

## **Modul 13**

### **Design Pattern 2 (Factory & Singleton)**

#### **Tujuan**

1. Praktikan mampu memahami konsep creational pattern
2. Praktikan mampu memahami konsep factory & singleton
3. Praktikan mampu mengimplementasikan pemahaman tersebut ke dalam penyelesaian kasus tertentu

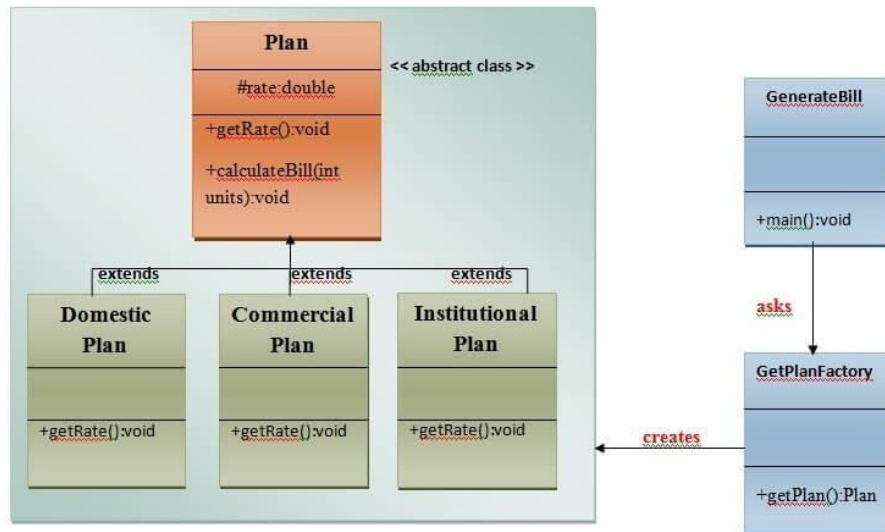
#### **Pembahasan**

Creational Pattern memudahkan kita dalam membuat instans atau objek dari suatu kelas yang cocok dalam kasus atau situasi tertentu. Terdapat beberapa jenis creational pattern antara lain: Abstract Factory, Builder, Factory, Object Pool, Prototype, Singleton. Pada praktikum kali ini akan dibahas mengenai Factory dan Singleton.

##### **1. Factory**

Pattern Factory merupakan pattern yang digunakan untuk menciptakan sebuah objek. Pattern ini menyatakan “mendefinisikan sebuah antar muka untuk penciptaan objek, dengan sebuah kelas yang akan memutuskan kelas turunan mana yang akan diinstantiasi”. Dalam membuat objek dengan pattern ini, kita tidak dihadapkan dengan logik bagaimana objek dibuat, melainkan penciptaan objek dilakukan melalui sebuah antarmuka.

Factory bekerja dengan cara menentukan pembuatan instans atau objek dari beberapa kelas turunan. Kelas turunan ini nantinya bisa merupakan kelas-kelas yang diturunkan dari kelas abstrak, kelas konkret atau kelas-kelas yang mengimplementasikan interface yang sama. Kemudian akan ada sebuah method di dalam sebuah kelas Factory, yang akan menentukan objek dari kelas turunan mana yang akan dibuat. Berikut contoh factory pattern



Pada contoh tersebut terdapat beberapa paket pembayaran (Domestic, Commercial, Institutional) yang bisa dipilih saat melakukan pembayaran (GenerateBill). Main class (GenerateBill) hanya perlu mengirimkan jenis objek paket pembayaran yang diinginkan ke Factory (GetPlanFactory) tanpa perlu mengetahui bagaimana objek tersebut dibuat. Penentuan objek paket pembayaran yang diciptakan, dilakukan di dalam kelas GetPlanFactory. Method getPlan pada GetPlanFactory akan mengembalikan objek kelas turunan dari kelas abstrak Plan yang berhasil diciptakan.

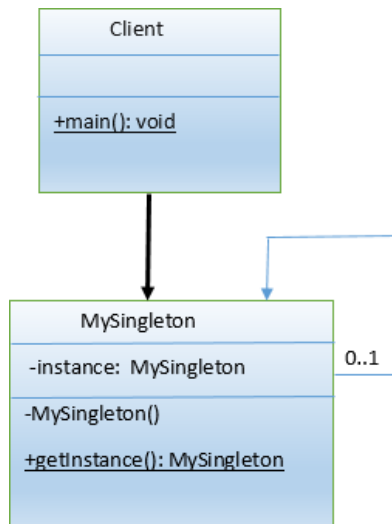
## 2. Singleton

Singleton memastikan suatu kelas hanya bisa memiliki 1 (satu) instans atau objek saja. Singleton menyediakan poin akses terhadap instans atau objek tersebut melalui sebuah method. Sebuah kelas harus memastikan bahwa hanya ada satu objek kelas tersebut yang diciptakan dan dapat digunakan oleh kelas-kelas lain. Supaya sebuah kelas dapat menerapkan pattern Singleton maka kelas tersebut harus:

- Mendeklarasikan variabel bertipe variabel static
- Menyembunyikan konstruktor kelas, untuk menghindari instantiasi objek kelas singleton dari luar kelas singleton tersebut.

- Mendefinisikan sebuah method statik yang mengembalikan objek tunggal dari sebuah kelas singleton. Method ini sekaligus menyediakan akses global terhadap objek kelas singleton.

Berikut contoh singleton pattern.

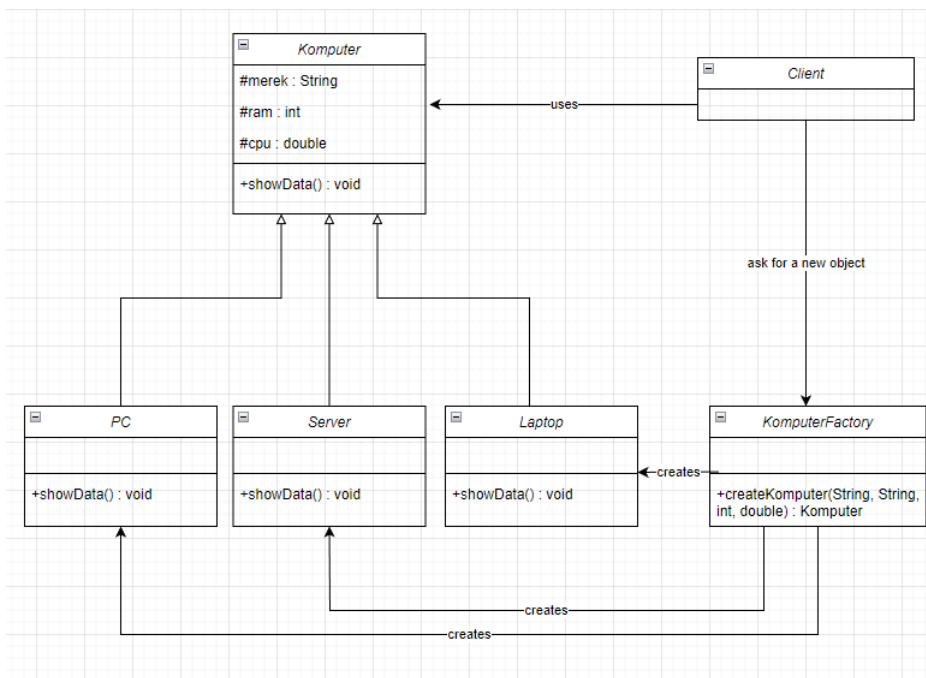


- Pada contoh tersebut terdapat variable static bernama “instance” bertipe MySingleton. Karena static maka “instance” merupakan class variable yang artinya digunakan bersama di seluruh objek MySingleton.
- Pada contoh tersebut konstruktor MySingleton memiliki visibilitas private yang artinya konstruktor tersebut hanya bisa diakses di dalam kelas MySingleton sehingga dapat menghindari instantiasi objek di luar kelas MySingleton.
- Pada contoh tersebut terdapat sebuah method static getInstance yang mengembalikan objek tunggal MySingleton yaitu atribut “instance”. Method tersebut juga memiliki visibilitas public sehingga bisa diakses dari mana saja (menyediakan akses global).

## Guided

### a. Factory

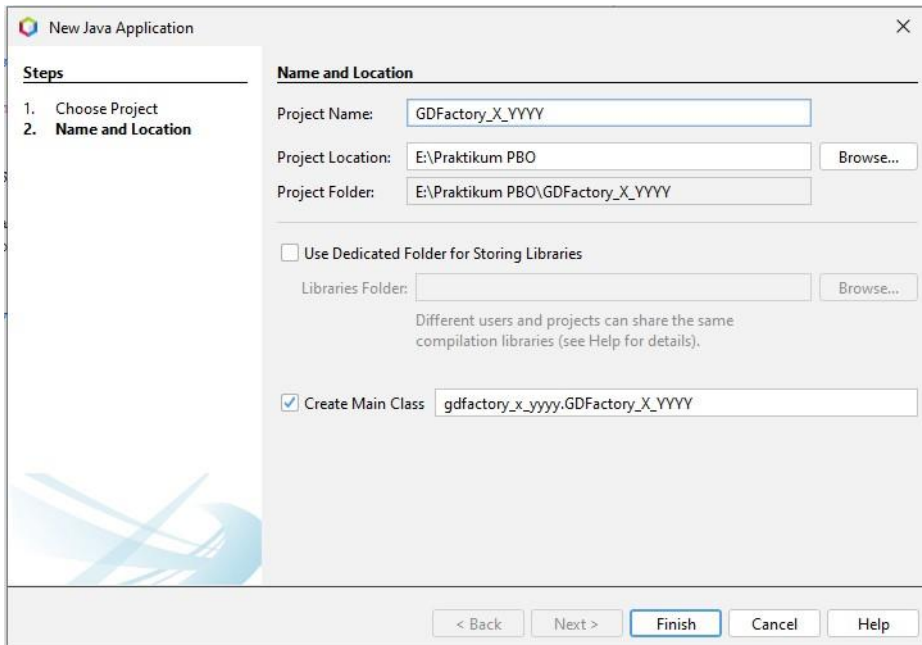
Pada Guided Factory ini, terdapat 3 jenis komputer yaitu PC, Server, dan Laptop. Kita akan menggunakan Factory pattern untuk membuat objek dari PC, Server, dan Laptop. Kemudian kita



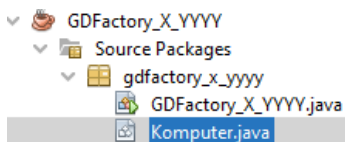
akan menampilkan data dari objek yang berhasil dibuat.

PC, Server, dan Laptop merupakan kelas turunan dari kelas abstrak Komputer. KomputerFactory adalah kelas yang berisikan method `createKomputer` yang bertugas menentukan objek dari kelas turunan mana yang akan diinstantiasi. Client (main class) akan menggunakan variabel bertipe kelas Komputer untuk menampung objek yang dikembalikan oleh method `createKomputer` dari kelas KomputerFactory.

1. Buat proyek baru dengan nama GDFactory\_X\_YYYY (X = Kelas, Y= 4 digit terakhir NPM)



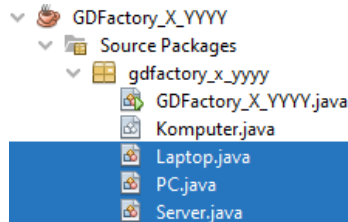
2. Buat sebuah abstract class dengan nama “Komputer”



isikan dengan code berikut

```
12 public abstract class Komputer {
13     protected String merek;
14     protected int ram;
15     protected double cpu;
16
17     public Komputer(String merek, int ram, double cpu) {
18         this.merek = merek;
19         this.ram = ram;
20         this.cpu = cpu;
21     }
22
23     public abstract void showData();
24 }
```

3. Buat 3 class baru dengan nama “PC”, “Server”, “Laptop”



isikan masing-masing kelas dengan code berikutPC.java

```
11 public class PC extends Komputer {
12     public PC(String merek, int ram, double cpu) {
13         super(merek, ram, cpu);
14     }
15
16     @Override
17     public void showData() {
18         System.out.println("\n=== PC ===");
19         System.out.println("Merek : " + this.merek);
20         System.out.println("RAM : " + this.ram + " GB");
21         System.out.println("CPU : " + this.cpu + " GHz");
22     }
23 }
```

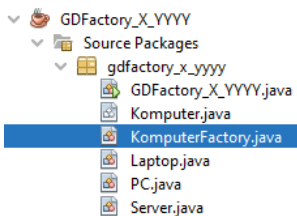
Server.Java

```
11 public class Server extends Komputer {
12     public Server(String merek, int ram, double cpu) {
13         super(merek, ram, cpu);
14     }
15
16     @Override
17     public void showData() {
18         System.out.println("\n=== Server ===");
19         System.out.println("Merek : " + this.merek);
20         System.out.println("RAM : " + this.ram + " GB");
21         System.out.println("CPU : " + this.cpu + " GHz");
22     }
23 }
```

## Laptop.java

```
11 public class Laptop extends Komputer {
12     public Laptop(String merek, int ram, double cpu) {
13         super(merek, ram, cpu);
14     }
15
16     @Override
17     public void showData() {
18         System.out.println("\n=== Laptop ===");
19         System.out.println("Merek : " + this.merek);
20         System.out.println("RAM : " + this.ram + " GB");
21         System.out.println("CPU : " + this.cpu + " GHz");
22     }
23 }
```

## 4. Buat sebuah class baru dengan nama “KomputerFactory”



isikan dengan code berikut

```
11 public class KomputerFactory {
12     public Komputer createKomputer(String jenis, String merek, int ram, double cpu) {
13         if(jenis.equalsIgnoreCase("PC")) {
14             return new PC(merek, ram, cpu);
15         } else if(jenis.equalsIgnoreCase("Server")) {
16             return new Server(merek, ram, cpu);
17         } else if(jenis.equalsIgnoreCase("Laptop")) {
18             return new Laptop(merek, ram, cpu);
19         }
20         return null;
21     }
22 }
```

## 5. Isi main class dengan code berikut

```
11 public class GDFactory_X_YYYY {
12
13     public static void main(String[] args) {
14         // TODO code application logic here
15         KomputerFactory kf = new KomputerFactory();
16
17         Komputer komputer1 = kf.createKomputer("PC", "Lenovo", 8, 3.4);
18         Komputer komputer2 = kf.createKomputer("Server", "HP", 16, 4.2);
19         Komputer komputer3 = kf.createKomputer("Laptop", "Asus", 16, 3.6);
20
21         komputer1.showData();
22         komputer2.showData();
23         komputer3.showData();
24     }
25 }
```

## 6. Output

```
Output - GDFactory_X_YYYY (run) X
run:

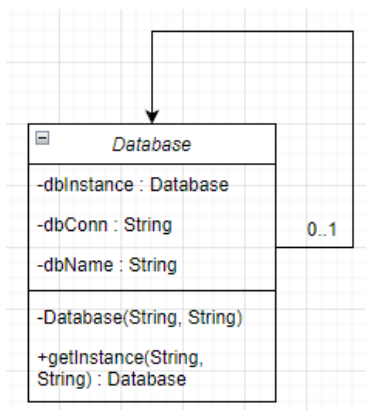
=== PC ===
Merek : Lenovo
RAM : 8 GB
CPU : 3.4 GHz

=== Server ===
Merek : HP
RAM : 16 GB
CPU : 4.2 GHz

=== Laptop ===
Merek : Asus
RAM : 16 GB
CPU : 3.6 GHz
BUILD SUCCESSFUL (total time: 0 seconds)
```

### b. Singleton

Kasus Guided Singleton yaitu untuk menghubungkan aplikasi ke database maka dibuat sebuah kelas Database. Untuk memastikan hanya bisa ada 1 (satu) koneksi dari aplikasi ke



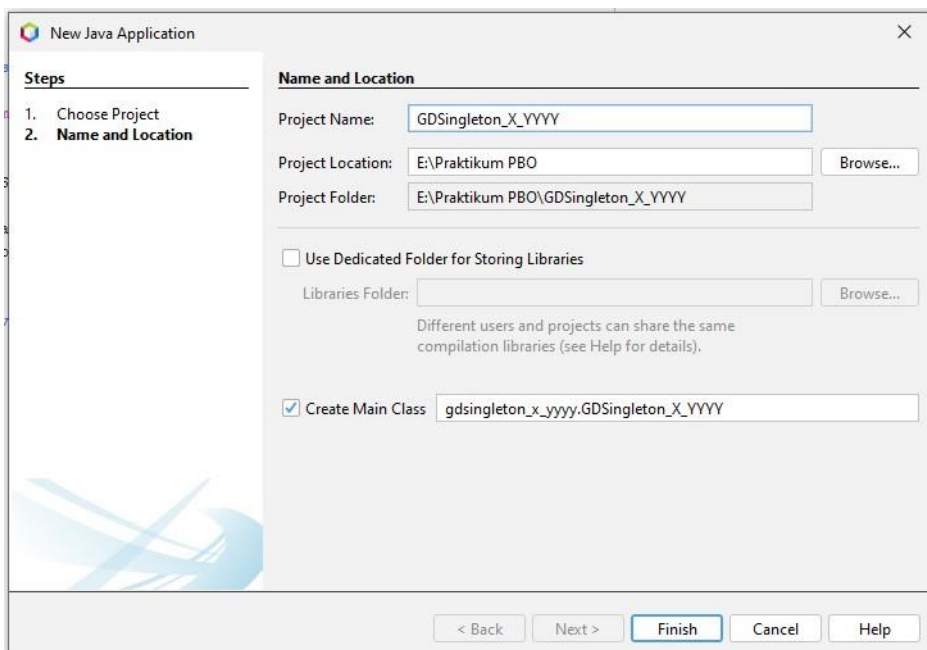
database maka kelas Database dibuat menggunakan Singleton pattern.

Kelas Database memiliki atribut `dbInstance`, `dbConn`, `dbName`. Atribut `dbInstance` memiliki keyword `static` sehingga menjadi class variabel dan hanya bisa 1 objek yang dapat

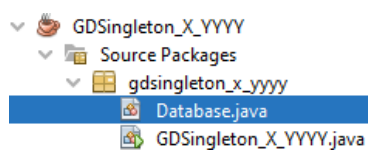


diciptakan dari kelas Database. Kelas Database memiliki konstruktor dengan visibilitas private sehingga konstruktor tersebut tidak dapat diakses dari manapun kecuali dari dalam kelas Database untuk menghindari instantiasi objek di luar kelas Database. Kelas Database memiliki method getInstance untuk mengembalikan objek Database yang berhasil diciptakan. Method getInstance memiliki visibilitas public yang artinya method tersebut dapat diakses dari mana saja sehingga method tersebut menjadi poin akses global.

1. Buat projek baru dengan nama GDSingleton\_X\_YYYY (X = Kelas, Y= 4 digit terakhir NPM)



2. Buat class baru dengan nama “Database”



isikan dengan code berikut

```

11 public class Database {
12     private static Database dbInstance = null;
13
14     private String dbConn, dbName;
15
16     private Database(String dbConn, String dbName) {
17         this.dbConn = dbConn;
18         this.dbName = dbName;
19     }
20
21     public static Database getInstance(String dbConn, String dbName) {
22         System.out.println("\nMembuat objek Database baru...");
23
24         if(Database.dbInstance == null) {
25             Database.dbInstance = new Database(dbConn, dbName);
26             System.out.println("Objek Database berhasil dibuat");
27             System.out.println("Connection : " + dbConn);
28             System.out.println("DB Name : " + dbName);
29         } else {
30             System.out.println("Objek Database gagal dibuat karena sudah");
31             System.out.println("ada objek Database yang dibuat sebelumnya");
32             System.out.println("Connection : " + dbConn);
33             System.out.println("DB Name : " + dbName);
34         }
35
36         return Database.dbInstance;
37     }
38 }

```

pada method `getInstance` terdapat percabangan dengan kondisi `Database.dbInstance == null`. Jika `dbInstance` bernilai `null` artinya belum ada instans atau objek dari singleton `Database` sehingga bisa dibuat objeknya. Jika masuk ke `else` berarti sudah ada instans atau objek dari singleton `Database` dan akan menampilkan pesan bahwa “Objek Database gagal dibuat karena sudah ada objek Database yang dibuat sebelumnya”. Jadi objek Singleton akan gagal dibuat jika sudah ada instans atau objek dari Singleton tersebut.

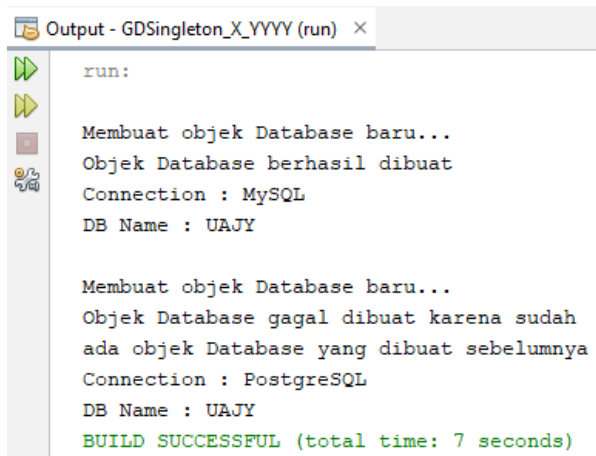
3. Isi main class dengan code berikut

```

11 public class GDSingleton_X_YYYY {
12
13     public static void main(String[] args) {
14         Database db1 = Database.getInstance("MySQL", "UAJY");
15         Database db2 = Database.getInstance("PostgreSQL", "UAJY");
16     }
17 }

```

#### 4. Output



```
run:

Membuat objek Database baru...
Objek Database berhasil dibuat
Connection : MySQL
DB Name : UAJY

Membuat objek Database baru...
Objek Database gagal dibuat karena sudah
ada objek Database yang dibuat sebelumnya
Connection : PostgreSQL
DB Name : UAJY

BUILD SUCCESSFUL (total time: 7 seconds)
```

#### Pengumpulan

Masukkan **GDFactory\_X\_YYYY** dan **GDSingleton\_X\_YYYY** ke dalam 1 folder dengan nama **GD13\_X\_YYYY** kemudian di zip menjadi **GD13\_X\_YYYY.zip**