Responsi 10

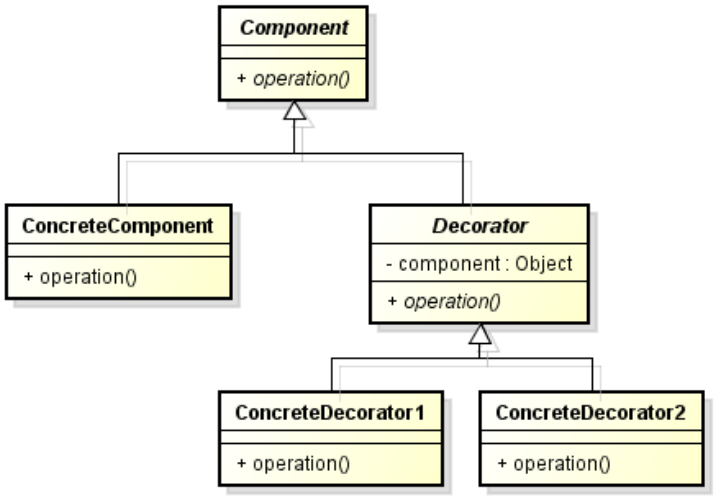
Analisis dan Desain Berorientasi Objek

Decorator Design Pattern

Decorator design pattern adalah salah satu bentuk dari structural pattern. Decorator design pattern memungkinkan adanya penambahan fungsional dari sebuah objek yang sudah ada tanpa perlu mengubah struktur dari objek tersebut. Pattern ini termasuk structural pattern karena bertindak sebagai wrapper untuk class yang sudah ada.

Struktur dari Decorator Design Pattern

Berikut ini adalah struktur class diagram untuk Decorator Design Pattern:



Setiap component selalu terkait dengan ConcreteComponent.operation(), namun belum tentu mengoperasikan ConcreteDecorator1.operation() dan/atau ConcreteDecorator2.operation().

R1001xxyyy

Saat ini kita akan mencoba mengimplementasikan Decorator Design Pattern untuk sebuah kasus sederhana yang sempat disebutkan di kelas sebagai contoh, yaitu decorator untuk pizza. (Contoh ini dibuat sangat sederhana karena bertujuan untuk “mencoba” decorator design pattern saja)

Seperti yang kita ketahui, pizza selalu terdiri dari basic pizza dan tambahan topping-toppingnya. Topping pizza dapat disesuaikan dengan selera pembeli. Semakin banyak topping yang dipilih, tentunya harga pizza tersebut semakin mahal (walau rasanya belum tentu lebih enak ☺).

Saat ini kita diminta bantuan oleh pemilik restoran pizza untuk membuat program penghitung harga pizza. (Sebagai imbalannya, Anda diperkenankan makan pizza gratis di restoran itu selama pembangungan program tersebut ☺). Karena restoran ini masih baru, hanya terdapat tiga topping, yaitu jamur, ayam, dan pepperoni. Harga plain pizza pada restoran tersebut adalah 30.000. Harga untuk mushroom topping adalah 10.000. Harga untuk chicken topping adalah 15.000. Harga untuk pepperoni topping adalah 20.000.

Buatlah abstract class `Pizza` dengan satu abstract method, yaitu `getPrice`():

```
public abstract class Pizza {
    public abstract double getPrice();
}
```

Implementasikan class untuk plain pizza:

```
public class PlainPizza extends Pizza{
    @Override
    public double getPrice() {
        return 30000;
    }
}
```

Buatlah juga kelas-kelas lain untuk mengimplementasikan penambahan dan pemilihan topping dari pizza tersebut. Buatlah juga kelas `Tester` untuk menguji penggunaan berbagai topping. Kelas Tester **minimal** menghasilkan output seperti berikut:

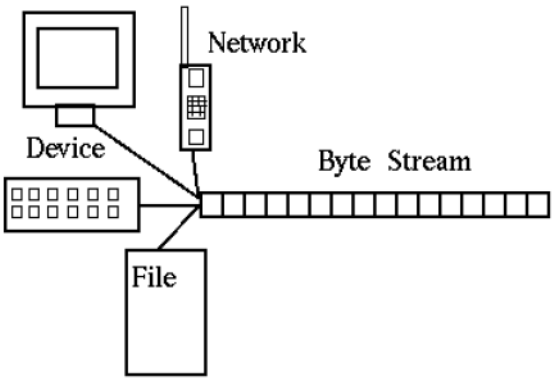
```
Plain Pizza 30000.0
Pizza with mushroom topping 40000.0
Pizza with chicken and mushroom toppings 55000.0
Pizza with chicken, mushroom, and pepperoni toppings 75000.0
```

Buatlah juga class diagram untuk soal tersebut (R1001xxyyy.jpg).

InputStream Decorator

Seterlah mencoba kasus sedarhana, mari coba kita implementasikan decorator design pattern untuk hal yang lebih serius.

Salah satu contoh dari decorator pattern yang sering dipakai adalah Stream. Stream merupakan sebuah abstraksi sekuens data bit yang mengalir dari entitas sumber ke entitas tujuan. Entitas-entitas yang dimaksud dapat berupa **files** pada hard disk, **program** lain, **komputer** lain pada suatu jaringan, **peralatan input** (keyboard, mouse, dll), **halaman web**, dan sebagainya. Ilustrasi mengenai stream dapat dilihat pada gambar di bawah ini:



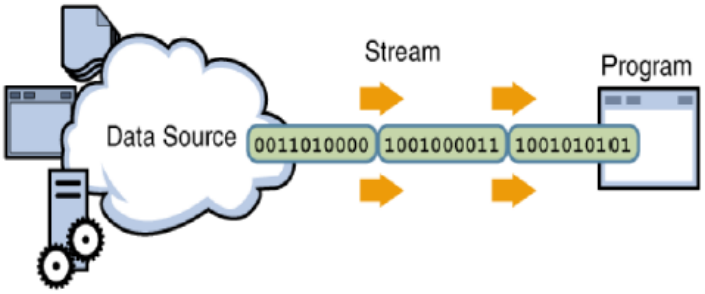
Secara umum, Java membagi stream dalam 2 kategori, yaitu ***byte stream*** dan ***character stream***. Dalam responsi kali ini, kita hanya akan belajar sekilas cara penggunaan *byte stream*.

BYTE STREAM

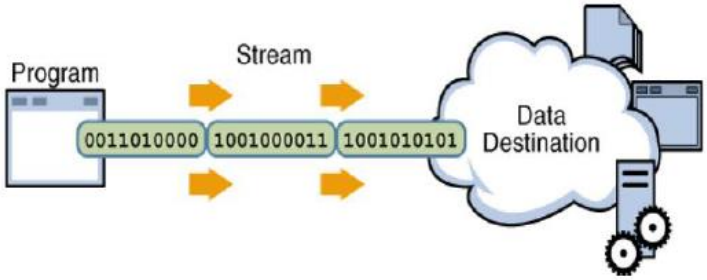
Byte stream merepresentasikan stream yang dikirim oleh input lalu ditangkap oleh output dalam bentuk deretan per 8-bit. Byte stream cocok digunakan apabila sedang melakukan *stream* terhadap data *binary*.

Terdapat 2 kelas abstract (terdapat di package `java.io`) yang merepresentasikan input dan output yang akan memanfaatkan byte stream, yaitu

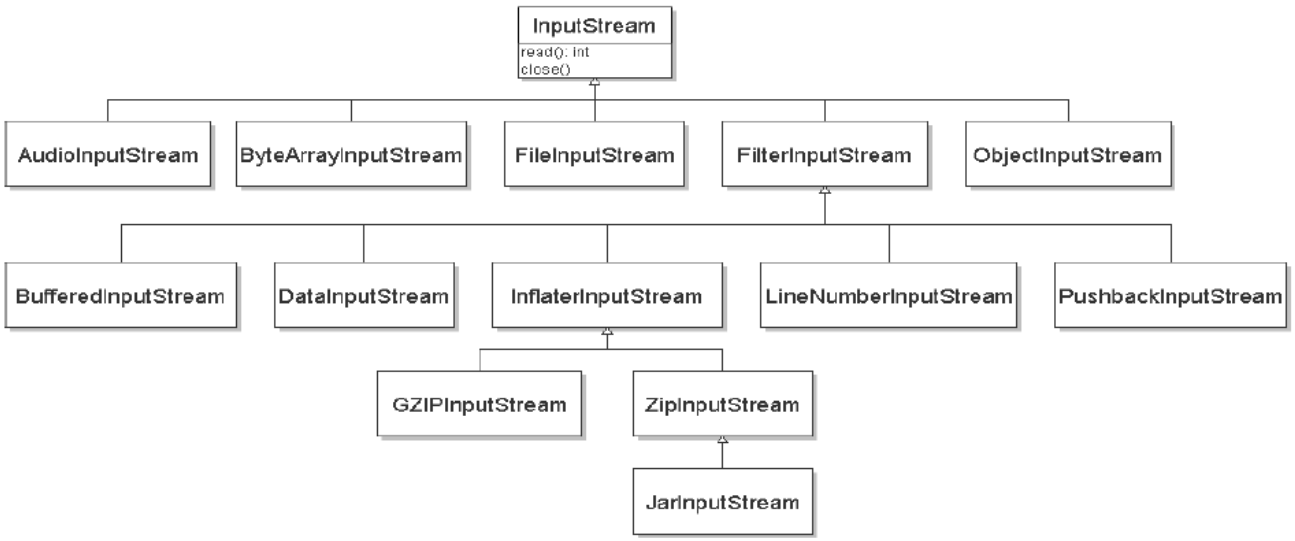
- **InputStream**
  Program menggunakan input stream untuk membaca data dari *source*, 1 item dalam 1 waktu



- **OutputStream**
  Program menggunakan input stream untuk menuliskan data ke *destination*, 1 item dalam 1 waktu



Untuk memanfaatkan byte stream, kita dapat menggunakan kelas-kelas lainnya yang meng-*extend* kedua kelas tersebut. Terdapat banyak kelas yang dimaksud yang dapat kita gunakan sesuai kebutuhan, bergantung dari jenis input dan output-nya. Berikut ini merupakan diagram yang menggambarkan berbagai kelas turunan dari kelas InputStream:



Pada responsi kali ini, kita akan fokus ke salah satu kelas stream yang menangani pembacaan dan penulisan terhadap file, yaitu kelas ***FileInputStream***.

Berikut diberikan sebuah contoh program sederhana untuk membaca isi dari sebuah file text.

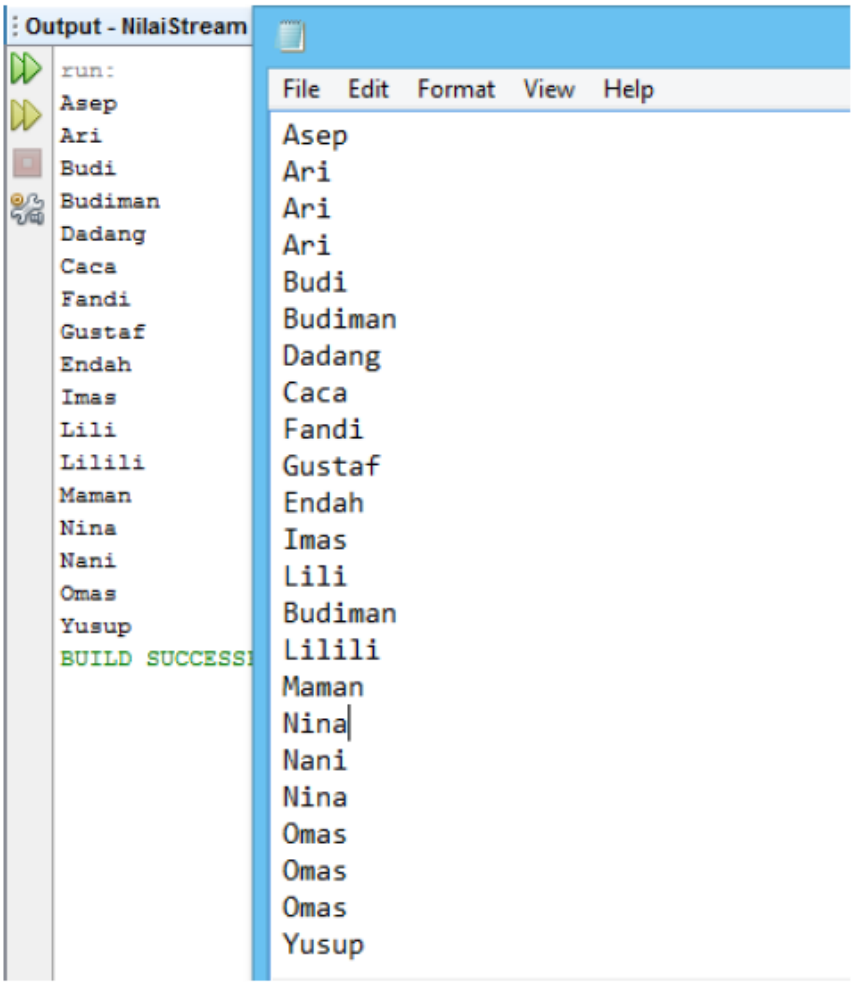
```java
public class ByteStream {

    public static void main(String[] args) throws IOException {
        FileInputStream in = null;
        try {
            in = new FileInputStream("xanadu.txt");
            int c;
            while ((c = in.read()) != -1) {
                System.out.print((char) c);
            }
        } catch (FileNotFoundException ex) {
            System.err.println(ex.getMessage());
        } finally {
            if (in != null) {
                in.close();
            }
        }
    }
}
```

*) Bila ingin mencoba, telah diberikan file xanadu.txt, jika anda ingin mencoba kode dibawah dengan berbasis project, letakkan file tersebut sejajar dengan (build, src)

R1002xxyyy

Diberikan sebuah file dengan nama karyawan.txt. Didalamnya terdapat nama-nama dan banyak yang terduplikasi. Tugas anda adalah menampilkan ke layar tanpa adanya duplikasi nama.
Anda diperbolehkan untuk menambah kelas baru sesuai kebutuhan, dengan syarat tidak boleh mengubah kelas yang telah ada (Tester.java).



Kiri (Hasil output) || Kanan (isi dari file karyawan)

Tidak perlu dilakukan sort pada hasil. Anda cukup untuk tidak mengeprint bila namanya telah muncul di baris sebelumnya.

*R1002 disadur dari soal tahun lalu