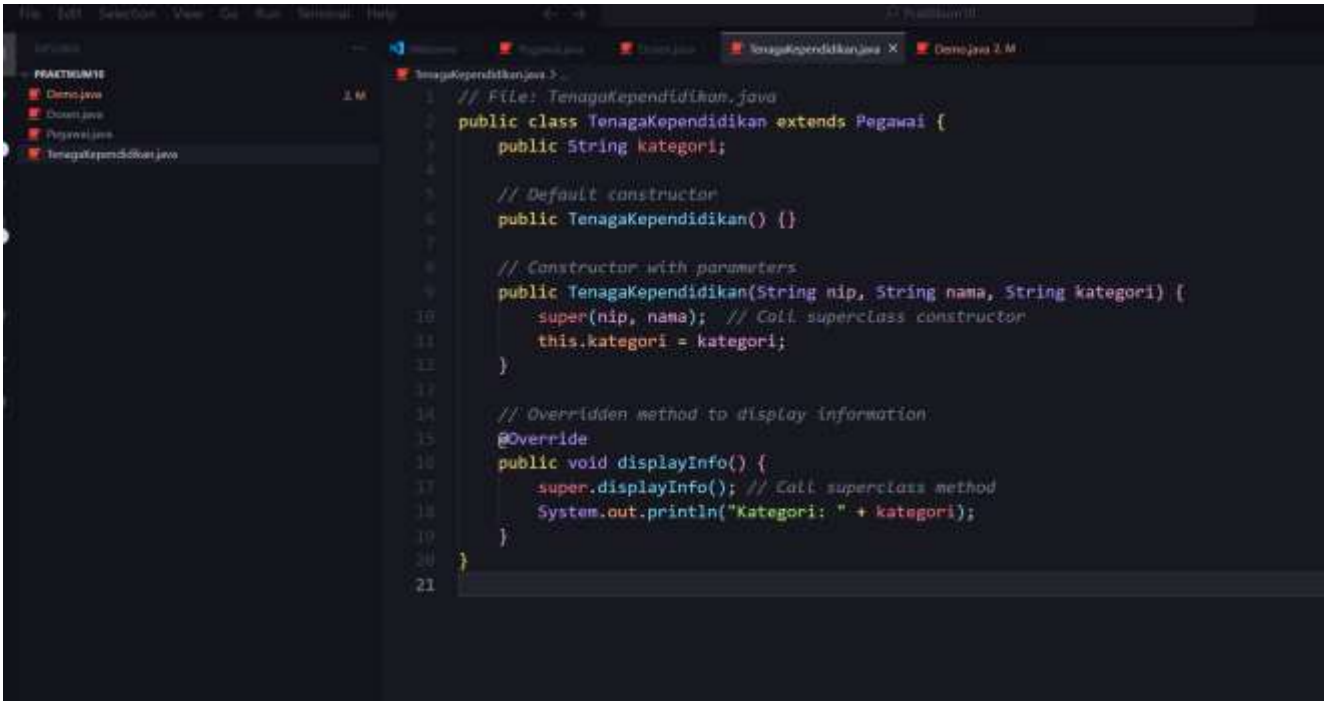
```java
public class Dosen extends Pegawai {
    public String nidn;

    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");
    }
}
```



```java
// File: Dosen.java
public class Dosen extends Pegawai {
    public String nidn;

    // Default constructor
    public Dosen() {}

    // Constructor with parameters
    public Dosen(String nip, String nama, String nidn) {
        super(nip, nama);  // Call superclass constructor
        this.nidn = nidn;
    }

    // Overridden method to display information
    @Override
    public void displayInfo() {
        super.displayInfo(); // Call superclass method
        System.out.println("NIDN: " + nidn);
    }

    // Specific method for Dosen
    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

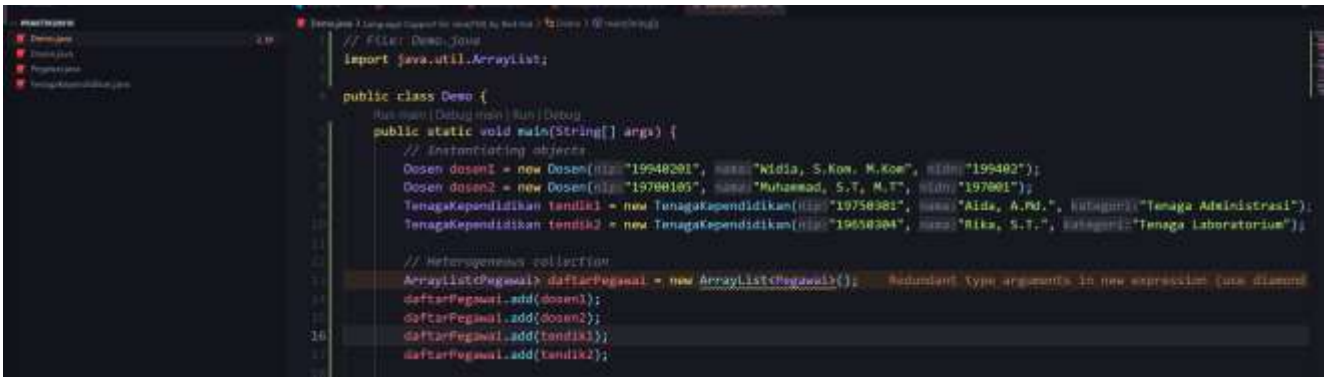
```java
public class TenagaKependidikan extends Pegawai {
    public String kategori;

    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);
    }
}
```



```java
// File: TenagaKependidikan.java
public class TenagaKependidikan extends Pegawai {
    public String kategori;

    // Default constructor
    public TenagaKependidikan() {}

    // Constructor with parameters
    public TenagaKependidikan(String nip, String nama, String kategori) {
        super(nip, nama);  // Call superclass constructor
        this.kategori = kategori;
    }

    // Overridden method to display information
    @Override
    public void displayInfo() {
        super.displayInfo(); // Call superclass method
        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

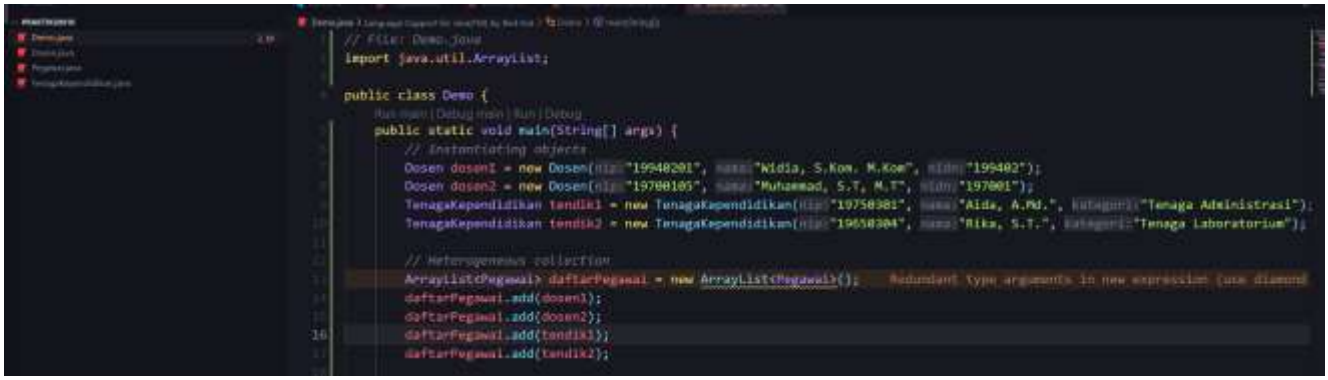
```java
Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");
Dosen dosen2 = new Dosen("19700105", "Muhammad, S.T, M.T", "197001");
TenagaKependidikan tendik1 = new TenagaKependidikan("19750301", "Aida, A.Md.", "Tenaga Administrasi");
TenagaKependidikan tendik2 = new TenagaKependidikan("19650304",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



```java
// File: Demo.java
import java.util.ArrayList;

public class Demo {
    public static void main(String[] args) {
        // Instantiating objects
        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");

        // Heterogeneous collection
        ArrayList<Pegawai> daftarPegawai = new ArrayList<Pegawai>();   Redundant type arguments in new expression (use diamond
        daftarPegawai.add(dosen1);
        daftarPegawai.add(dosen2);
        daftarPegawai.add(tendik1);
        daftarPegawai.add(tendik2);
```

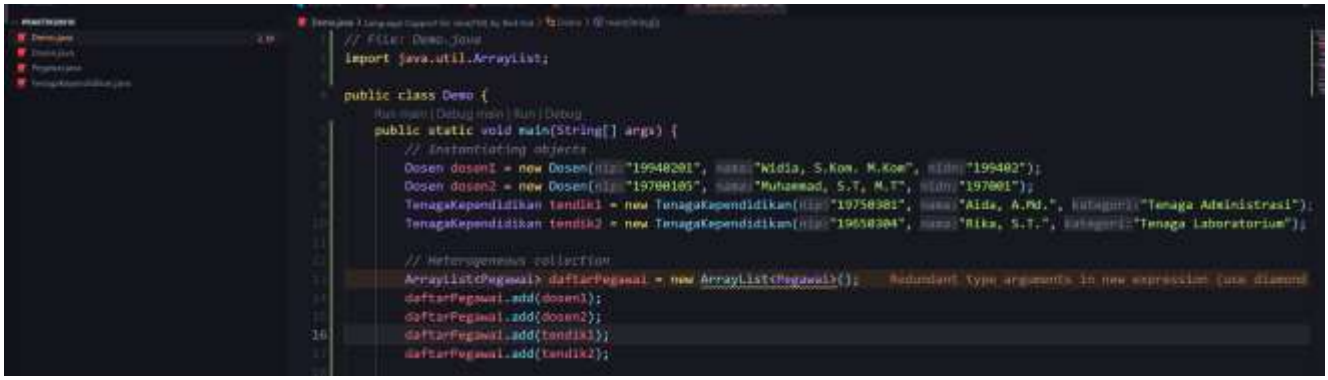
5. Buatlah arrayList daftarPegawai bertipe ArrayList of Pegawai.

```java
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

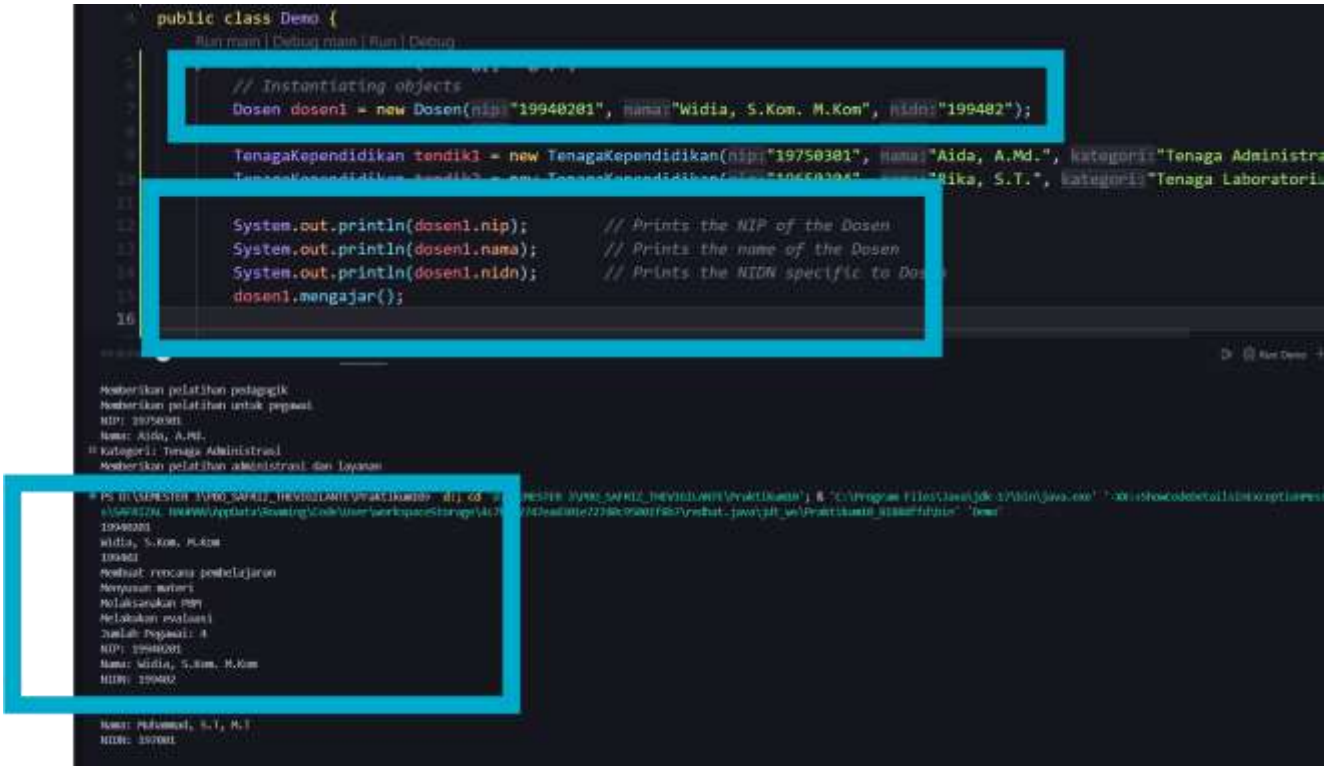
- Upcasting dilakukan untuk mengubah parent object menjadi child object
- Harus dilakukan secara eksplisit (nama child class dituliskan) karena bisa jadi parent class-nya memiliki lebih dari 1 child class

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
```

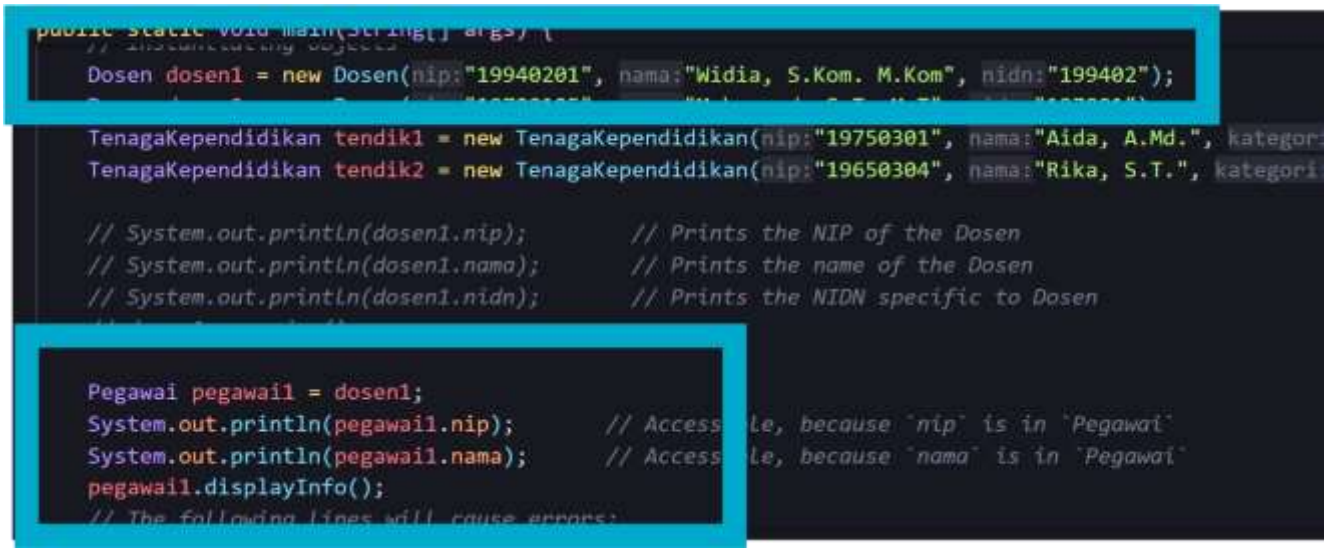


4. Lakukan upcasting object dosen1 menjadi object dari parent class nya, yaitu Pegawai. Object pegawai1 merupakan hasil instansiasi dari class Dosen, tetapi proses upcasting ini membuat pegawai1 dikenali dan diperlakukan sebagai object bertipe Pegawai.

```
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();
```



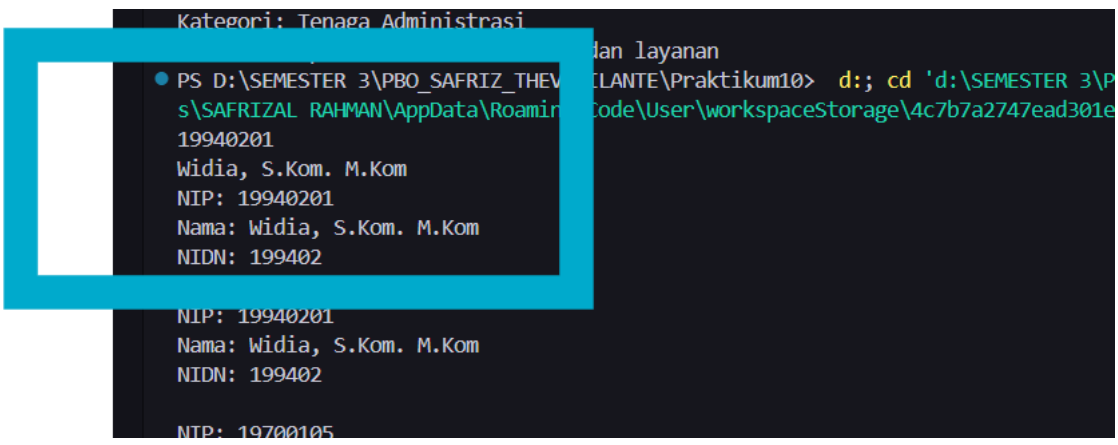
6. Tidak ada compile error pada baris kode upcasting. Error muncul saat mengakses atribut NIDN dan memanggil method mengajar() karena object1 dikenali sebagai object bertipe Pegawai, sementara class Pegawai tidak memiliki atribut NIDN dan method mengajar() 7.

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 TenagaKependididikan. 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;
```

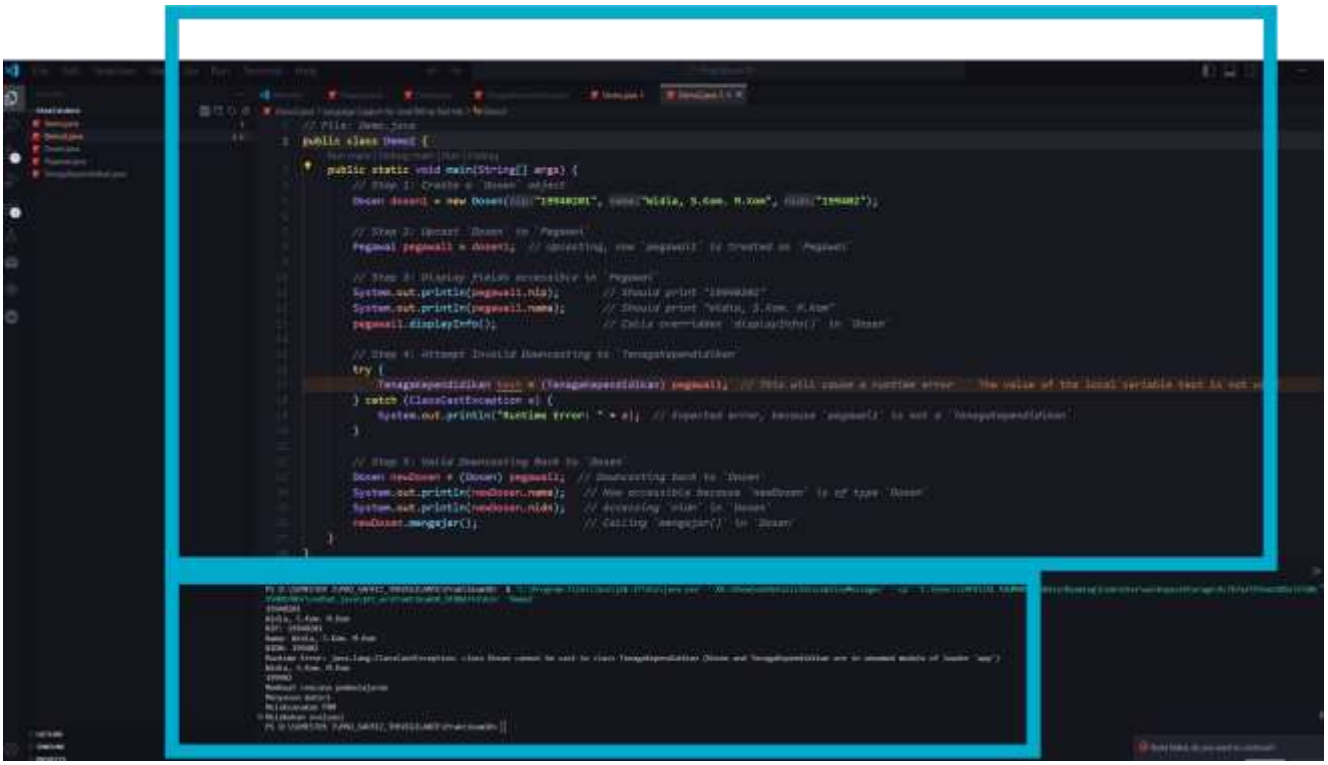
11. Tidak terdapat warning pada kode program karena tidak ada compile error sebab TenagaKependidikan merupakan subclass dari class Pegawai.
12. Run program dan amati bahawa terdapat runtime error java.lang.ClassCastException kerena object tersebut bukan instance dari class TenagaKependidikan
13. Cobalah lakukan downcasting object pegawai1 kembali ke class Dosen.

```
Dosen newDosen = (Dosen) pegawai1;
```

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

```
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();
```

15. 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 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.

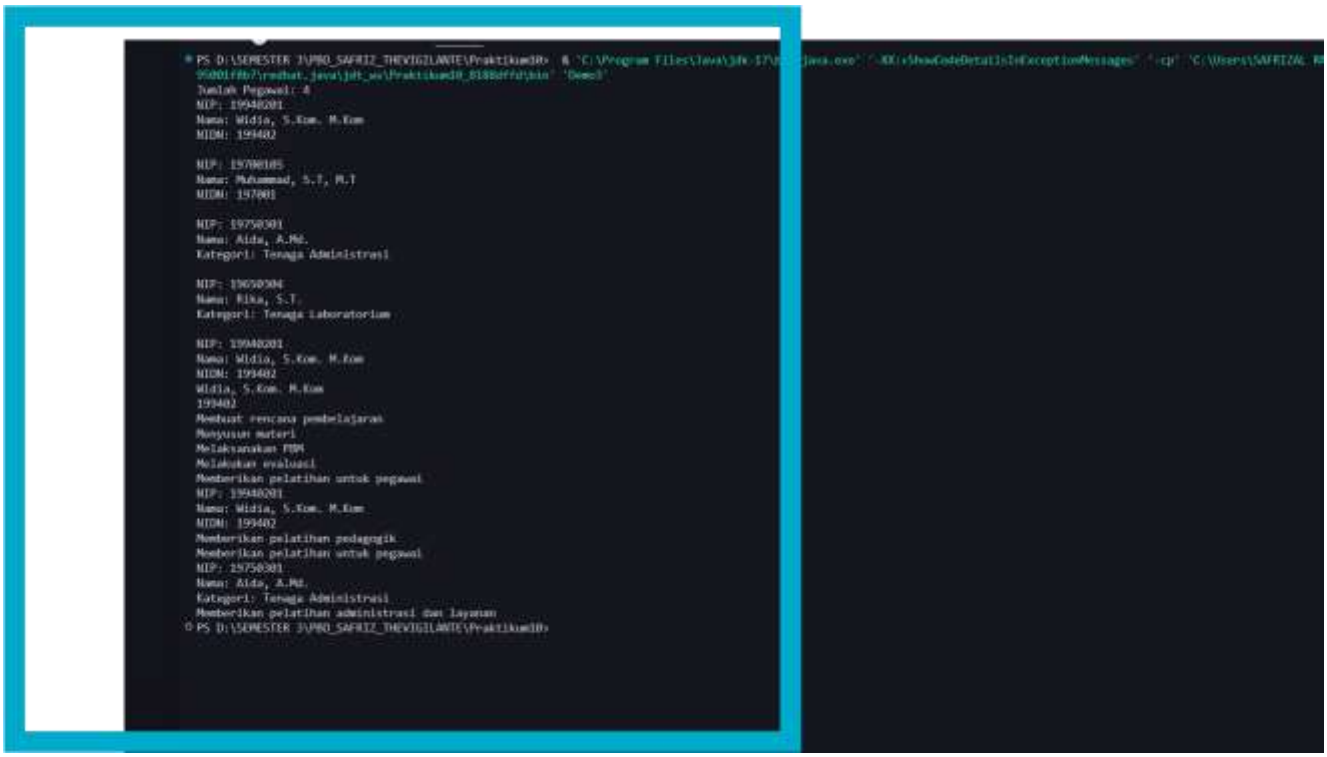
```
public class Demo {
    Run | Debug
    public static void main(String[] args) {
        Dosen dosen1 = new Dosen("19940201", "Widia, S.Kom. M.Kom", "199402");
        TenagaKependidikan tendik1 = new TenagaKependidikan("19750301", "Aida, A.Md.", "Tenaga Administrasi");

        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.

```java
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");
        }
    }
}
```

```
SAFRIZAL RAHMANNNNNN
// File: Demo.java
import java.util.ArrayList;

public class Demo3 {
    public static void main(String[] args) {
        // Instantiating objects
        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");

        // Heterogeneous collection
        ArrayList<Pegawai> daftarPegawai = new ArrayList<Pegawai>();
        daftarPegawai.add(dosen1);
        daftarPegawai.add(dosen2);
        daftarPegawai.add(tendik1);
        daftarPegawai.add(tendik2);

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

        // Displaying information from the collection
        for (Pegawai pegawai : daftarPegawai) {
            pegawai.displayInfo();
            System.out.println();
        }

        // Demonstrating upcasting and downcasting
        Pegawai pegawai1 = dosen1;  // Upcasting
        pegawai1.displayInfo();  // Calls overridden method in Dosen

        // Downcasting
        if (pegawai1 instanceof Dosen) {
            Dosen newDosen = (Dosen) pegawai1;  // Downcasting to Dosen
            System.out.println(newDosen.nama);
            System.out.println(newDosen.nidn);
            newDosen.mengajar();
        }

        // Attempting invalid downcasting (uncomment to test and observe runtime error)
        /*
        TenagaKependidikan test = (TenagaKependidikan) pegawai1;
        */

        // Using train method to demonstrate polymorphic argument
        train(dosen1);
        train(tendik1);
    }

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

        // Checking object type and customizing output
        if (pegawai instanceof Dosen) {
            System.out.println("Memberikan pelatihan pedagogik");
        } else if (pegawai instanceof TenagaKependidikan) {
            System.out.println("Memberikan pelatihan administrasi dan layanan");
        }
    }
}

// Photo by Milad Fakurian on Unsplash
```

7. Run program kemudian amati hasilnya



2.6 Pertanyaan

1. Apakan upcasting dapat dilakukan dari suatu class terhadap class lain yang tidak memiliki relasi inheritance?

   Tidak, upcasting hanya dapat dilakukan pada objek yang mempunyai hubungan pewarisan (superclass dan subclass).

2. Dari 2 baris kode program berikut, manakan proses upcasting yang tepat? Jelaskan

```
Pegawai pegawai1 = new Dosen();
Pegawai pegawai1 = (Pegawai) new Dosen();
```

3. Apa fungsi dari keyword instanceOf?

   instanceOf digunakan untuk memeriksa apakah sebuah objek adalah instance dari suatu class atau subclass tertentu, sangat berguna untuk validasi sebelum melakukan downcasting

4. Apa yang dimaksud heterogenous collection?

   Heterogenous collection adalah kumpulan data yang dapat menyimpan objek dari berbagai tipe, selama mereka berada dalam satu hierarki inheritance.

5. Sebuah object diinstansiasi dari class Pegawai. Kemudian dilakukan downcasting menjadi object bertipe Dosen. Apakah hal ini dapat dilakukan? Lakukan percobaan untuk membuktikannya.

   Tidak, objek dari Pegawai tidak bisa di-downcast ke Dosen jika objek asli tidak diinstansiasi sebagai Dosen. Downcasting akan menimbulkan ClassCastException saat runtime.