MODUL 13

DESIGN PATTERN II

Pertemuan lalu kita sudah pelajari **Design Pattern I** (Observer dan Composite). Pertemuan ini kita akan mempelajari 2 lagi design pattern, yakni **Factory** dan **Singleton**. Kedua design pattern ini digunakan untuk **mengkonstruksi sebuah obyek baru**, sehingga disebut **creational patterns**.

1. FACTORY

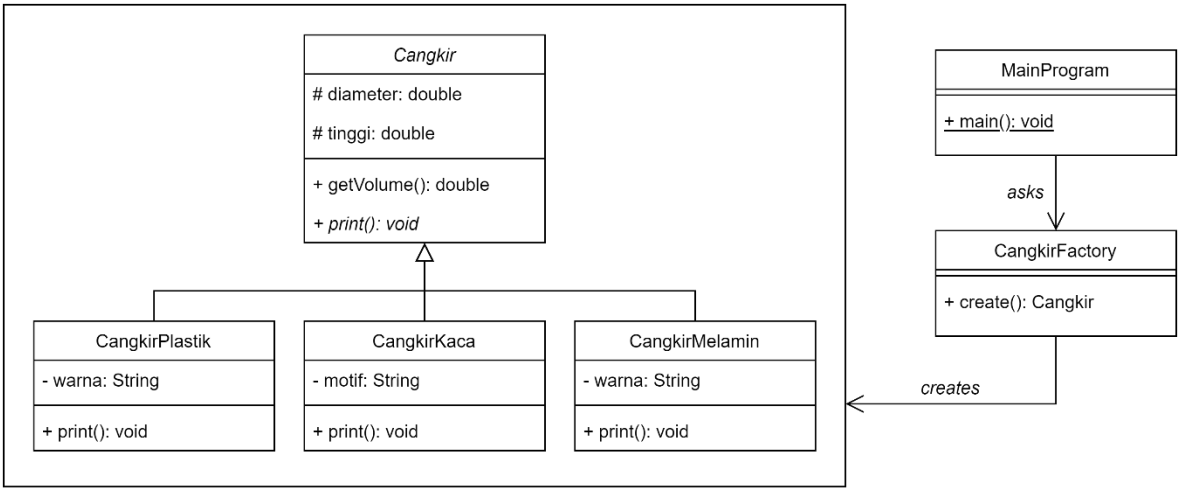
Ilustrasi dari https://topdev.vn/blog/huong-dan-su-dung-factory-trong-design-pattern/

Penjelasan

Pada dasarnya, "factory" berarti "pabrik", dan biasanya sebuah "pabrik" akan menghasilkan output yang kurang lebih seragam. Contohnya sebuah pabrik cangkir yang menghasilkan cangkir plastik, cangkir kaca, dan cangkir melamin.

Design pattern factory dalam pengertiannya juga kurang lebih sama dengan pabrik-pabrik yang kita temui dalam kehidupan sehari-hari. Dalam implementasinya, kita akan **membuatkan sebuah *factory class*** yang **digunakan untuk mengkonstruksi obyek baru** yang merupakan turunan (*sub class*) dari sebuah kelas konkret atau kelas abstrak (*super class*), maupun kelas yang mengimplementasikan *interface* yang sama.

Sama seperti pabrik di dunia nyata, **kita tidak perlu tahu proses apa saja yang terjadi di dalam pabrik tersebut**, yang penting **output yang dihasilkan sudah sesuai dengan keinginan kita**.



Dari contoh diagram kelas di atas, sebuah factory pattern memiliki struktur:

Struktur	**Nama Class/Interface**
Abstract Product	`Cangkir`

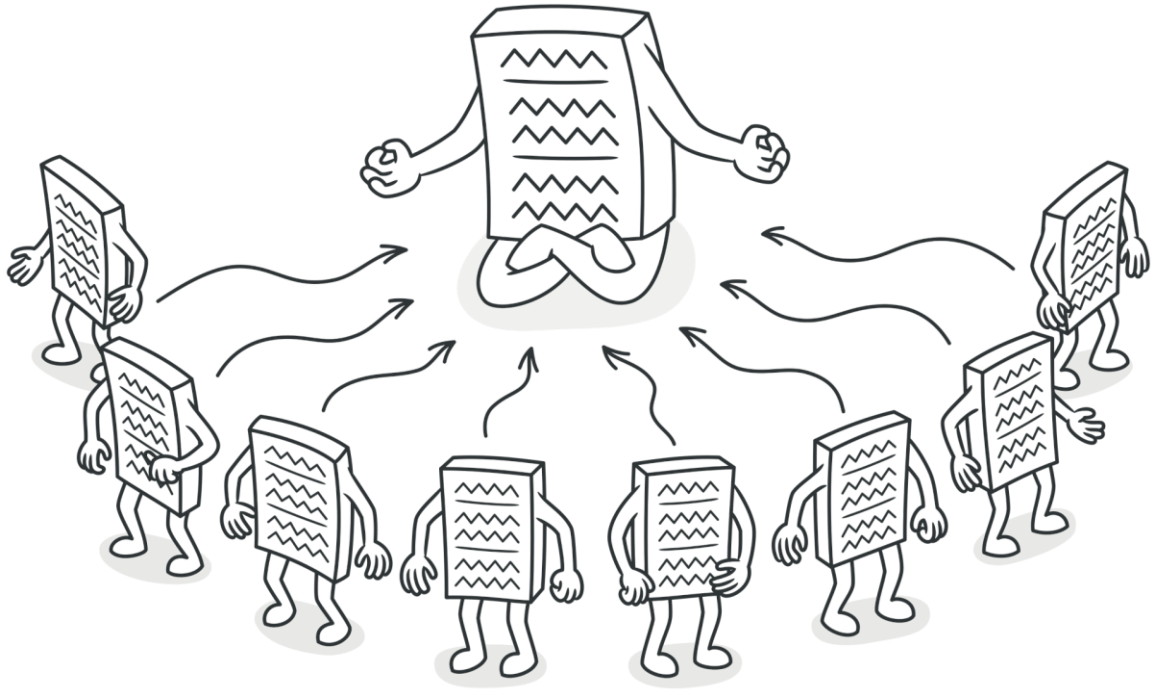
Concrete Product	`CangkirPlastik`
	`CangkirKaca`
	`CangkiirMelamin`

Factory	`CangkirFactory`

Pada contoh di atas, terdapat beberapa cangkir yang dapat diciptakan (`CangkirPlastik`, `CangkirKaca`, dan `CangkirMelamin`). Program utama (`MainProgram`) bisa meminta factory (`CupFactory`) untuk menciptakan objek baru yang sesuai dengan yang diinginkan user tanpa perlu mengetahui cara objek tersebut dibuat.

Method `create()` pada `CupFactory` akan mengembalikan salah satu dari ketiga cangkir yang dapat diciptakan (`CangkirPlastik`, `CangkirKaca`, dan `CangkirMelamin`).

2. SINGLETON



Ilustrasi: https://refactoring.guru/design-patterns/singleton

Penjelasan

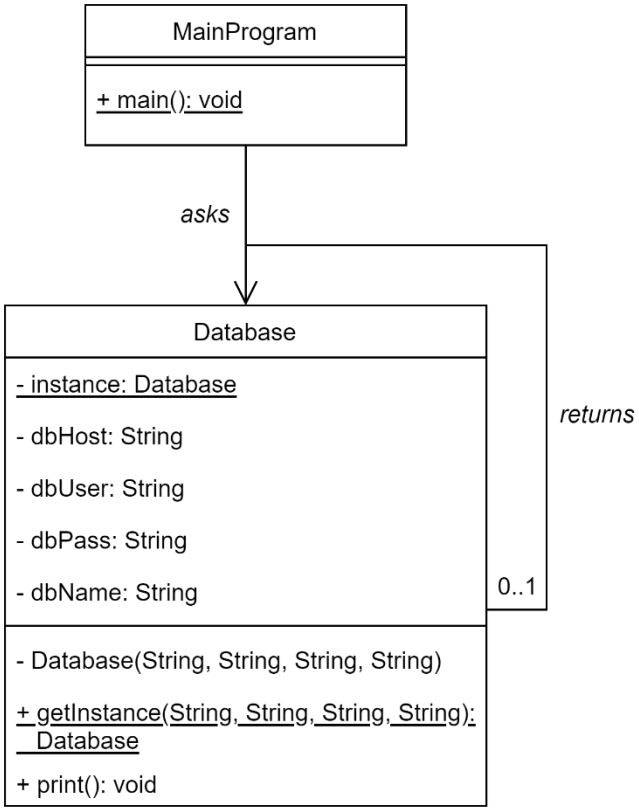
Singleton merupakan sebuah design pattern yang memastikan bahwa **suatu kelas hanya memiliki satu *instance*/obyek**. Kelas singleton hanya memberikan **satu method yang digunakan untuk mengakses objek** kelas tersebut.

Dalam kehidupan sehari-hari, bisa saja kita menganalogikan kelas singleton dengan **sesuatu yang digunakan bersama-sama oleh beberapa orang**. Misalnya menggunakan bersama satu mesin pencuci oleh satu keluarga, adanya hanya satu bentuk pemerintahan di dalam satu negara, dan sebagainya.

Supaya sebuah kelas memenuhi prinsip design pattern singleton, maka kelas tersebut harus:

- Memiliki **static member**: hanya mengalokasikan memori untuk satu *instance*/obyek. Obyek tersebut harus merupakan **variabel kelas** (*class* variable, menggunakan keyword `static`).
- Memiliki **private constructor**: menghindari pembuatan obyek baru dari luar kelas itu sendiri. Constructor kelas singleton harus menggunakan keyword `private`.

- Memiliki **static method**: satu method yang menyediakan akses terhadap obyek singleton. Method ini harus merupakan **method kelas** (*class method*, menggunakan keyword `static`).



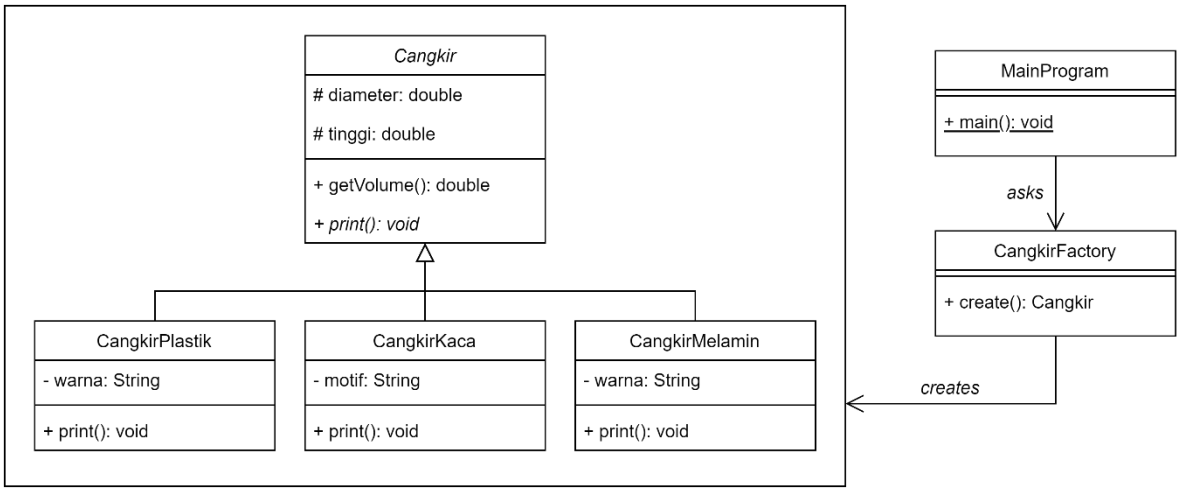
Di atas merupakan salah satu contoh penerapaan singleton, yakni pada **class Database**. Secara umum, class Database biasanya digunakan untuk mengakses dan mengelola data dari sebuah basis data (seperti SQL atau NoSQL), sehingga secara umum class Database sebaiknya dituliskan menggunakan design pattern singleton untuk menghindari risiko duplikasi atau konflik data.

Beberapa prinsip design pattern singleton sudah kita terapkan di atas, yakni:

- **Static member:** ada 1 variabel untuk menampung instance/obyek yang dibuatkan, yakni atribut bernama `instance`, bersifat private (tanda "-") dan static (digarisbawahi).
- **Private constructor:** pada bagian method di diagram UML di atas, kita menuliskan tanda "-" pada constructor `Database` yang artinya visibilitas constructor sudah private.
- **Static method:** ada 1 method untuk mengembalikan instance Database yang ditampung: `getInstance(...)`, bersifat public (tanda "+") dan static (digarisbawahi).

GUIDED FACTORY

Di guided ini kita akan membuatkan implemetasi dari diagram kelas yang sudah kita lihat di modul Factory di atas:



Langkah-langkah:

1. Buka NetBeans dan buatkan project baru (Java with Ant → Java Application), namanya bebas, yang penting kalian ingat :)

   Kita akan membuatkan berbagai class dengan urutan:

   - Cangkir *(abstract product)*
     - CangkirPlastik *(concrete product)*
     - CangkirKaca *(concrete product)*
     - CangkirMelamin *(concrete product)*
       - CangkirFactory *(factory)*
       - MainProgram (program utama)

2. Buatlah sebuah *abstract class* baru bernama **Cangkir**, isikan dengan code berikut:

```java
public abstract class Cangkir {
    protected double diameter;
    protected double tinggi;

    public Cangkir(double diameter, double tinggi) {
        this.diameter = diameter;
        this.tinggi = tinggi;
    }

    public double getVolume() {
        return Math.PI * Math.pow(diameter / 2, 2) * tinggi;
    }

    abstract public void print();
}
```

3. Buatlah sebuah *class* baru bernama **CangkirPlastik**, isikan dengan code berikut:

```
public class CangkirPlastik extends Cangkir {
    private String warna;

    public CangkirPlastik(double diameter, double tinggi, String warna) {
        super(diameter, tinggi);
        this.warna = warna;
    }

    @Override
    public void print() {
        System.out.println("Cangkir Plastik");
        System.out.println("Diameter  : " + diameter);
        System.out.println("Tinggi    : " + tinggi);
        System.out.println("Warna     : " + warna);
        System.out.println("Volume    : " + getVolume());
    }
}
```

4. Buatlah sebuah *class* baru bernama **CangkirKaca**, isikan dengan code berikut:

```
public class CangkirKaca extends Cangkir {
    private String motif;

    public CangkirKaca(double diameter, double tinggi, String motif) {
        super(diameter, tinggi);
        this.motif = motif;
    }

    @Override
    public void print() {
        System.out.println("Cangkir Kaca");
        System.out.println("Diameter  : " + diameter);
        System.out.println("Tinggi    : " + tinggi);
        System.out.println("Motif     : " + motif);
        System.out.println("Volume    : " + getVolume());
    }
}
```

5. Buatlah sebuah *class* baru bernama **CangkirMelamin**, isikan dengan code berikut:

```
public class CangkirMelamin extends Cangkir {
    private String warna;

    public CangkirMelamin(double diameter, double tinggi, String warna) {
        super(diameter, tinggi);
        this.warna = warna;
    }

    @Override
    public void print() {
        System.out.println("Cangkir Melanin");
        System.out.println("Diameter  : " + diameter);
        System.out.println("Tinggi    : " + tinggi);
        System.out.println("Warna     : " + warna);
        System.out.println("Volume    : " + getVolume());
    }
}
```

6. Buatlah sebuah *class* baru bernama **CangkirFactory**, isikan dengan code berikut:

```
public class CangkirFactory {
    public Cangkir create(String jenis, double diameter, double tinggi, String wrnOrMtf)
        jenis = jenis.toLowerCase();
        if (jenis.equals("melamin")) {
            return new CangkirMelamin(diameter, tinggi, wrnOrMtf);
        } else if (jenis.equals("kaca")) {
            return new CangkirKaca(diameter, tinggi, wrnOrMtf);
        } else if (jenis.equals("plastik")) {
            return new CangkirPlastik(diameter, tinggi, wrnOrMtf);
        }
        return null;
    }
}
```

7. Isikan code pada **main program** sebagai berikut:

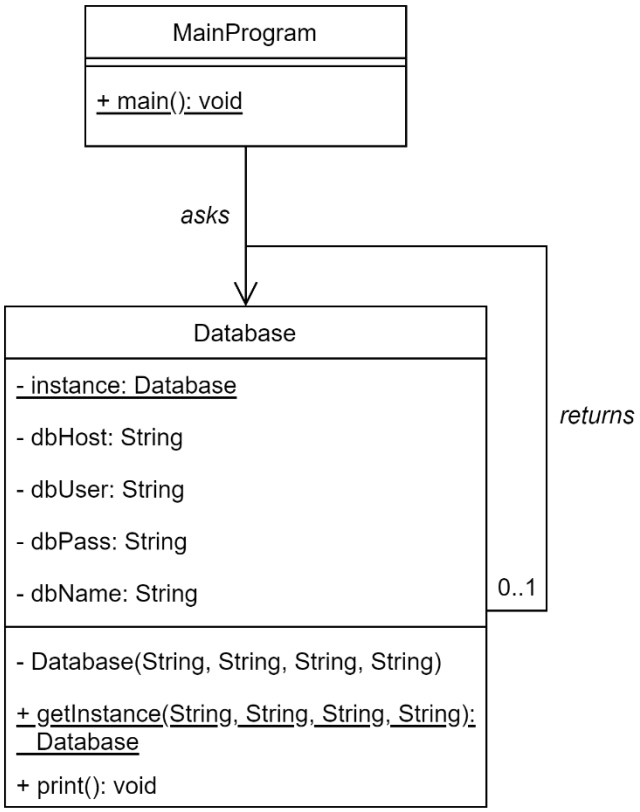
```
    public static void main(String[] args) {
        CangkirFactory cf = new CangkirFactory();
        Cangkir[] cangkir = new Cangkir[3];
        cangkir[0] = cf.create("melamin", 10, 20, "Merah");
        cangkir[1] = cf.create("kaca", 7, 20, "Bola");
        cangkir[2] = cf.create("plastik", 10, 14, "Hijau");
        for (Cangkir c : cangkir) {
            c.print();
        }
    }
```

8. Selesai. Coba jalankan, outputnya:

```
Cangkir Melamin
Diameter  : 10.0
Tinggi    : 20.0
Warna     : Merah
Volume    : 1570.7963267948967
Cangkir Kaca
Diameter  : 7.0
Tinggi    : 20.0
Motif     : Bola
Volume    : 769.6902001294993
Cangkir Plastik
Diameter  : 10.0
Tinggi    : 14.0
Warna     : Hijau
Volume    : 1099.5574287564277
```

GUIDED SINGLETON

Di guided ini kita akan membuatkan implemetasi dari diagram kelas yang sudah kita lihat di modul Singleton di atas:



Langkah-langkah:

1. Buka NetBeans dan buatkan project baru (Java with Ant → Java Application), namanya bebas, yang penting kalian ingat :)

   Kita akan membuatkan berbagai class dengan urutan:

   - Database *(singleton class)*
   - MainProgram (program utama)

2. Buatlah sebuah *class* baru bernama **Database**, isikan dengan code berikut:

```
1  public class Database {
2      private static Database instance = null;
3      private String dbHost;
4      private String dbUser;
5      private String dbPass;
6      private String dbName;
7
8      private Database(String dbHost, String dbUser, String dbPass, String dbName) {
9          this.dbHost = dbHost;
10         this.dbUser = dbUser;
11         this.dbPass = dbPass;
12         this.dbName = dbName;
13     }
14
```

```
14
15      public static Database getInstance(String dbH, String dbU, String dbP, String dbN) {
16          if (instance == null) {
17              instance = new Database(dbH, dbU, dbP, dbN);
18              System.out.println("Database instance created");
19              instance.print();
20          } else {
21              System.out.println("Database instance already created");
22              instance.print();
23          }
24          return instance;
25      }
26
27      public void print() {
28          System.out.println("DB Host: " + dbHost);
29          System.out.println("DB User: " + dbUser);
30          System.out.println("DB Pass: " + dbPass);
31          System.out.println("DB Name: " + dbName);
32      }
33  }
```

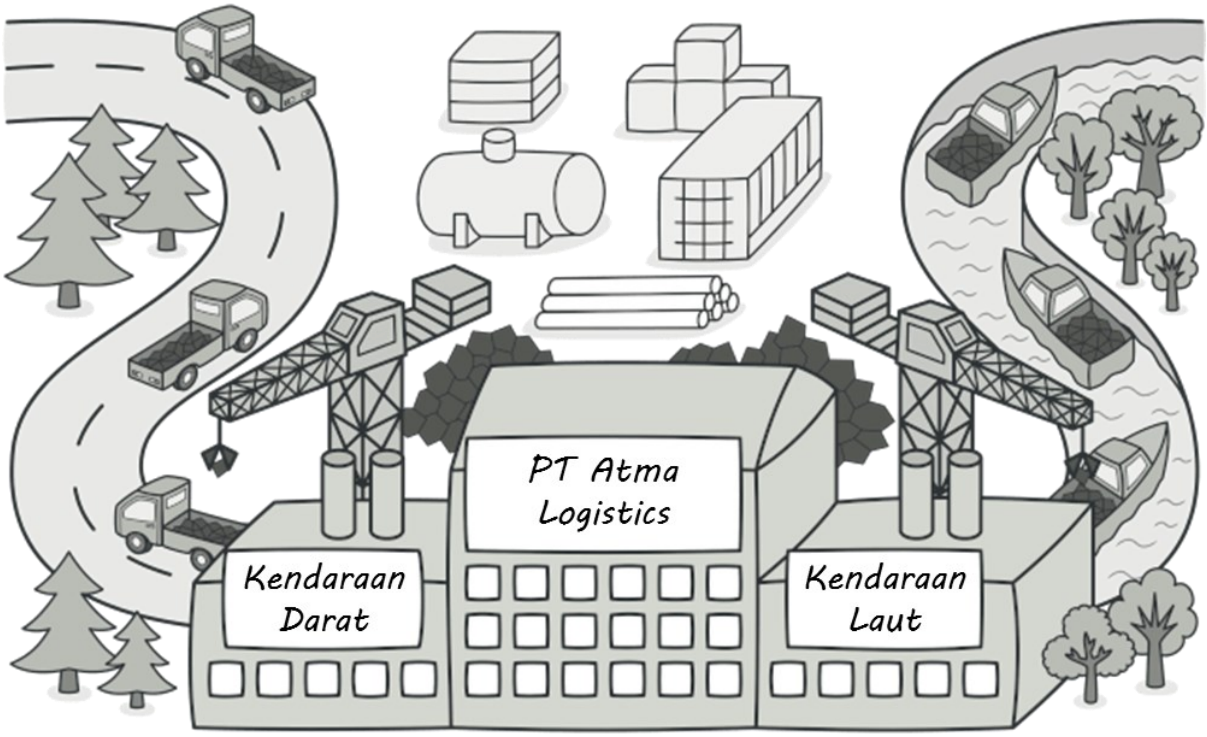
3. Isikan code pada **main program** sebagai berikut:

```
1       public static void main(String[] args) {
2           Database db;
3           System.out.println("Membuat database baru:");
4           db = Database.getInstance("localhost", "root", "", "kspj");
5
6           System.out.println("\nMembuat database baru:");
7           db = Database.getInstance("127.0.0.1", "kspj", "p@55w0rd", "ksp_javagui");
8       }
```

4. Selesai. Coba dijalankan, outputnya:

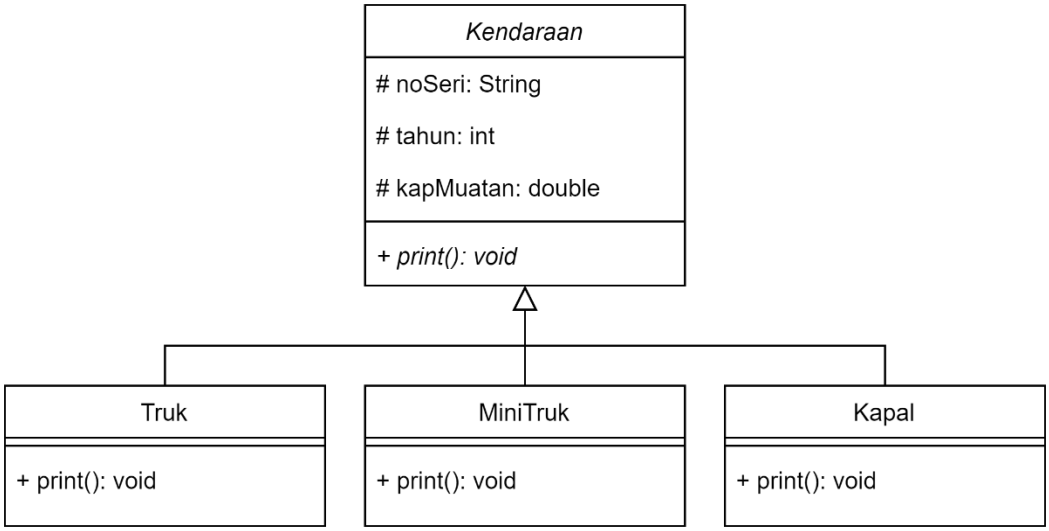
```
1  Membuat database baru:
2  Database instance created
3  DB Host: localhost
4  DB User: root
5  DB Pass:
6  DB Name: kspj
7
8  Membuat database baru:
9  Database instance already created
10 DB Host: localhost
11 DB User: root
12 DB Pass:
13 DB Name: kspj
```

LATIHAN MANDIRI FACTORY

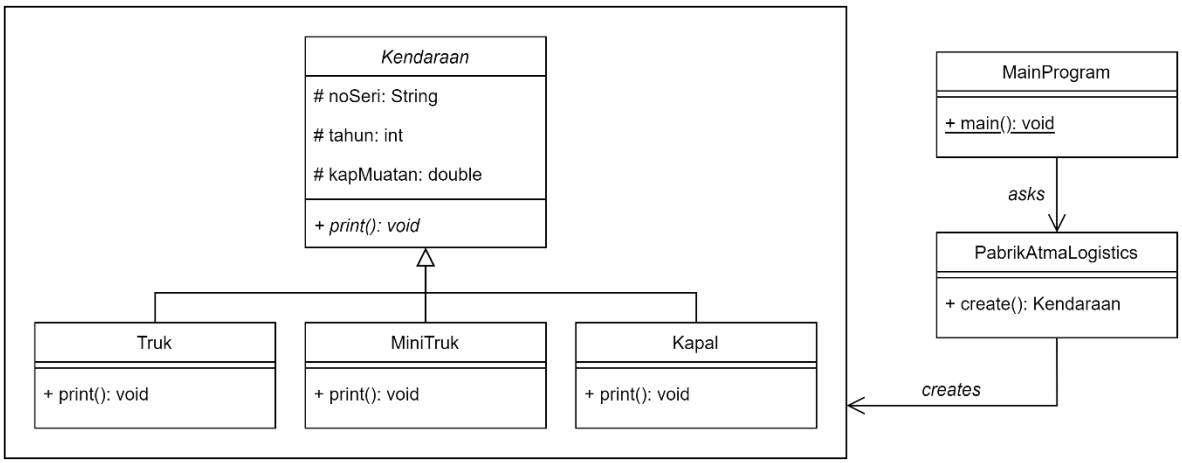


Adaptasi dari https://refactoring.guru/design-patterns/factory-method.

Mari kita lihat contoh pada gambar di atas. Terdapat sebuah **pabrik PT Atma Logistics** yang memiliki 2 divisi: divisi kendaraan darat dan divisi kendaraan laut. Pabrik ini dapat menciptakan beberapa jenis kendaraan, yakni **Truk**, **MiniTruk**, dan **Kapal**. Diagram kelas yang terbentuk seperti ini:



Kemudian, karena **diciptakan pada pabrik yang sama**, kita bisa mengimplementasikan design pattern **factory** seperti ini:



Buatlah **implementasi design pattern factory** berdasarkan diagram kelas di atas!

Contoh pseudocode di **MainProgram**:

```
class MainProgram do
    static function main do
        PabrikAtmaLogistics factory = new PabrikAtmaLogistics()
        Kendaraan[] kendaraan = new Kendaraan[6]
        kendaraan[0] = factory.create("truk", "Seri K1", 2023, 15.0)
        kendaraan[1] = factory.create("truk", "Seri J9", 2022, 20.0)

        kendaraan[2] = factory.create("minitruk", "MT-981", 2023, 5.0)
        kendaraan[3] = factory.create("minitruk", "MT-662", 2024, 10.0)

        kendaraan[4] = factory.create("kapal", "KP-12", 2025, 100.0)
        kendaraan[5] = factory.create("kapal", "KP-13", 2026, 200.0)

        for k in kendaraan do
            k.print()
        end for
    end function
end class
```

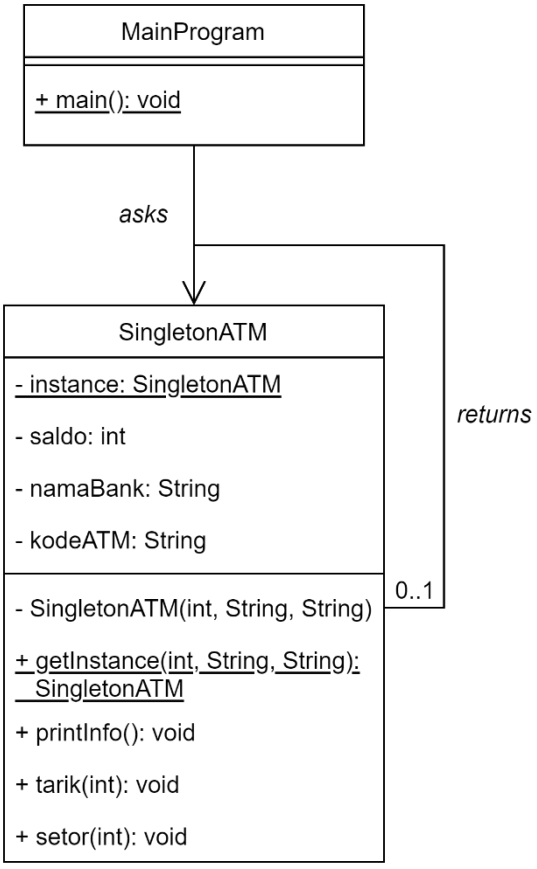
Contoh output:

```
Truk
No Seri      : Seri K1
Tahun        : 2023
Kap. Muatan  : 15.0

Truk
No Seri      : Seri J9
Tahun        : 2022
Kap. Muatan  : 20.0

MiniTruk
No Seri      : MT-981
Tahun        : 2023
Kap. Muatan  : 5.0

MiniTruk
No Seri      : MT-662
Tahun        : 2024
Kap. Muatan  : 10.0

Kapal
No Seri      : KP-12
Tahun        : 2025
Kap. Muatan  : 100.0

Kapal
No Seri      : KP-13
Tahun        : 2026
Kap. Muatan  : 200.0
```

LATIHAN MANDIRI SINGLETON

Ilustrasi: Dreamstime (linknya terlalu panjang)

Contoh di atas merupakan salah satu contoh kelas singleton dalam dunia nyata. Ada **satu ATM** yang dapat digunakan berulang kali oleh banyak orang. Unit ATM ini memiliki atribut (variabel) **saldo, nama bank,** dan **kode ATM**. Di ATM ini dapat dilakukan **tarik uang** dan **setor uang**. Dengan demikian, diagram kelas yang terbentuk seperti ini:



Buatlah **implementasi design pattern singleton** berdasarkan diagram kelas di atas!

Contoh pseudocode di **MainProgram**:

```
class MainProgram do
    static function main do
        SingletonATM atm;
        print "Mengubah data ATM menjadi Bank BRI, Kode ATM001, Saldo Rp 10.000.000"
        atm = SingletonATM.getInstance(10000000, "Bank BRI", "ATM001")
        atm.printInfo()

        print "\nPenarikan Rp 500.000:"
        atm.tarik(500000)
        atm.printInfo()

        print "\nPenarikan Rp 10.000.000:"
        atm.tarik(10000000)
        atm.printInfo()

        print "\nMengubah data ATM menjadi Bank BCA, Kode BCA001, Saldo Rp 50.000.000"
        atm = SingletonATM.getInstance(50000000, "Bank BCA", "BCA001")
        atm.printInfo()

        print "\nSetor Rp 14.000.000:"
        atm.setor(14000000)
        atm.printInfo()
    end
end
```

Contoh output:

```
Mengubah data ATM menjadi Bank BRI, Kode ATM001, Saldo Rp 10.000.000
ATM baru dibuat.
Nama Bank  : Bank BRI
Kode ATM   : ATM001
Saldo ATM  : 10000000

Penarikan Rp 500.000:
Nama Bank  : Bank BRI
Kode ATM   : ATM001
Saldo ATM  : 9500000

Penarikan Rp 10.000.000:
Saldo di ATM tidak mencukupi! Silakan hubungi bank Anda.
Nama Bank  : Bank BRI
Kode ATM   : ATM001
Saldo ATM  : 9500000

Mengubah data ATM menjadi Bank BCA, Kode BCA001, Saldo Rp 50.000.000
ATM sudah ada. Tidak bisa membuat ATM baru.
Nama Bank  : Bank BRI
Kode ATM   : ATM001
Saldo ATM  : 9500000

Setor Rp 14.000.000:
Nama Bank  : Bank BRI
Kode ATM   : ATM001
Saldo ATM  : 23500000
```