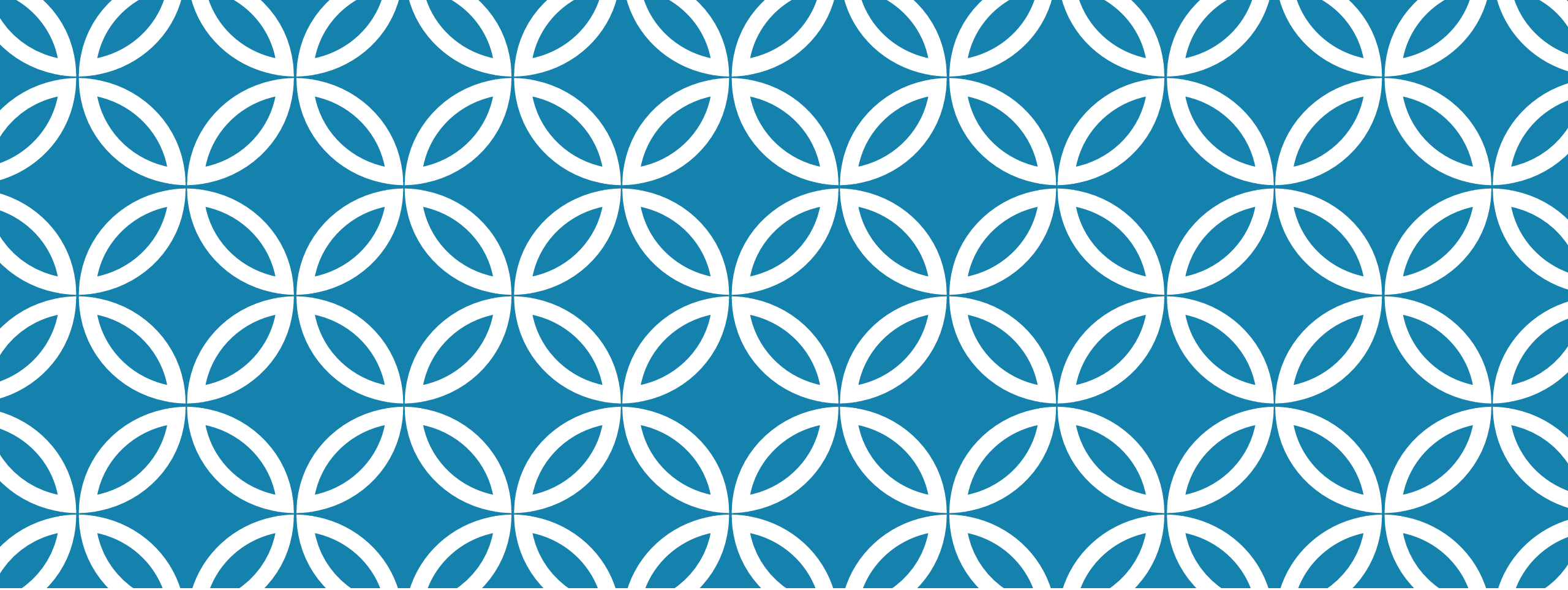


OVERRIDING & OVERLOADING (PART OF POLYMORPHISM)

POLYMORPHISM (DASAR)

Polimorfisme berasal dari bahasa Yunani yang artinya banyak bentuk. Dalam OOP, konsep ini memungkinkan penggunaan antarmuka yang sama untuk memerintahkan objek untuk melakukan tindakan atau aktivitas yang mungkin pada prinsipnya sama tetapi berbeda dalam proses.

Polimorfisme adalah kemampuan suatu metode untuk bekerja dengan lebih dari satu jenis argumen. Dalam bahasa lain, konsep ini sering disebut metode overloading. Pada dasarnya, Python tidak menangani ini secara khusus. Ini karena Python adalah bahasa pemrograman yang memiliki pengetikan dinamis, yang tidak memerlukan deklarasi tipe.

POLYMORPHISM (DASAR)

Polimorfisme merupakan suatu benda yang dapat memiliki berbagai bentuk, baik sebagai objek kelasnya sendiri maupun objek kelas supernya.

- Overloading: Menggunakan satu nama untuk beberapa metode berbeda (parameter berbeda)
- Overriding: terjadi ketika metode subclass dideklarasikan dengan nama dan parameter yang sama dengan metode superclass-nya.

OVERLOADING

Metode overloading berarti metode dengan nama yang sama, tetapi memiliki parameter berbeda, dan metode ini berada di kelas yang sama atau bisa di kelas lain yang terkait dalam hierarki pewarisan.

Karakteristik kelebihan beban:

- Nama metode yang sama.
- Daftar parameter (signature) berbeda.
- Jenis pengembalian bisa sama atau berbeda.

OVERLOADING EXAMPLE

```
public class Manusia
{
    public void setBiodata(String nama)
    {
        System.out.println("Method setBiodata yg pertama dipanggil");
    }

    public void setBiodata(String nama, int umur)
    {
        System.out.println("Method setBiodata yg kedua dipanggil");
    }

    public void setBiodata(String nama, int umur, String alamat)
    {
        System.out.println("Method setBiodata yg ketiga dipanggil");
    }
}
```

Nama metode sama, tetapi pengaturan tanda tangan atau parameter berbeda

Di kelas Manusia di atas, ada tiga metode setBiodata () tetapi dengan signature yang berbeda.

Perhatikan TestManusia berikut ini:

OVERLOADING EXAMPLE

Jika kelas Manusia di atas diuji dengan kelas Tes Manusia berikut:

```
public class TestManusia
{
    public static void main(String[] args)
    {
        Manusia man = new Manusia();
        man.setBiodata("Joko");
        man.setBiodata("Joko", 20, "Jakarta");
    }
}
```

Kemudian hasilnya:

```
D:\Java\Polimorfisme>java TestManusia
Method setBiodata yg pertama dipanggil
Method setBiodata yg ketiga dipanggil
```

CONSTRUCTOR : OVERLOADING

Konstruktor juga bisa Overloading

EXAMPLE: OVERLOADING OF A CONSTRUCTOR

```
public class Kucing
{
    public Kucing()
    {
        System.out.println("Konstruktor yg pertama dipanggil");
    }

    public Kucing(String jenis)
    {
        System.out.println("Konstruktor yg kedua dipanggil");
    }
}
```

Pada kelas Kucing (cat) di atas, terdapat dua konstruktor, tetapi dengan signature yang berbeda.

perhatikan Tes Kucing berikut:

EXAMPLE: OVERLOADING OF A CONSTRUCTOR

Jika kelas kucing di atas diuji dengan Tes Cat sebagai berikut:

```
public class TestKucing
{
    public static void main(String[] args)
    {
        Kucing jerry = new Kucing();
        Kucing garfield = new Kucing("Persian Mix");
    }
}
```

Kemudian hasilnya:

```
D:\Java\Polimorfisme>java TestKucing
Konstruktor yg pertama dipanggil
Konstruktor yg kedua dipanggil
```

WHEN DO WE USE OVERLOADING?

Untuk menyederhanakan kode program dan konsistensi dalam penamaan metode

Karena terkadang kita perlu membuat beberapa metode yang memiliki kegunaan yang sama, tetapi memiliki parameter yang berbeda

REAL WORLD USE OF OVERLOADING SCENARIOS

Misalnya bengkel sepeda motor bisa memperbaiki semua jenis sepeda motor. Namun, proses perbaikan berbeda-beda bergantung pada jenis motor yang ditangani.

```
public class BengkelMotor
{
    public void servis(MotorMatik mtk)
    {
        // lakukan proses perbaikan yang spesifik untuk motor matik
    }

    public void servis(MotorSport sprt)
    {
        // lakukan proses perbaikan yang spesifik untuk motor sport
    }
}
```

OVERRIDING

Overriding terjadi ketika ada metode dalam sub-kelas dengan nama dan tanda tangan yang sama dengan kelas induknya.

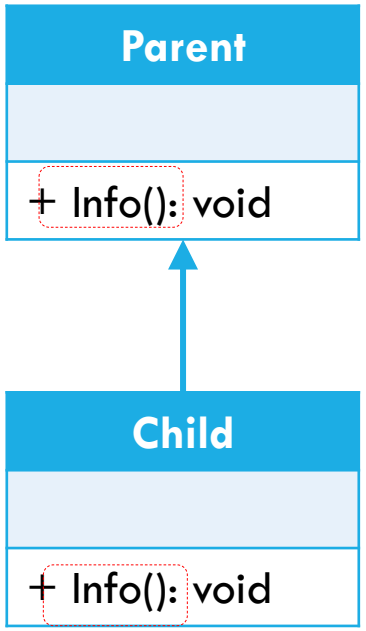
Karakteristik utama:

- Terjadi di sub kelas / kelas turunan
- Nama metode yang sama.
- Signature (pengaturan parameter) sama.
- Jenis pengembaliannya sama.

OVERRIDING

```
class Parent
{
    public void Info()
    {
        System.out.println("Ini milik class Parent");
    }
}
```

```
class Child extends Parent
{
    public void Info()
    {
        System.out.println("Ini milik class Child");
    }
}
```

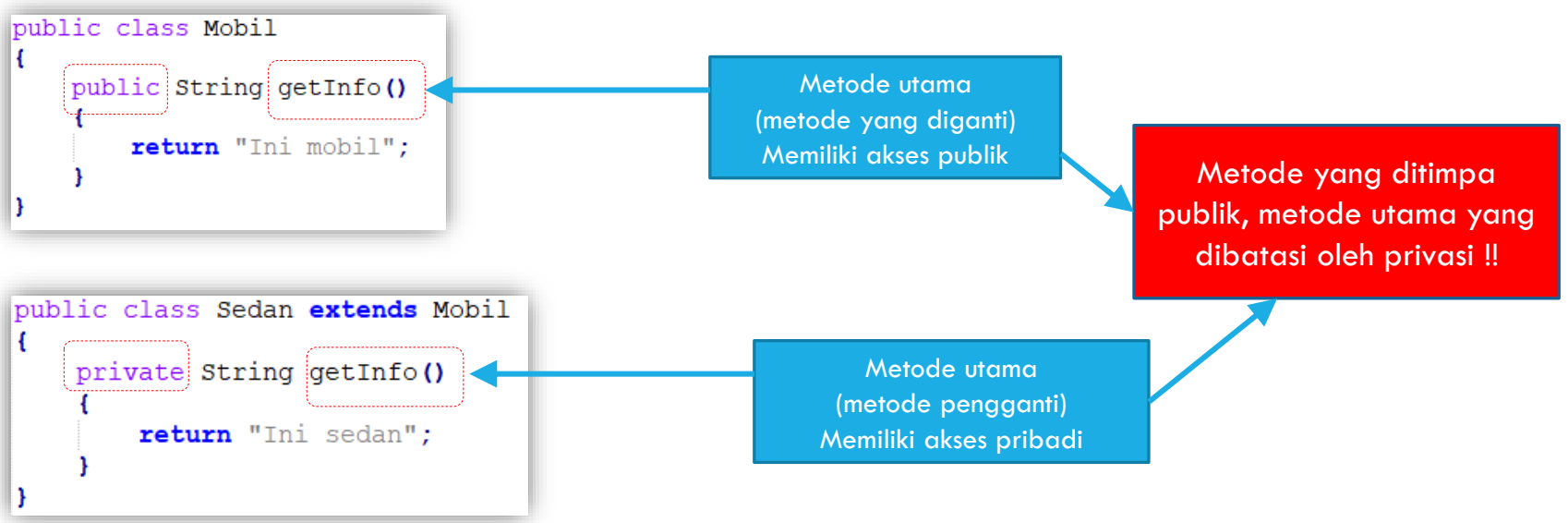


Pada contoh di atas, class Child merupakan turunan dari class Parent, baik di class Parent maupun class Child terdapat metode Info () yang memiliki signature yang sama persis.

Ini berarti bahwa metode Info () di kelas Parent diganti oleh kelas Anak

OVERRIDING

Metode yang diganti tidak dapat memiliki hak akses yang lebih luas daripada metode penggantian. Contoh:



CONTOH : OVERRIDING

```java
public class A
{
    public void tampilkanKeLayar()
    {
        System.out.println("Method milik class A dipanggil...");
    }
}
```

```java
public class B extends A
{
    public void tampilkanKeLayar()
    {
        System.out.println("Method milik class B dipanggil...");
    }
}
```

```java
public class TestOverriding
{
    public static void main(String[] args)
    {
        B objB = new B();
        objB.tampilkanKeLayar();
    }
}
```

```
D:\Java\Polimorfisme>java TestOverriding
Method milik class B dipanggil...
```

WHEN TO USE OVERRIDING

Overriding digunakan jika ada kelas yang diturunkan dari kelas lain, tetapi ingin mengubah fungsionalitas metode yang diwariskan, tanpa harus menambahkan metode baru.

REAL-WORLD OVERRIDING SCENARIOS

Misalnya ada kelas MobilListrik yang merupakan turunan dari kelas Mobil

Di dalam kelas Car, ada metode tambahKecepatan ()

Kemudian metode tambahKecepatan() juga diwarisi ke kelas MobilListrik, tetapi implementasinya berbeda.

tambahKecepatan () untuk kelas Mobil, implementasinya adalah dengan meningkatkan suplai bahan bakar dan udara ke mesin

tambahKecepatan () pada kelas MobilListrik, implementasinya adalah dengan meningkatkan tegangan listrik pada mesin

REAL-WORLD OVERRIDING SCENARIOS

```
public class Mobil
{
    public void tambahKecepatan()
    {
        // lakukan proses penambahan kecepatan
        // yaitu dgn menambah suplai bahan bakar dan udara ke mesin
    }
}
```

```
public class MobilListrik extends Mobil
{
    public void tambahKecepatan()
    {
        // lakukan proses penambahan kecepatan
        // yaitu dgn menambah tegangan listrik ke mesin
    }
}
```

Kita bisa melihat pada contoh di atas bahwa MobilListrik mengganti metode tambahKecepatan() dari kelas Mobil

EXAMPLE : OVERLOADING

Sebuah pompa bensin (SPBU) memiliki fitur pengisian otomatis. Jika mobil yang akan diisi adalah mobil mewah, maka isi dengan pertamax. Jika itu mobil tua, isi dengan pertalite. Buat program sederhana menggunakan overloading, uji dengan yang berikut:

```
public class TestGasStation
{
    public static void main(String[] args)
    {
        MobilMewah alphard = new MobilMewah();
        MobilKuno carry = new MobilKuno();
        GasStation gs = new GasStation();

        gs.isiBahanBakar(carry);
        gs.isiBahanBakar(alphard);
    }
}
```

```
run:
Mobil kuno telah diisi dengan Pertalite!
Mobil mewah telah diisi dengan Pertamax!
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXAMPLE: OVERLOADING (EXTENDED)

```java
public class MobilMewah
{

}
```

```java
public class MobilKuno
{

}
```

```java
public class GasStation
{
    public void isiBahanBakar(MobilMewah mw)
    {
        System.out.println("Mobil mewah telah diisi dengan Pertamax!");
    }

    public void isiBahanBakar(MobilKuno mk)
    {
        System.out.println("Mobil kuno telah diisi dengan Pertalite!");
    }
}
```

EXAMPLE : OVERRIDING

Sistem informasi kepegawaian, ada data untuk manajer dan supervisor. Supervisor adalah keturunan manajer. Manajer memiliki atribut nama dan gaji. Manajer memiliki metode cetak status yang mencetak semua atributnya. Semua kelas memiliki metode kenaikan gaji, yang menaikkan gaji dengan jumlah: manajer +1.000.000 dan supervisor + 1.500.000. Buat program, uji dengan yang berikut:

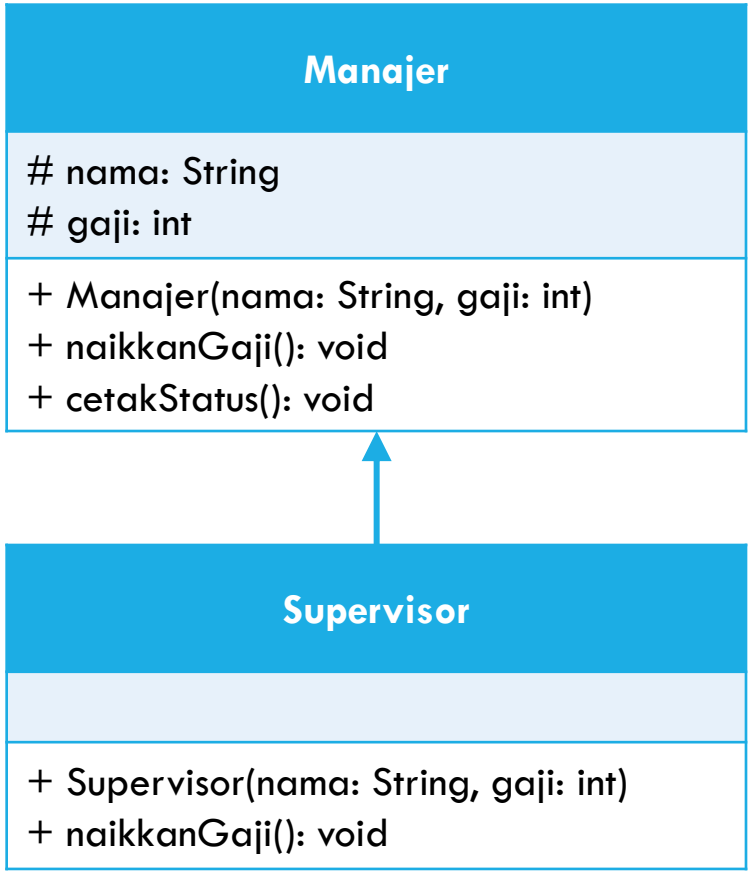
```
public class TestKepegawaian
{
    public static void main(String[] args)
    {
        Manajer man = new Manajer("Bill Gates", 5000000);
        Supervisor spv = new Supervisor("Susanto", 1000000);

        man.naikkanGaji();
        spv.naikkanGaji();

        man.cetakStatus();
        spv.cetakStatus();
    }
}
```

```
run:
Nama: Bill Gates
Gaji: 6000000
Nama: Susanto
Gaji: 2500000
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXAMPLE : OVERRIDING (EXTENDED)



EXAMPLE OVERRIDING (LANJUTAN)

```java
public class Manajer
{
    protected String nama;
    protected int gaji;

    public Manajer(String nama, int gaji)
    {
        this.nama = nama;
        this.gaji = gaji;
    }

    public void naikkanGaji()
    {
        gaji += 1000000;
    }

    public void cetakStatus()
    {
        System.out.println("Nama: " + nama);
        System.out.println("Gaji: " + gaji);
    }
}
```

```java
public class Supervisor extends Manajer
{
    public Supervisor(String nama, int gaji)
    {
        super(nama, gaji);
    }

    public void naikkanGaji()
    {
        gaji += 1500000;
    }
}
```