



POLITEKNIK ELEKTRONIKA NEGERI SURABAYA



DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

# CLASS DIAGRAM

PBO

*Yunia Ikawati*

Teknik Informatika-PENS

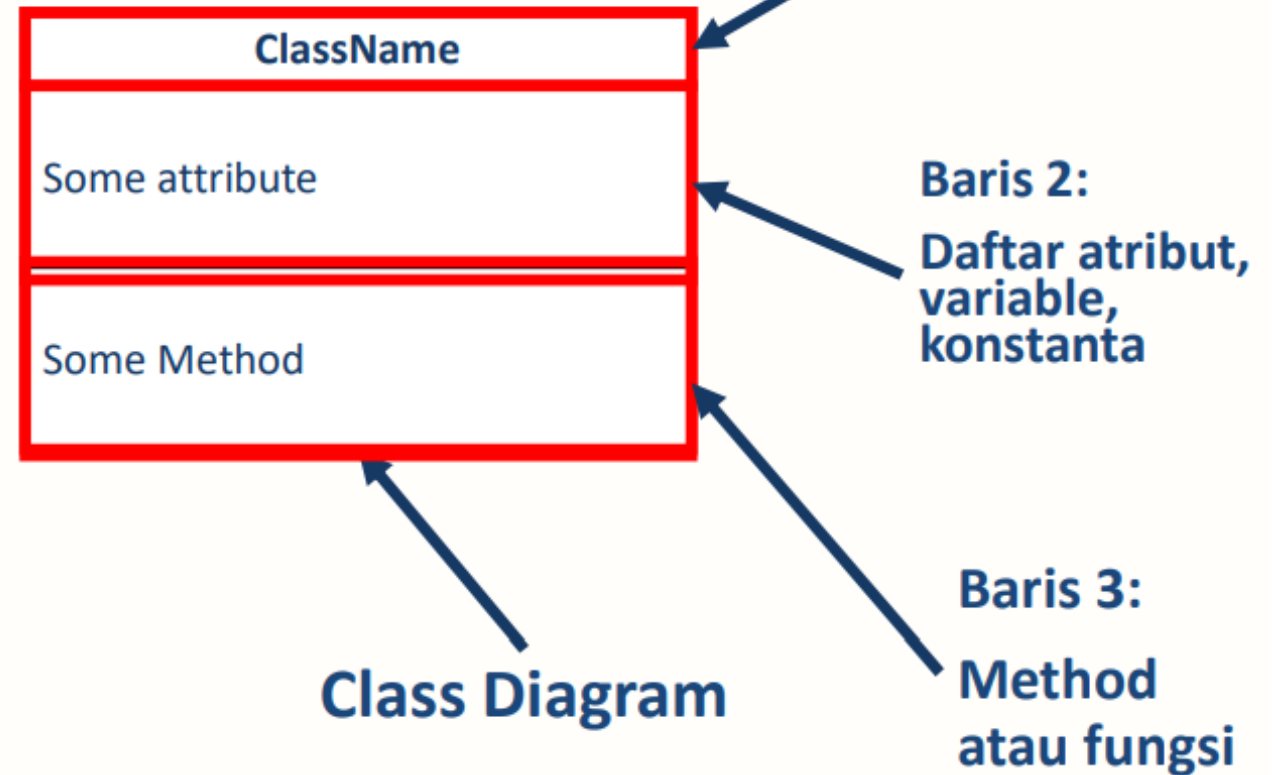


# OOP

tidak mungkin terlepas dari  
**Class Diagram**

## Apa itu **Class Diagram**?

Diagram yang  
menggambarkan **Class**

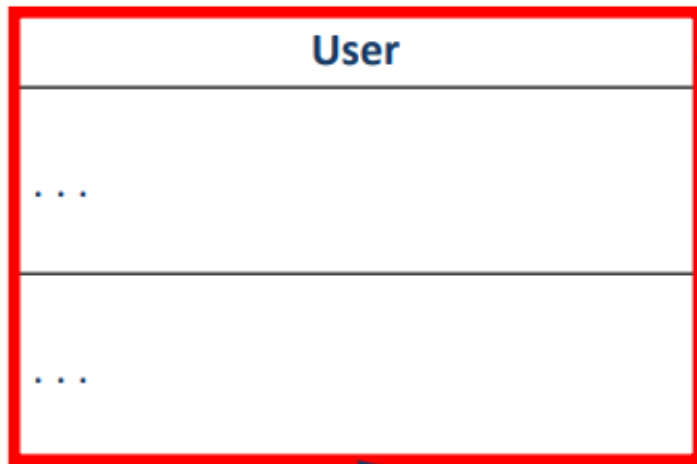




DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

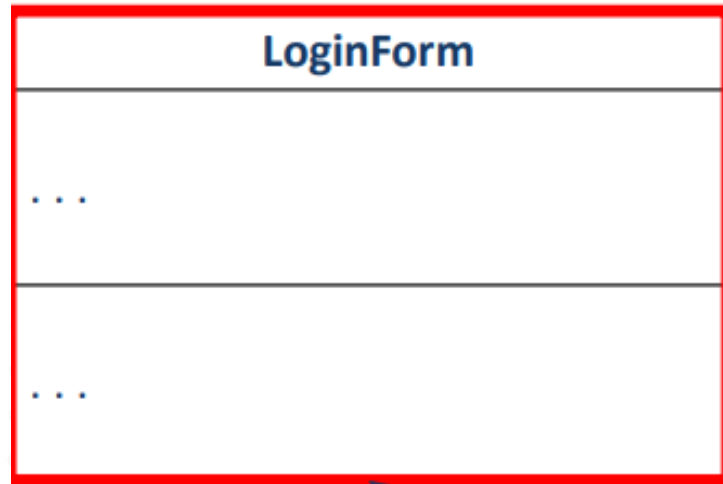
# CLASS NAME



Jika di implementasikan  
dalam Bahasa java:

```
public class User{  
    ....  
}
```

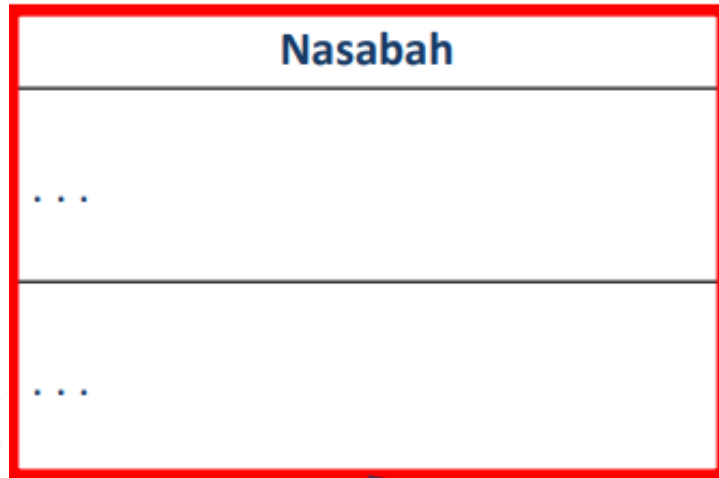
Ini adalah satu buah class  
dengan nama class : **User**



Jika di implementasikan dalam Bahasa java:

```
public class LoginForm{  
    ....  
}
```

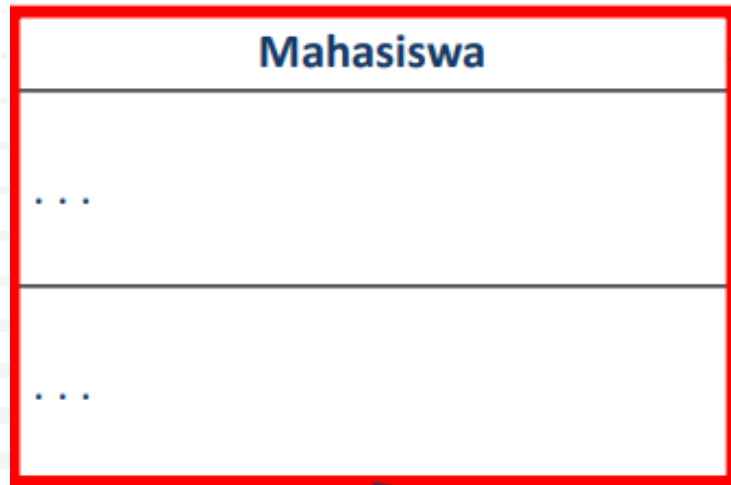
Ini adalah satu buah class dengan nama class : **LoginForm**



Jika di implementasikan dalam Bahasa java:

```
public class Nasabah{  
    ....  
}
```

Ini adalah satu buah class dengan nama class : **Nasabah**



Ini adalah satu buah class  
dengan nama class : **Mahasiswa**

Jika di implementasikan  
dalam Bahasa java:

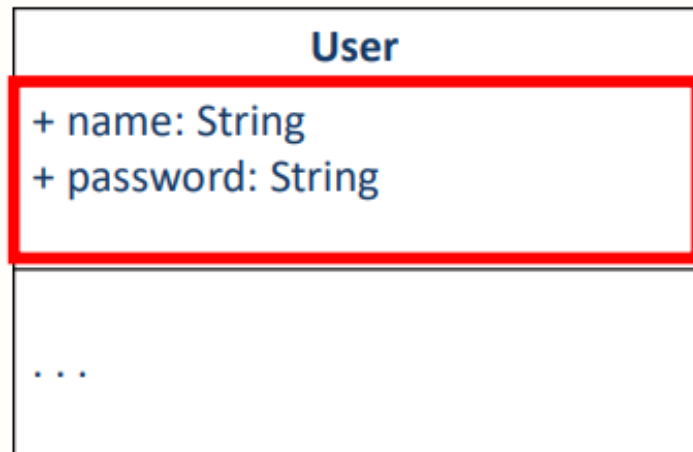
... ?



DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

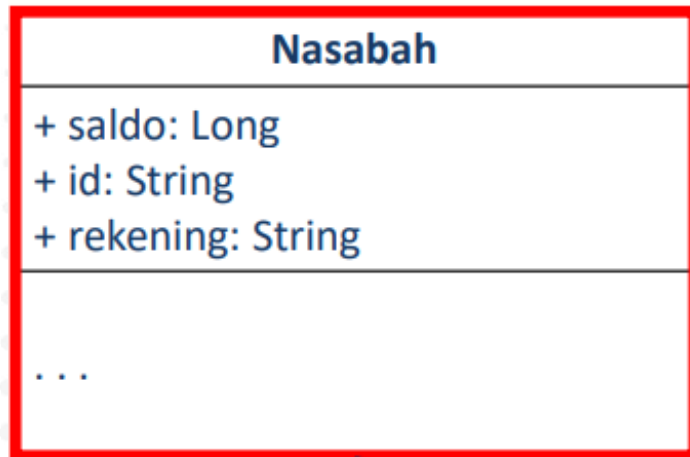
# ATTRIBUTE



Dalam class User terdapat  
dua variable : name dan  
password

Jika di implementasikan  
dalam Bahasa java:

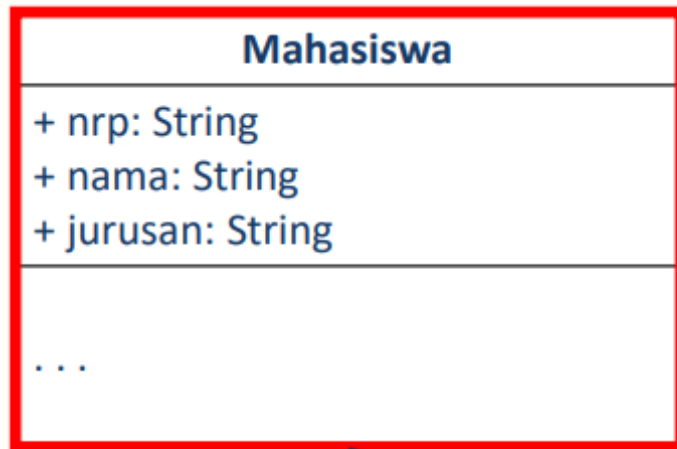
```
public class User{  
    public String name;  
    public String password;  
}
```



Dalam class Nasabah terdapat 3 buah variable: saldo, id, rekening

Jika di implementasikan dalam Bahasa java:

```
public class Nasabah{  
    public Long saldo;  
    public String id;  
    public String rekening;  
}
```



Dalam class Mahasiswa ada 3 variable: nrp, nama, jurusan

Jika di implementasikan dalam Bahasa java:

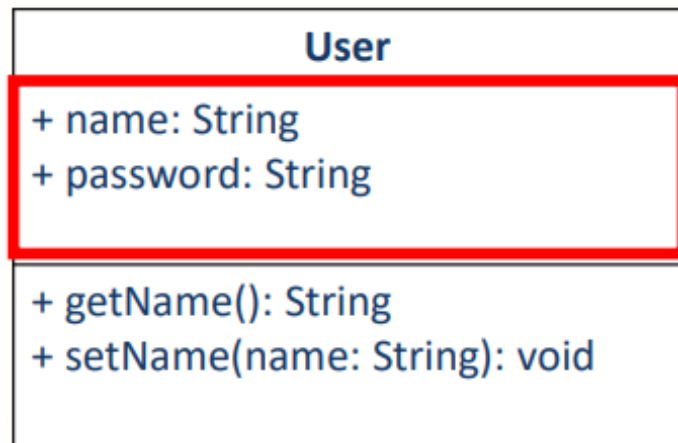
... ?



DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

# METHOD

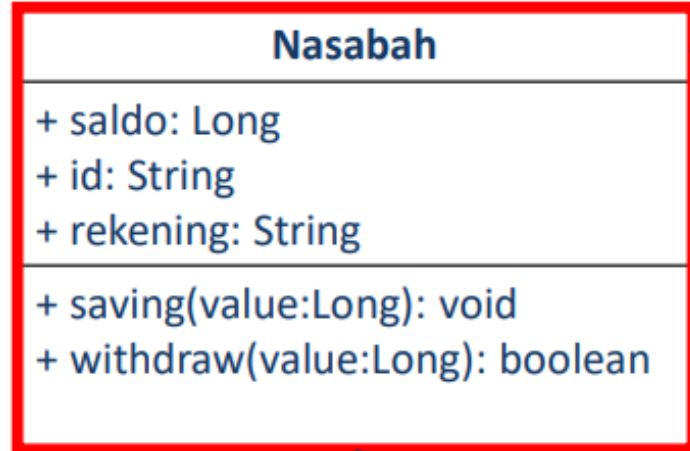


Dalam class User terdapat dua method : `getName()` dan `setName()`

Jika di implementasikan dalam Bahasa java:

```
public class User{
    public String name;
    public String password;

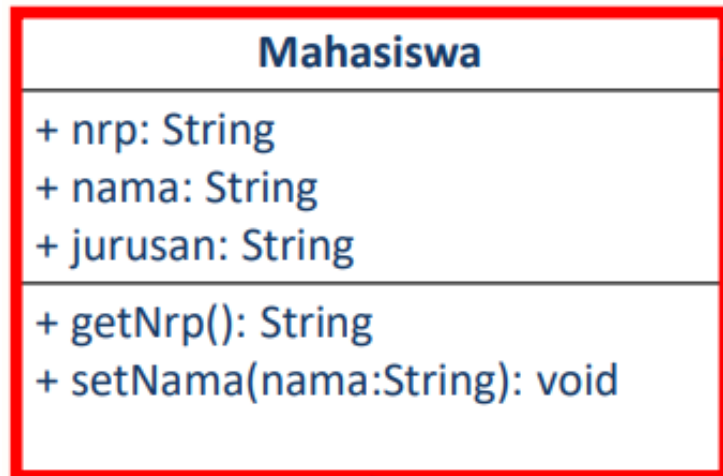
    public String getName(){...}
    public void setName(String
name){...}
}
```



Jika di implementasikan dalam Bahasa java:

```
public class Nasabah{  
    public Long saldo;  
    public String id;  
    public String rekening;  
  
    public void saving(Long value){...}  
    public Boolean withdraw(Long value){...}  
}
```

Dalam class Nasabah terdapat 2 buah method: saving() dan withdraw()



Jika di implementasikan  
dalam Bahasa java:

... ?

Dalam class Mahasiswa ada 2  
method: getNrp() dan setName()



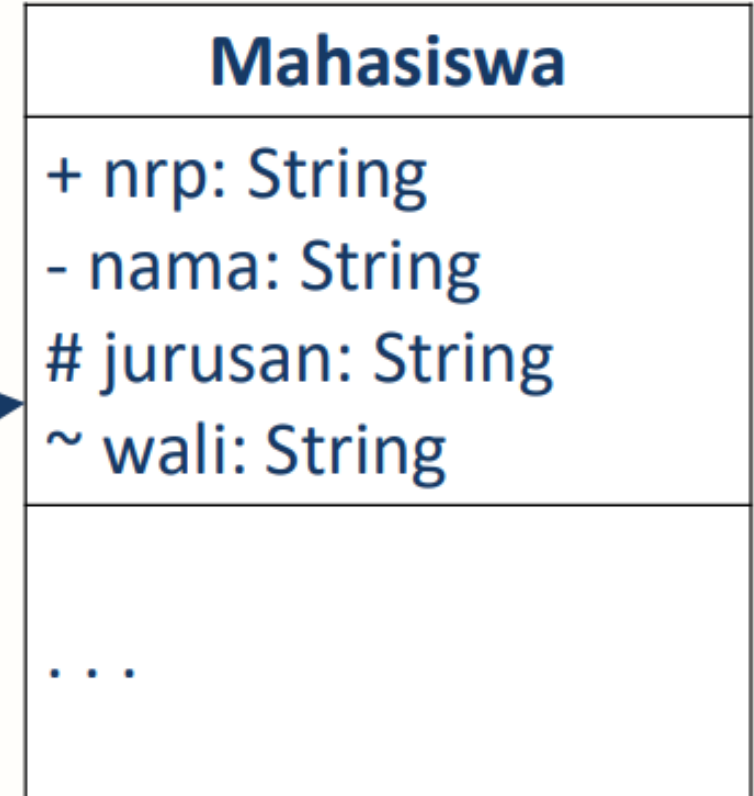
DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

# MODIFIER

| Modifier  | Kelas Sendiri | Subclass | Paket Sama | Luar Paket |
|-----------|---------------|----------|------------|------------|
| public    | ✓             | ✓        | ✓          | ✓          |
| protected | ✓             | ✓        | ✓          | ✗          |
| private   | ✓             | ✗        | ✗          | ✗          |
| default   | ✓             | ✗        | ✓          | ✗          |

+ adalah **public**  
- adalah **private**  
# adalah **protected**  
~ adalah **default**



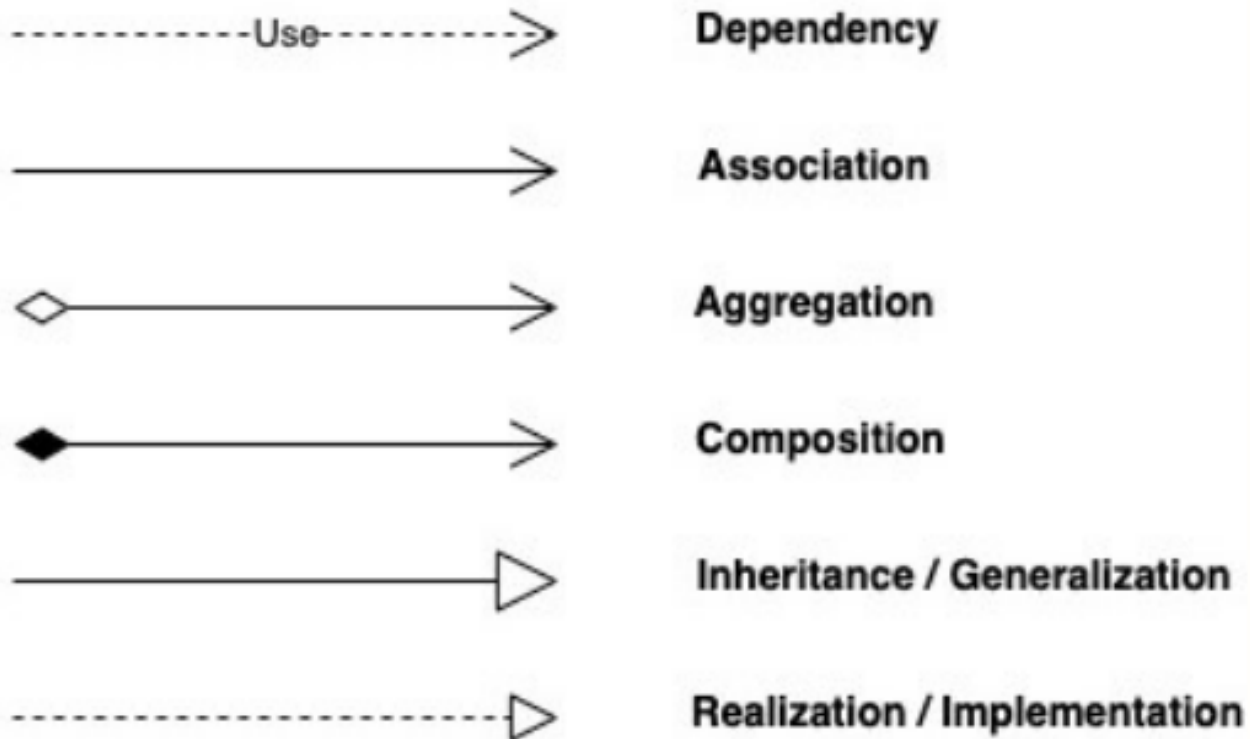


DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

# CLASS DIAGRAM RELATIONSHIP

# CLASS DIAGRAM RELATIONSHIP



- **Dependency** → penggunaan sementara.
- **Association** → hubungan umum.
- **Aggregation** → bagian yang bisa berdiri sendiri.
- **Composition** → bagian yang bergantung penuh.
- **Inheritance** → pewarisan antar kelas.
- **Realization** → implementasi dari interface.

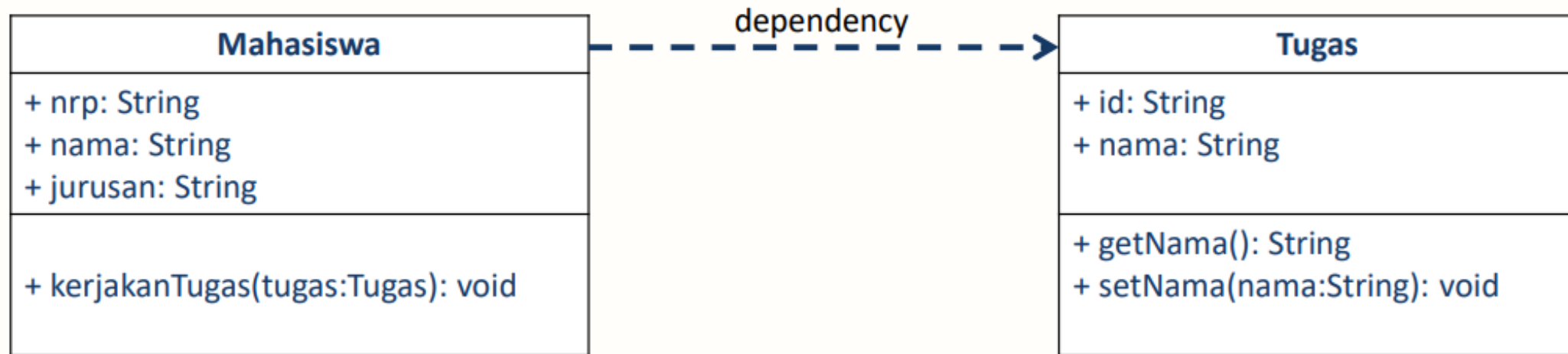


DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

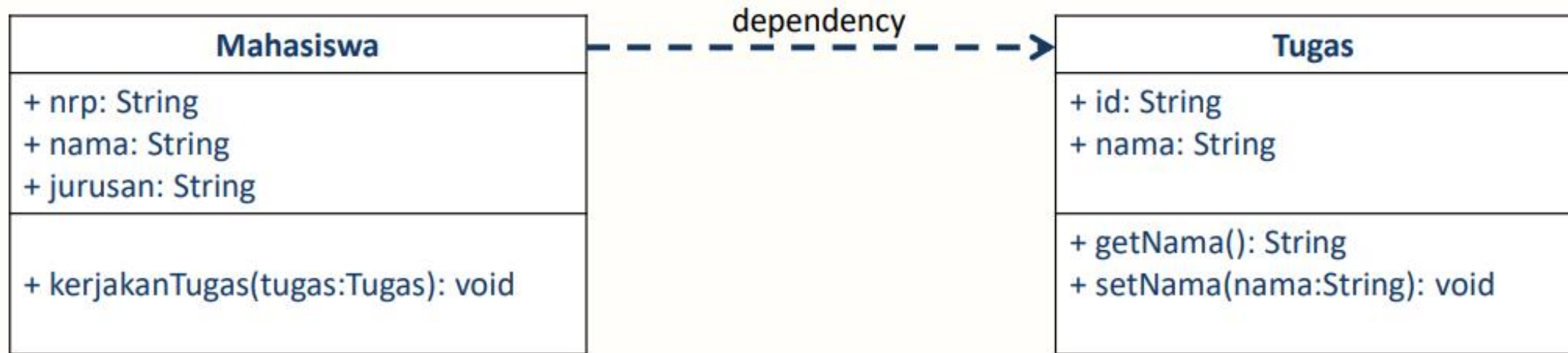
# CLASS DIAGRAM RELATIONSHIP

## Dependency



```
public class Mahasiswa{  
    public String nrp;  
    public String nama;  
    public String jurusan;  
  
    public void kerjakanTugas(Tugas tugas){  
        ...  
    }  
}
```

```
public class Tugas{  
    public String id;  
    public String nama;  
  
    public String getName(){ ... }  
    public void setName(String nama){ ... }  
}
```



```
public class Mahasiswa{
    public String nrp;
    public String nama;
    public String jurusan;

    public void kerjakanTugas(Tugas tugas){
        ...
    }
}
```

- Method `kerjakanTugas` memiliki sebuah parameter input berupa object bertipe `Tugas`.
- Object ini hanya menjadi **variable local** dalam method `kerjakanTugas`.
- **Tidak ada creational / instansiasi object** tugas baru  
→ "`new Tugas()`" dalam kelas `Mahasiswa`.
- Sehingga relasinya hanya berupa dependency.

Kelas `Mahasiswa` **bergantung sementara** pada kelas `Tugas`, hanya ketika menjalankan method `kerjakanTugas()`.

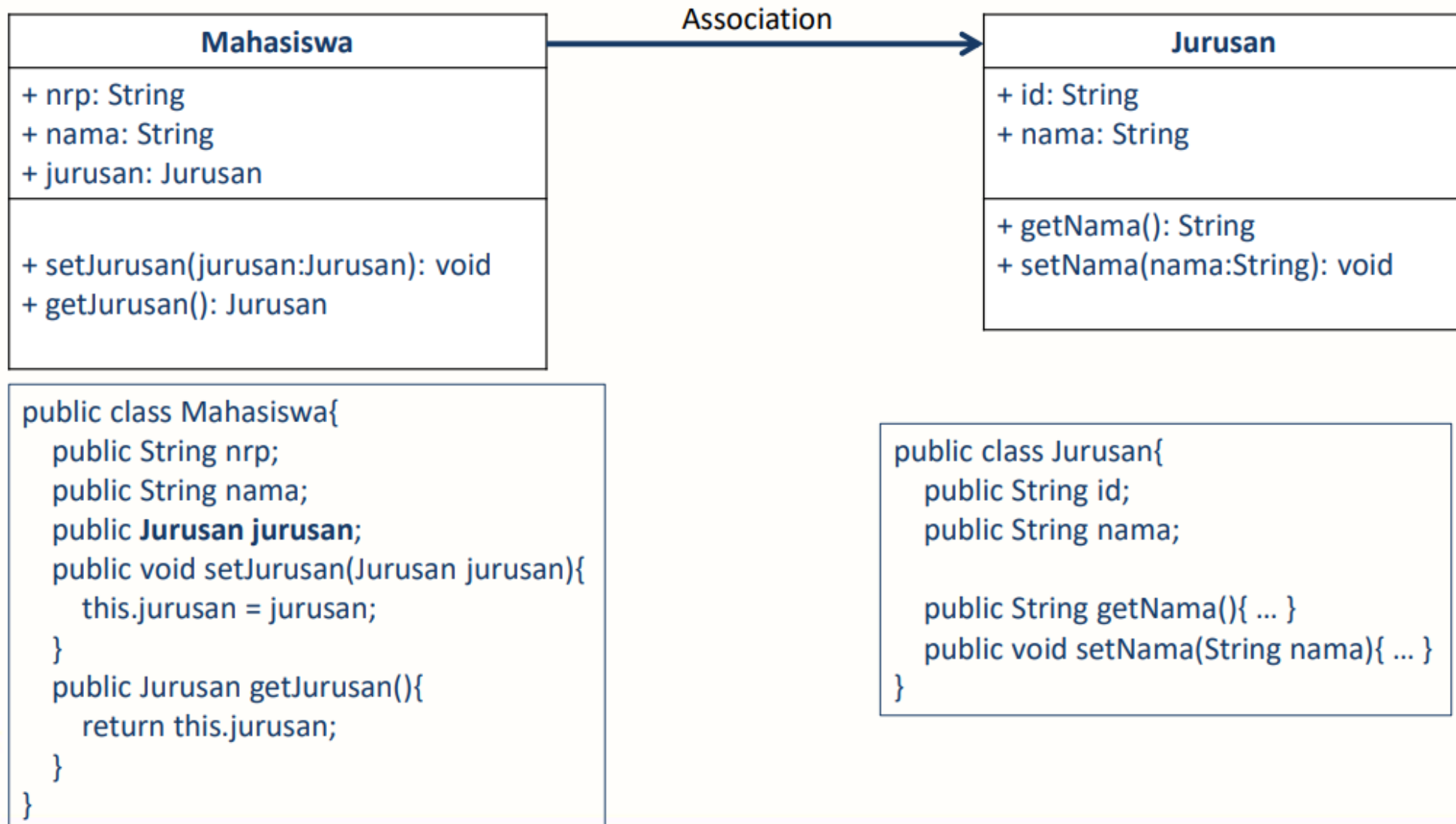


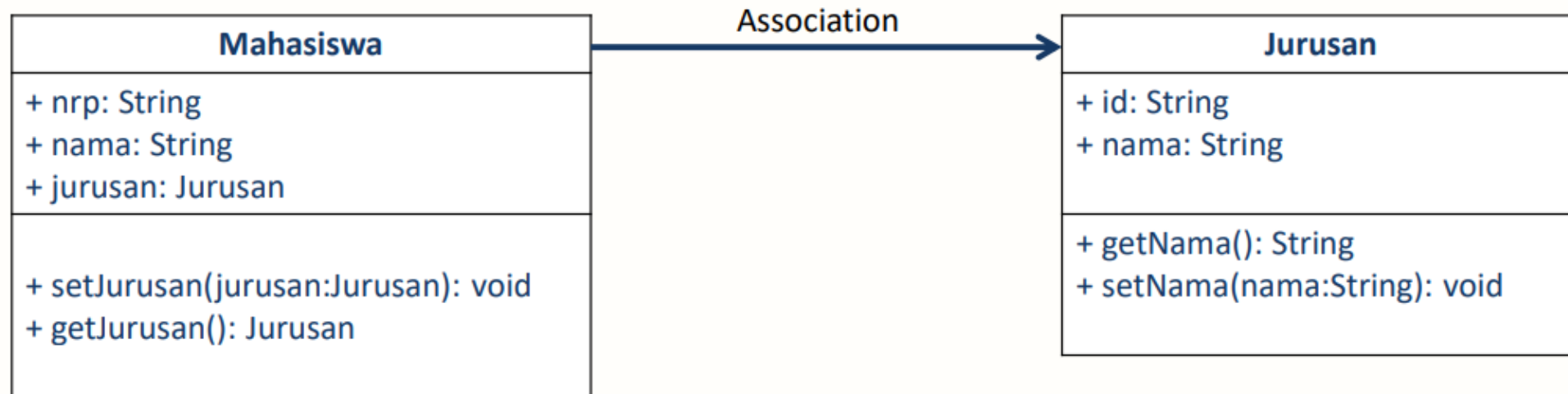
DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

# CLASS DIAGRAM RELATIONSHIP

## Association





```
public class Mahasiswa{
    public String nrp;
    public String nama;
    public Jurusan jurusan;
    public void setJurusan(Jurusan jurusan){
        this.jurusan = jurusan;
    }
    public Jurusan getJurusan(){
        return this.jurusan;
    }
}
```

- Jurusan menjadi sebuah object dalam kelas Mahasiswa
- Jurusan menjadi **variable global** dalam kelas Mahasiswa
- Namun kelas mahasiswa **tidak melakukan instansiasi** object Jurusan  
tidak melakukan → new Jurusan();
- Yang melakukan instansiasi object bertipe Jurusan adalah kelas lain.
- Kelas mahasiswa menerima object jurusan dari kelas lain dan meng-assign object tersebut dalam object jurusan miliknya.

Kelas Mahasiswa bisa ada tanpa Jurusan dan Kelas Jurusan juga bisa ada tanpa Mahasiswa.

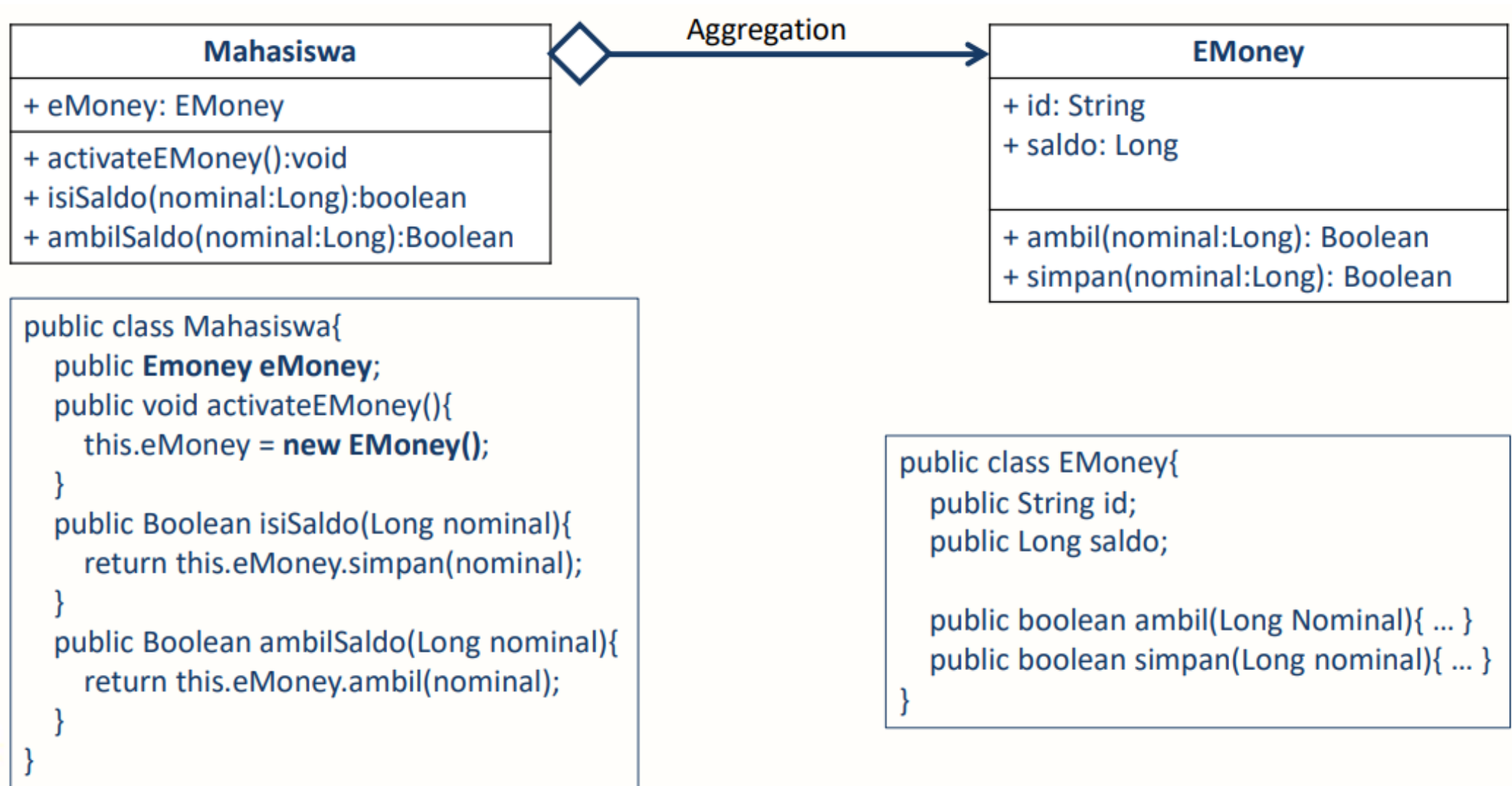


DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

# CLASS DIAGRAM RELATIONSHIP

## Aggregation





```
public class Mahasiswa{
    public EMoney eMoney;
    public void activateEMoney(){
        this.eMoney = new EMoney();
    }
    public Boolean isiSaldo(Long nominal){
        return this.eMoney.simpan(nominal);
    }
    public Boolean ambilSaldo(Long nominal){
        return this.eMoney.ambil(nominal);
    }
}
```

#### Aggregation versi 1

- EMoney menjadi sebuah object dalam kelas Mahasiswa
- Emoney menjadi **variable global** dalam kelas Mahasiswa
- Kelas mahasiswa **melakukan instansiasi** object EMoney  
instansiasi → new EMoney();

Kelas Mahasiswa *memiliki* EMoney, tetapi **EMoney dapat berdiri sendiri** (tidak sepenuhnya bergantung pada Mahasiswa).

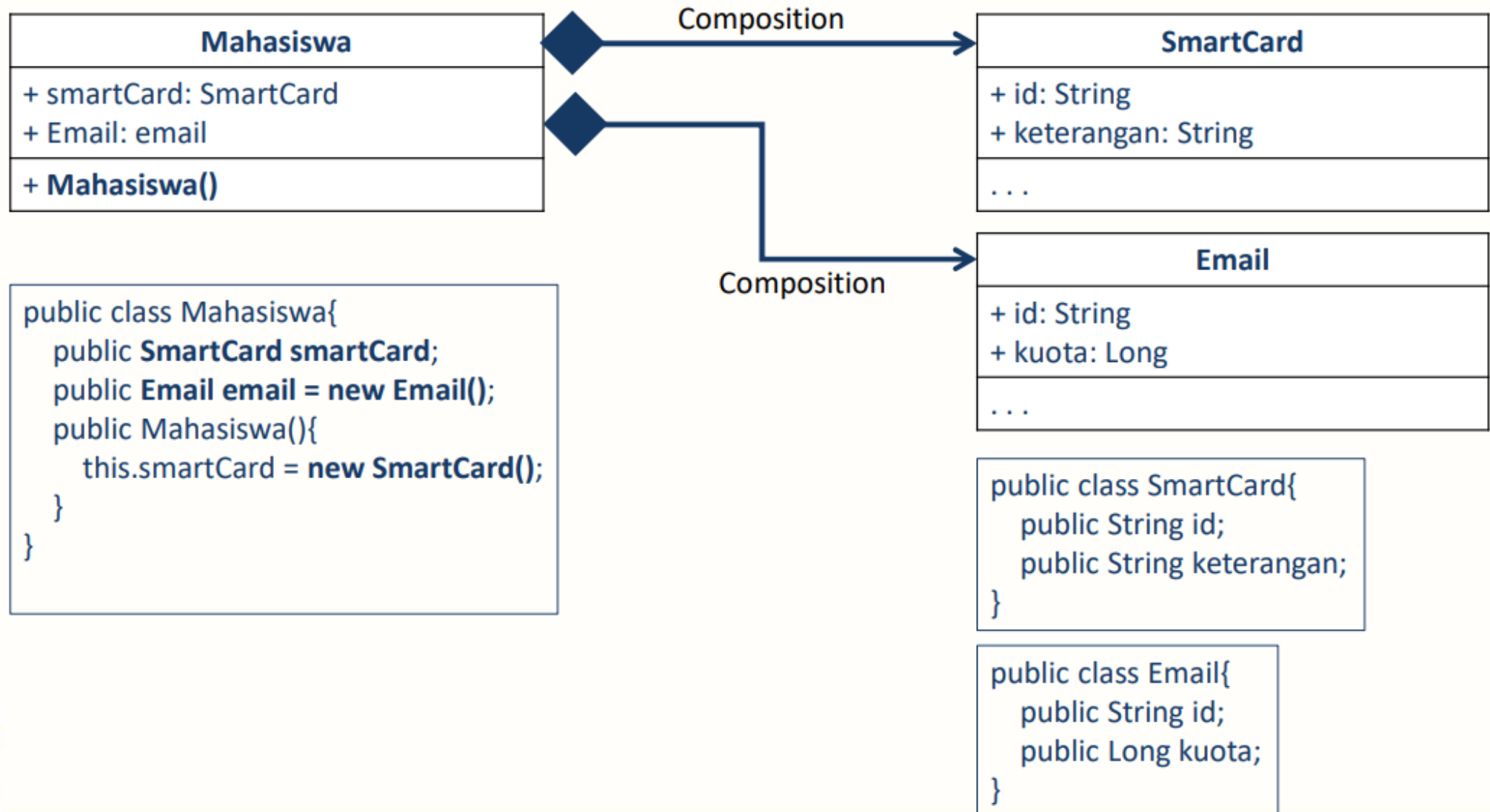


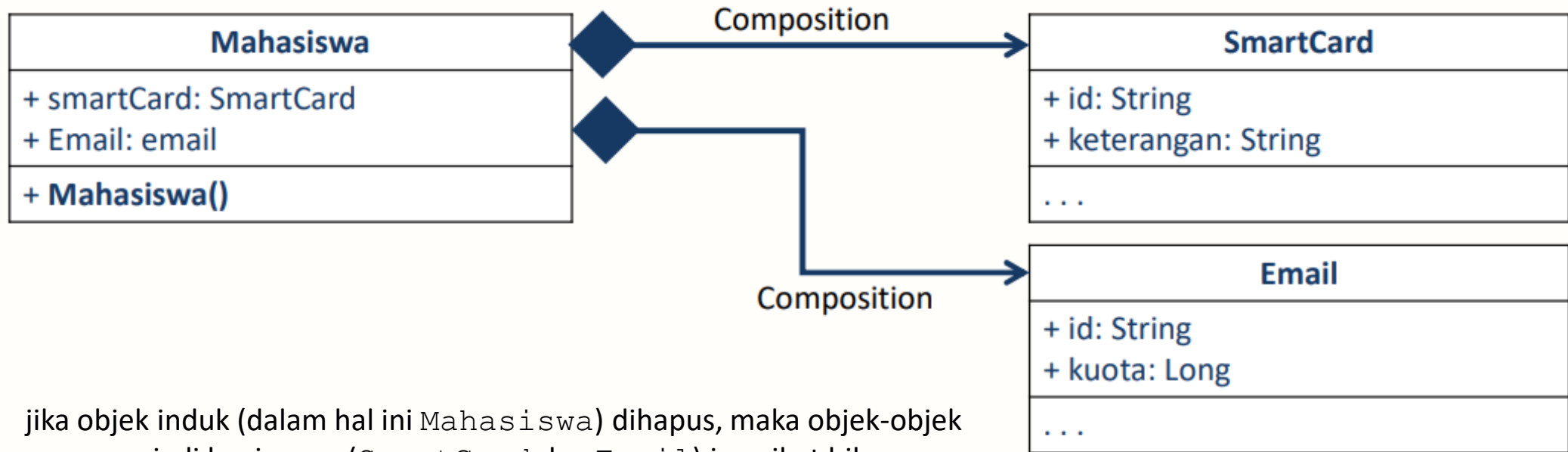
DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

# CLASS DIAGRAM RELATIONSHIP

## Composition





jika objek induk (dalam hal ini Mahasiswa) dihapus, maka objek-objek yang menjadi bagiannya (SmartCard dan Email) juga ikut hilang.

```
public class Mahasiswa{
    public SmartCard smartCard;
    public Email email = new Email();
    public Mahasiswa(){
        this.smartCard = new SmartCard();
    }
}
```

- SmartCard dan Email menjadi **variable global** dalam kelas Mahasiswa
- Terjadi instansiasi SmartCard dan Email dalam kelas Mahasiswa  
instansiasi → `new SmartCard()`  
→ `new Email()`
- Instansiasi dilakukan didalam konstruktor ataupun bersamaan dengan deklarasi object.
- Ketika Kelas **mahasiswa** menjadi **object aktif** / berjalan. Maka object smartcard dan email **sudah harus di instansiasi**
- Mahasiswa belum lengkap jika belum memiliki smartcard dan email



DIKTISAINTEK  
BERDAMPAK

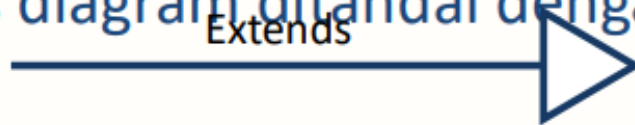
KAMPUS  
INOVASI

# CLASS DIAGRAM RELATIONSHIP

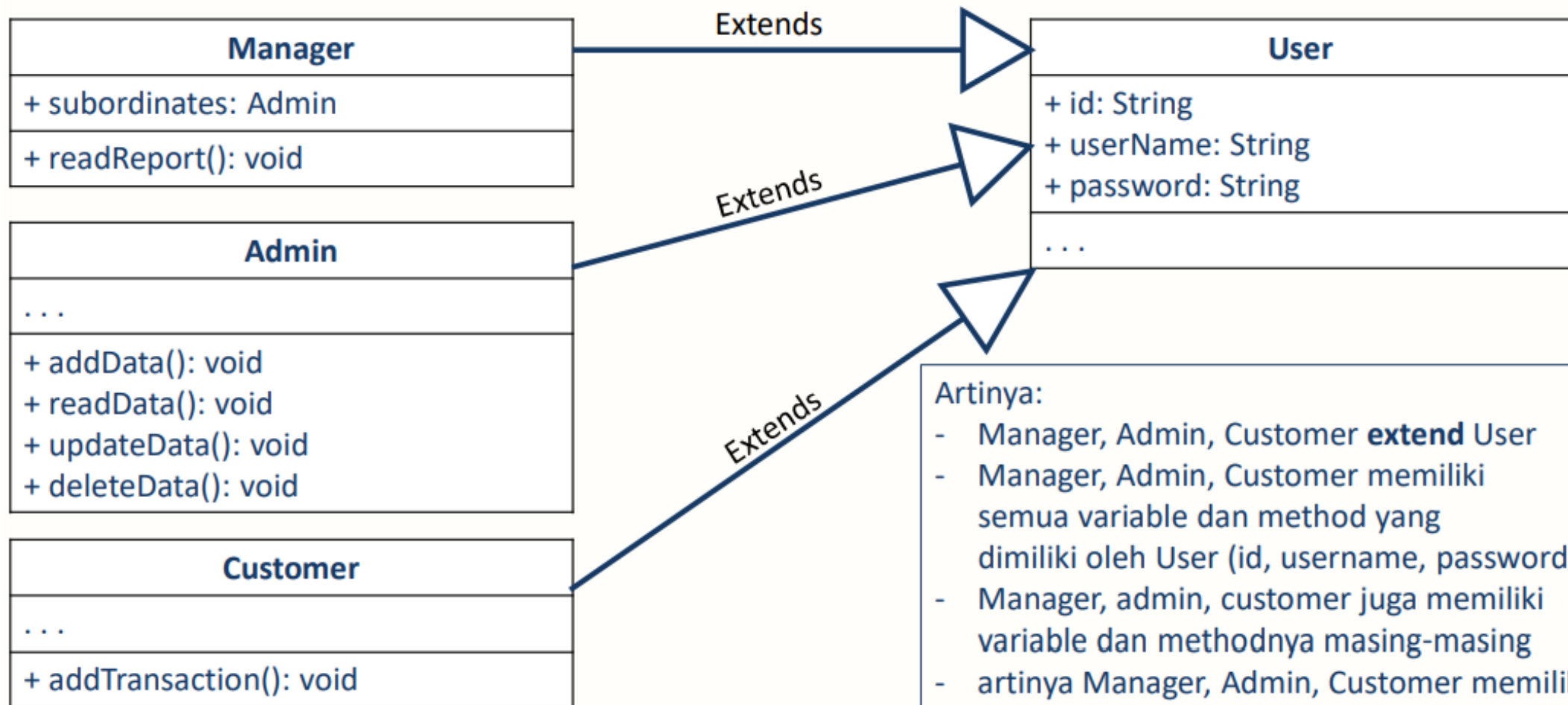
## Inheritance / Extends

# Inheritance

- **Inheritance** adalah pewarisan antar class parent terhadap child
- Disebut juga **Extends**
- Yang diwariskan adalah **semua variable dan method**
- Yang **tidak diwariskan** adalah **constructor**, variable dan method dengan modifier **private** atau **default(unspecified)**
- Dalam class diagram ditandai dengan garis relationship sebagai berikut:



- Jika class Child inherit terhadap class Parent maka child memiliki semua variable dan method yang dimiliki parent kecuali yang private dan default(unspecified) serta constructor.



Artinya:

- Manager, Admin, Customer **extend** User
- Manager, Admin, Customer memiliki semua variable dan method yang dimiliki oleh User (id, username, password)
- Manager, admin, customer juga memiliki variable dan methodnya masing-masing
- artinya Manager, Admin, Customer memiliki variable+method milik User sekaligus memiliki variabelnya sendiri.
- Manager, Admin, Customer lebih luas dari User
- Maka dari itu disebut extends/perluasan

Jika diimplementasikan menjadi kode program:

```
public class User{  
    public String id;  
    public String userName;  
    public String password;  
}
```

```
public class Manager extends User{  
    public Admin subordinates;  
    public void readReport(){  
        ...  
    }  
}
```

```
public class Admin extends User{  
    public void addData(){...}  
    public void readData(){...}  
    public void updateData(){...}  
    public void deleteData(){...}  
}
```

```
public class Customer extends User{  
    public void addTransaction(){...}  
}
```



# CLASS DIAGRAM RELATIONSHIP

## Realization

# Realization

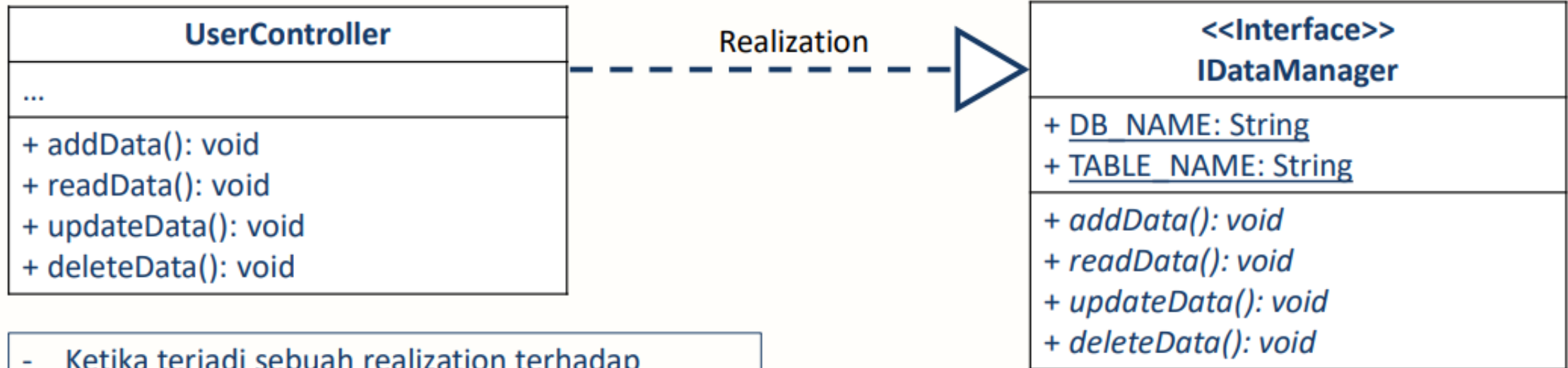
- Realization terjadi antara **Class** dengan **Interface**.
- Sama dengan class, **Interface dapat memiliki variables dan methods**.
- Namun Interface hanya bisa memiliki **method Abstract**.
- Method abstract adalah method kosong tanpa method body. Method ini hanya deklarasi saja. Contoh:

```
public int getData();  
public void addData();  
public Boolean edit(int data);
```
- Variable yang dideklarasikan dalam Interface harus **public, static & final**

# Realization

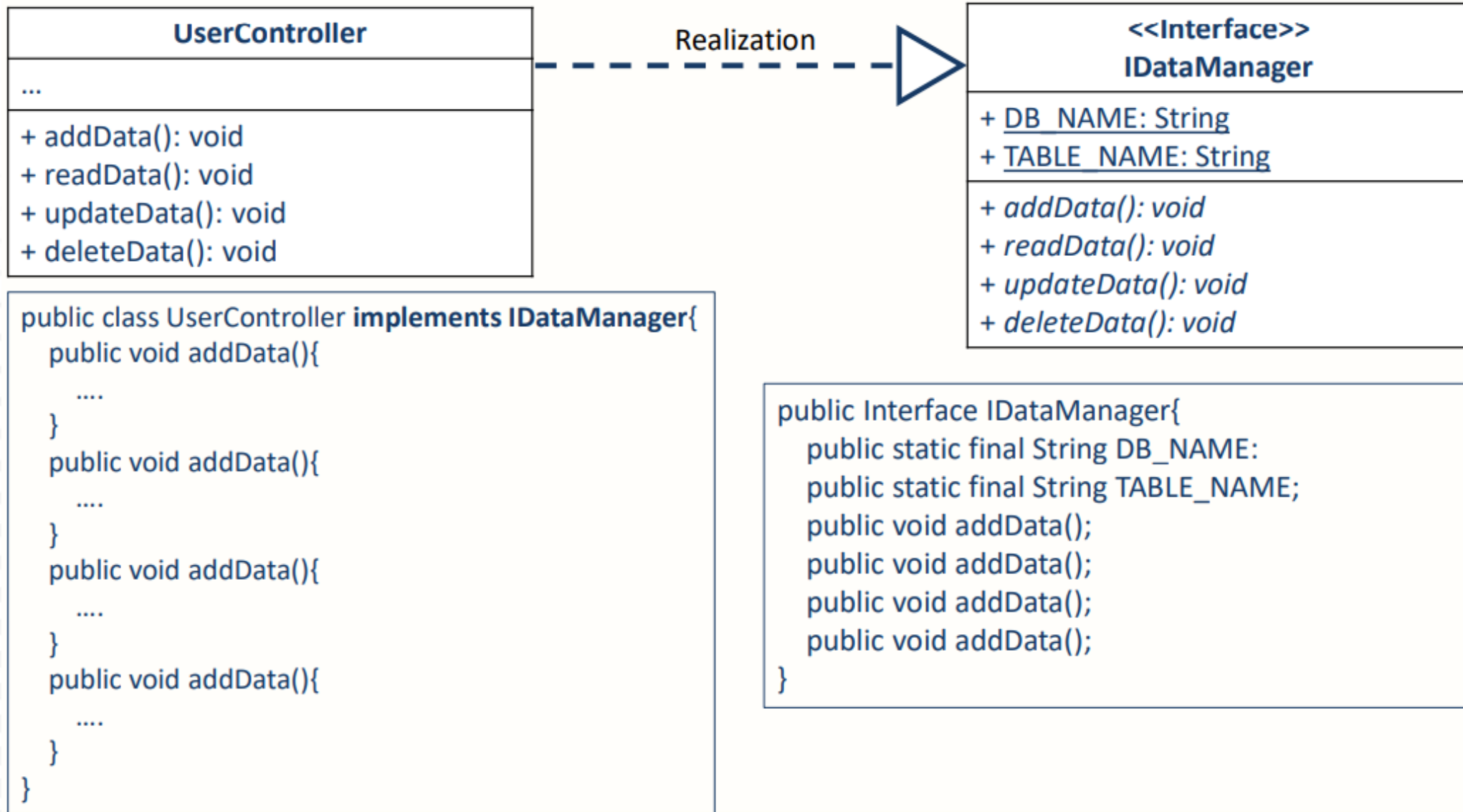
- Menggambarkan Interface dalam class diagram sedikit berbeda dengan class yaitu dengan menambahkan kata interface diatas nama interface
- variable static dituliskan dengan tanda underline/garis bawah
- Variable final dituliskan dengan huruf Capital semua. Dengan pemisah antar kata menggunakan underscore
- Method abstract ditandai dengan tulisan italic/miring

| <b>&lt;&lt;Interface&gt;&gt;<br/>IDataManager</b>                                                                   |
|---------------------------------------------------------------------------------------------------------------------|
| + <u>DB_NAME</u> : String<br>+ <u>TABLE_NAME</u> : String                                                           |
| + <i>addData(): void</i><br>+ <i>readData(): void</i><br>+ <i>updateData(): void</i><br>+ <i>deleteData(): void</i> |



- Ketika terjadi sebuah realization terhadap sebuah interface, maka class yang merealisasi wajib mengimplementasikan semua method abstract yang dimiliki oleh Interface.
- UserController wajib merealisasikan method `addData()`, `readData()`, `updateData()`, `deleteData()`
- Method yang ditulis miring/italic menunjukkan bahwa itu adalah method abstract. Sedangkan method yang ditulis tegak/normal menunjukkan itu method concrete/nyata.

- Jika UserController tidak mengimplementasikan semua method abstract milik Interface **IDataManager**, maka UserController juga harus menjadi Interface atau Abstract Class.



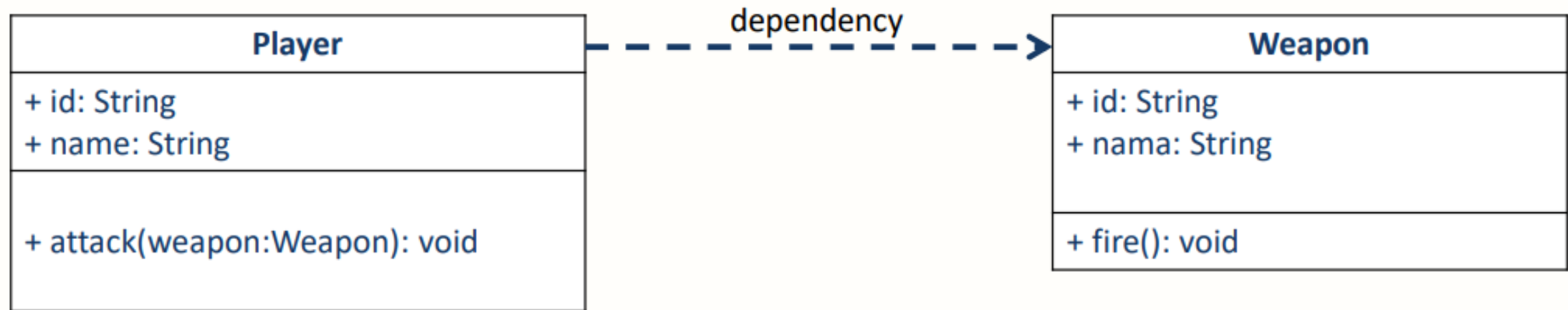


DIKTISAINTEK  
BERDAMPAK

KAMPUS  
INOVASI

# CLASS DIAGRAM

## LATIHAN

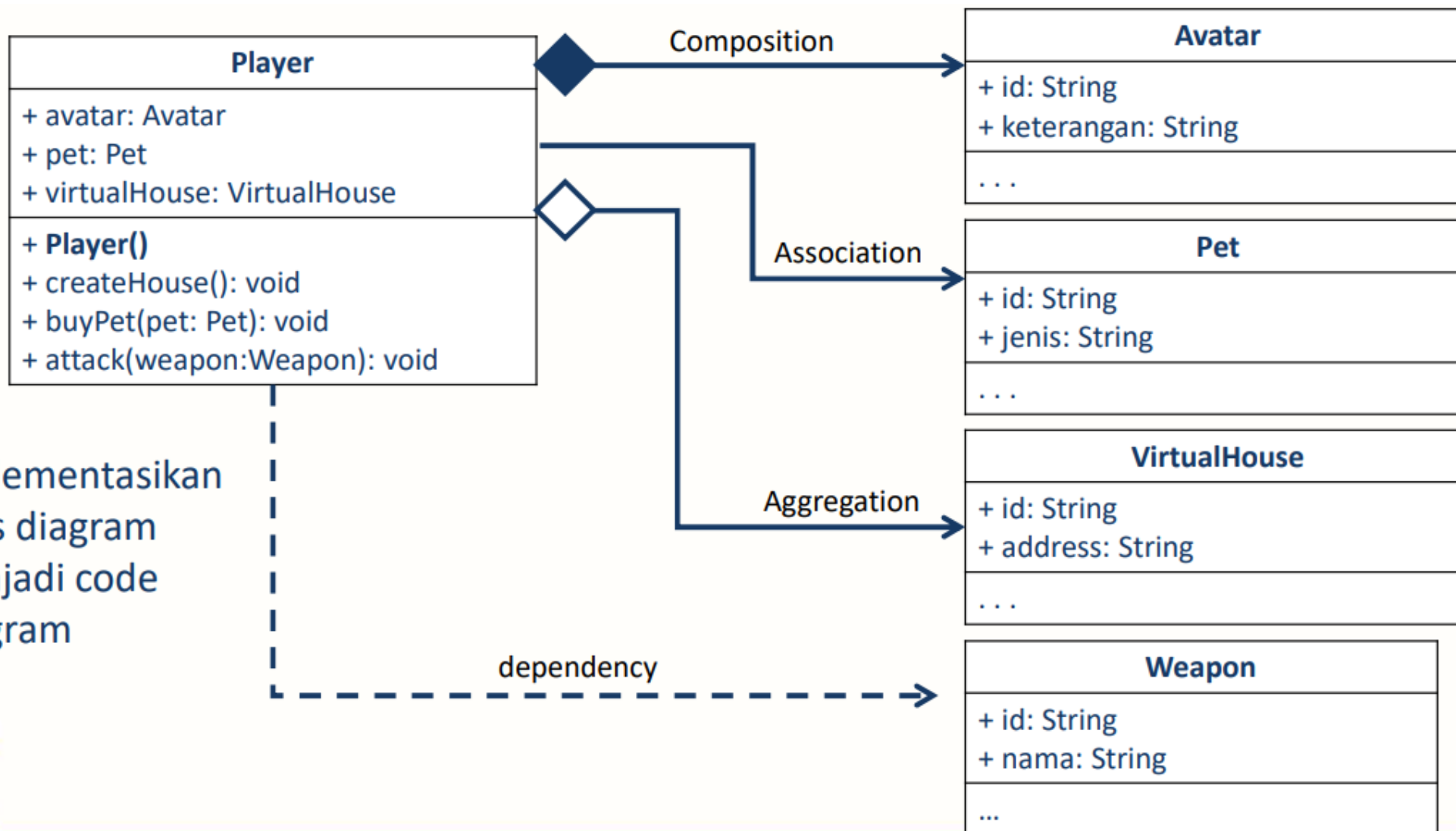


Implementasikan class diagram tersebut menjadi code program

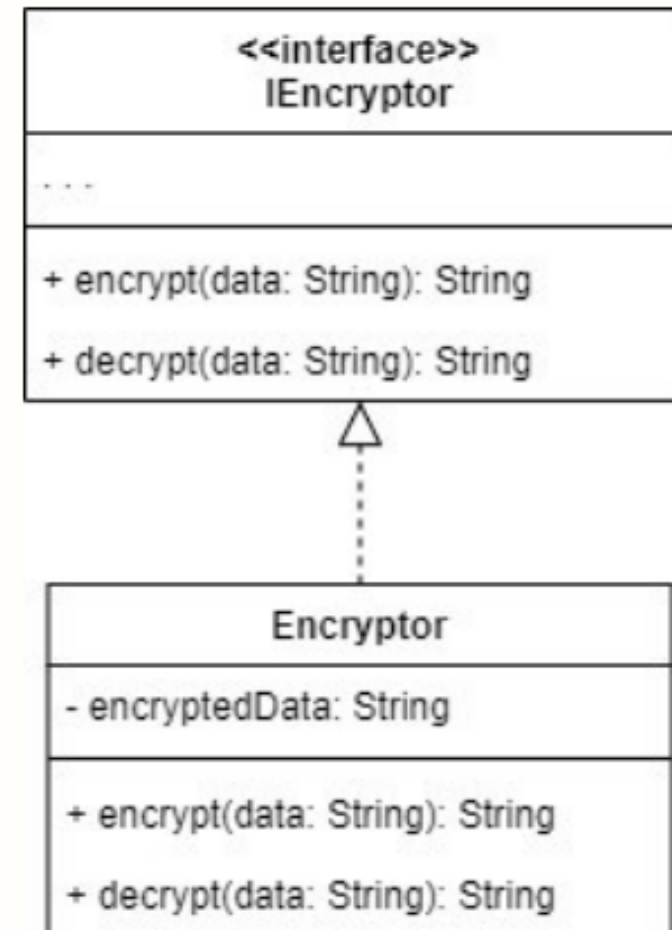
Gambarkan class diagram dari code program berikut

```
public class Car{  
    public String id;  
    public String type;  
  
    public void drive(Gasoline gasoline){  
        ...  
    }  
}
```

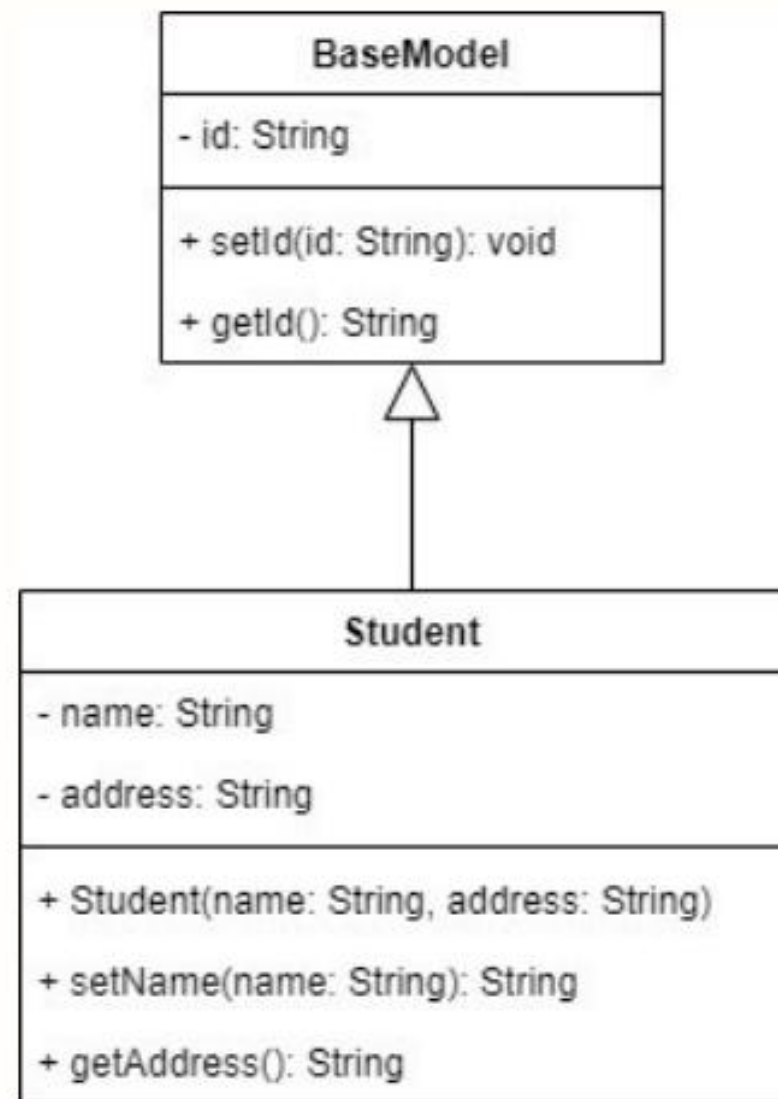
```
public class Gasoline{  
    public Long octane;  
    public Long volume;  
  
    public String runEngine(){ ... }  
}
```



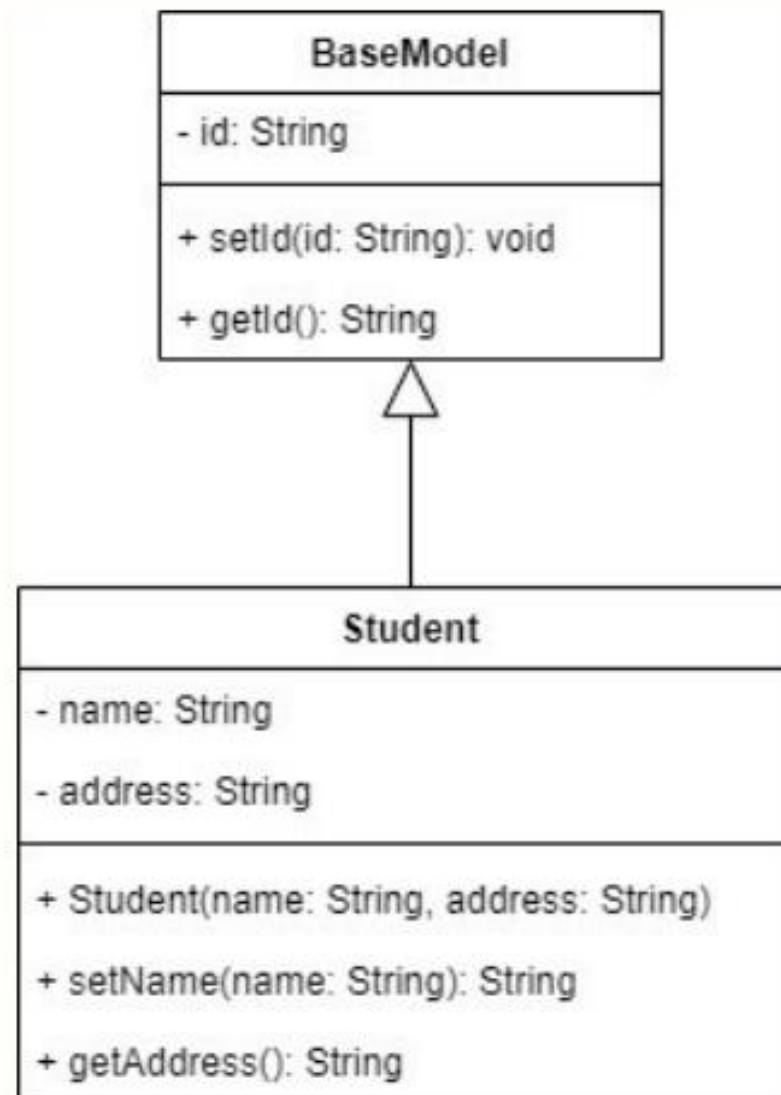
Implementasikan  
menjadi kode program



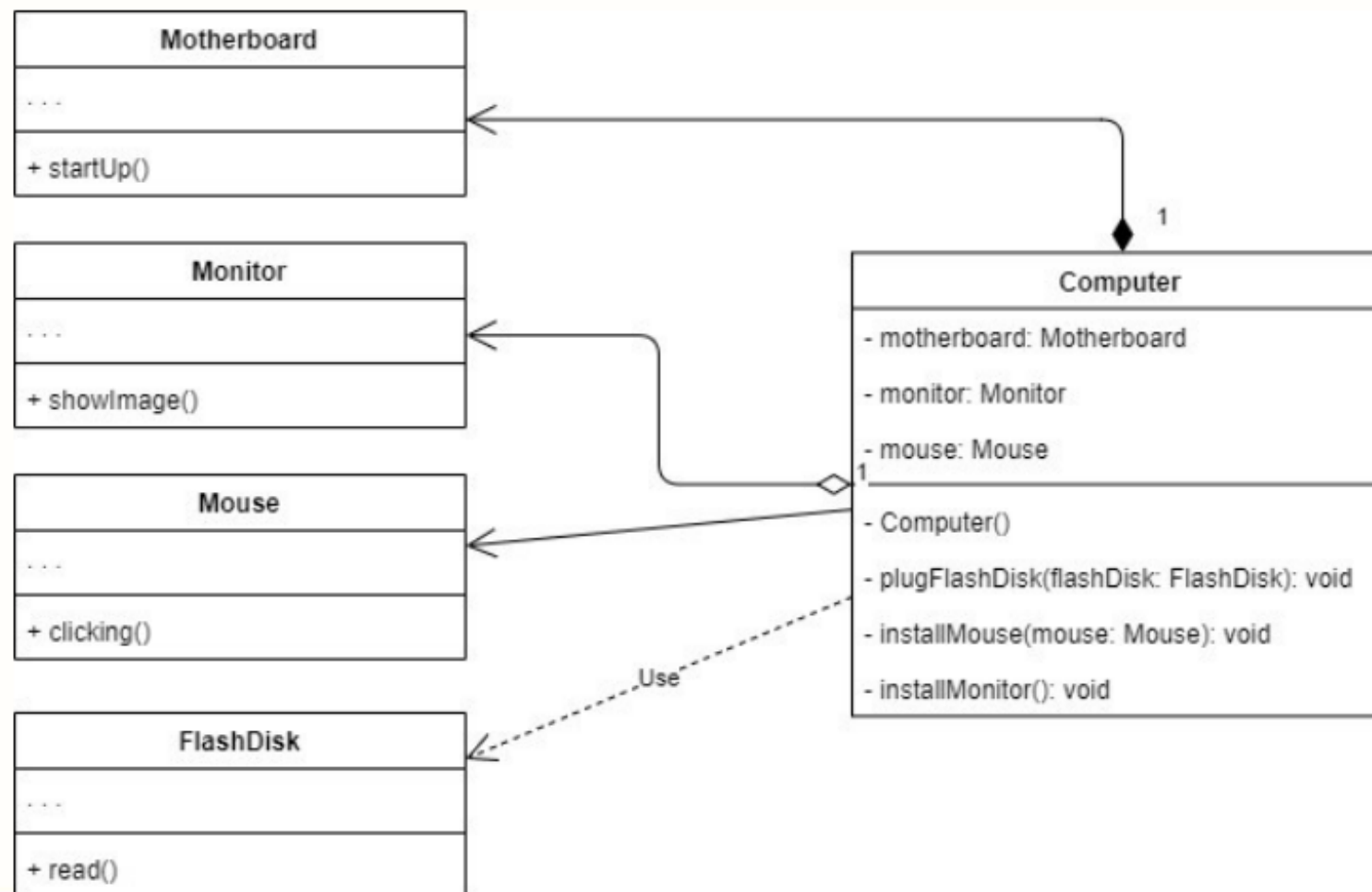
Implementasikan  
menjadi kode program



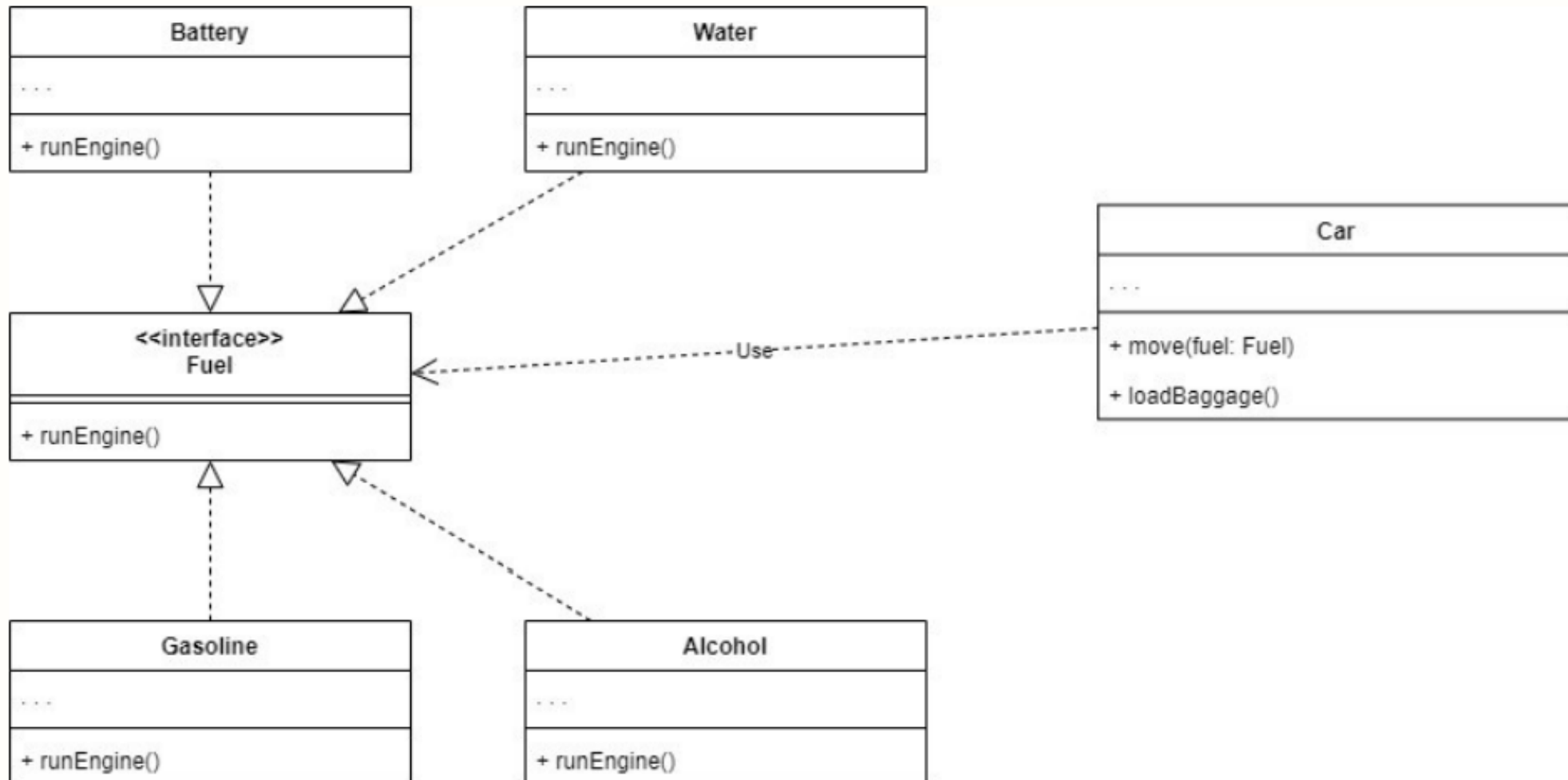
# Implementasikan menjadi kode program



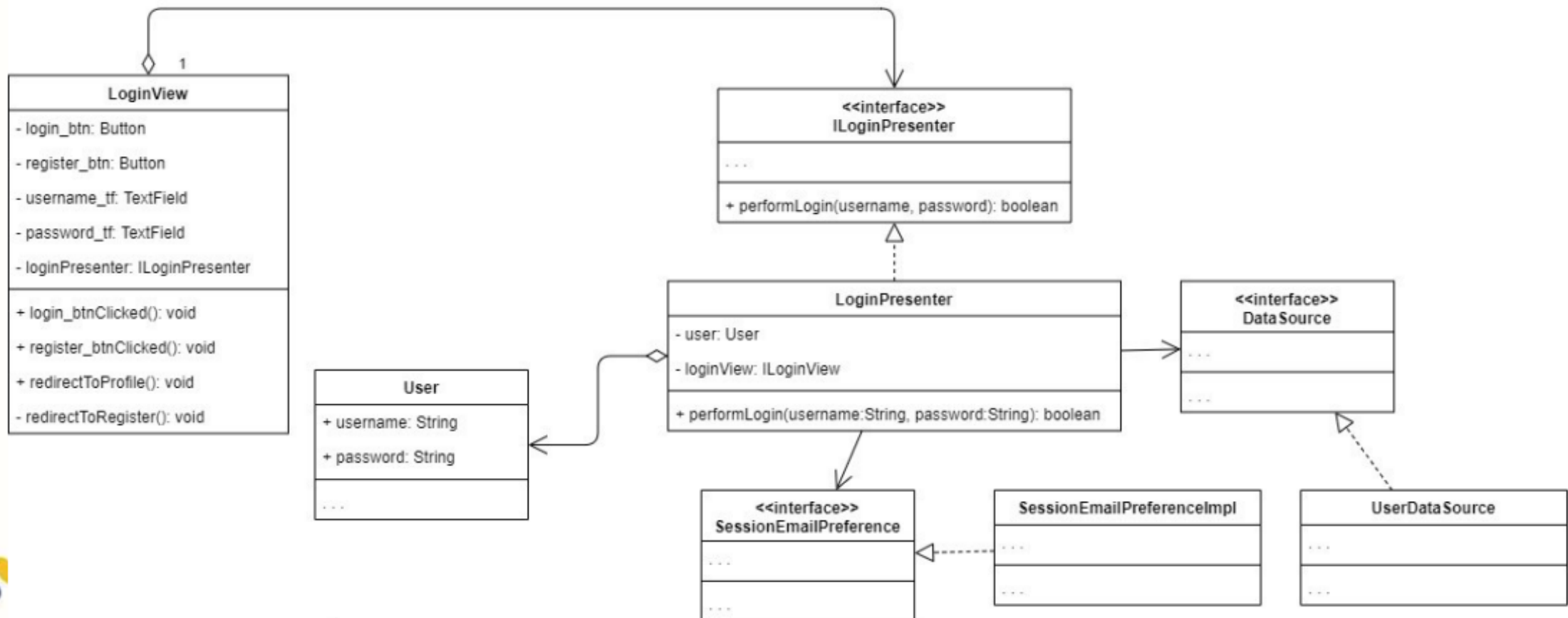
## Implementasikan menjadi kode program



## Implementasikan menjadi kode program



## Implementasikan menjadi kode program





1. Oracle Java Documentation, The Java™ Tutorials, <https://docs.oracle.com/javase/tutorial/>, Copyright © 1995, Oracle 2015.
2. Tita Karlita, Yuliana Setrowati, Rizky Yuniar Hakkun, Pemrograman Berorientasi Obyek, PENS-2012
3. Sun Java Programming, Sun Educational Services, Student Guide, Sun Microsystems, 2001.  
**bridge to the future**
4. John R. Hubbard, Programming With Java, McGraw-Hill, ISBN: 0-07-142040-1, 2004.
5. Patrick Niemeyer, Jonathan Knudsen, Learning Java, O'reilly, CA, ISBN: 1565927184, 2000.
6. Philip Heller, Simon Roberts, Complete Java 2 Certification Study Guide, Third Edition, Sybex, San Francisco, London, ISBN: 0-7821-4419-5, 2002.
7. Herbert Schildt, The Complete Reference, Java™ Seventh Edition, Mc Graw Hill, Osborne, ISBN: 978-0-07-163177-8, 2007