Pemrograman Berorientasi Obyek

Class Diagram

Oleh Politeknik Elektronika Negeri Surabaya
2020

Politeknik Elektronika Negeri Surabaya
Departemen Teknik Informatika dan Komputer

CLASS DIAGRAM

OOP

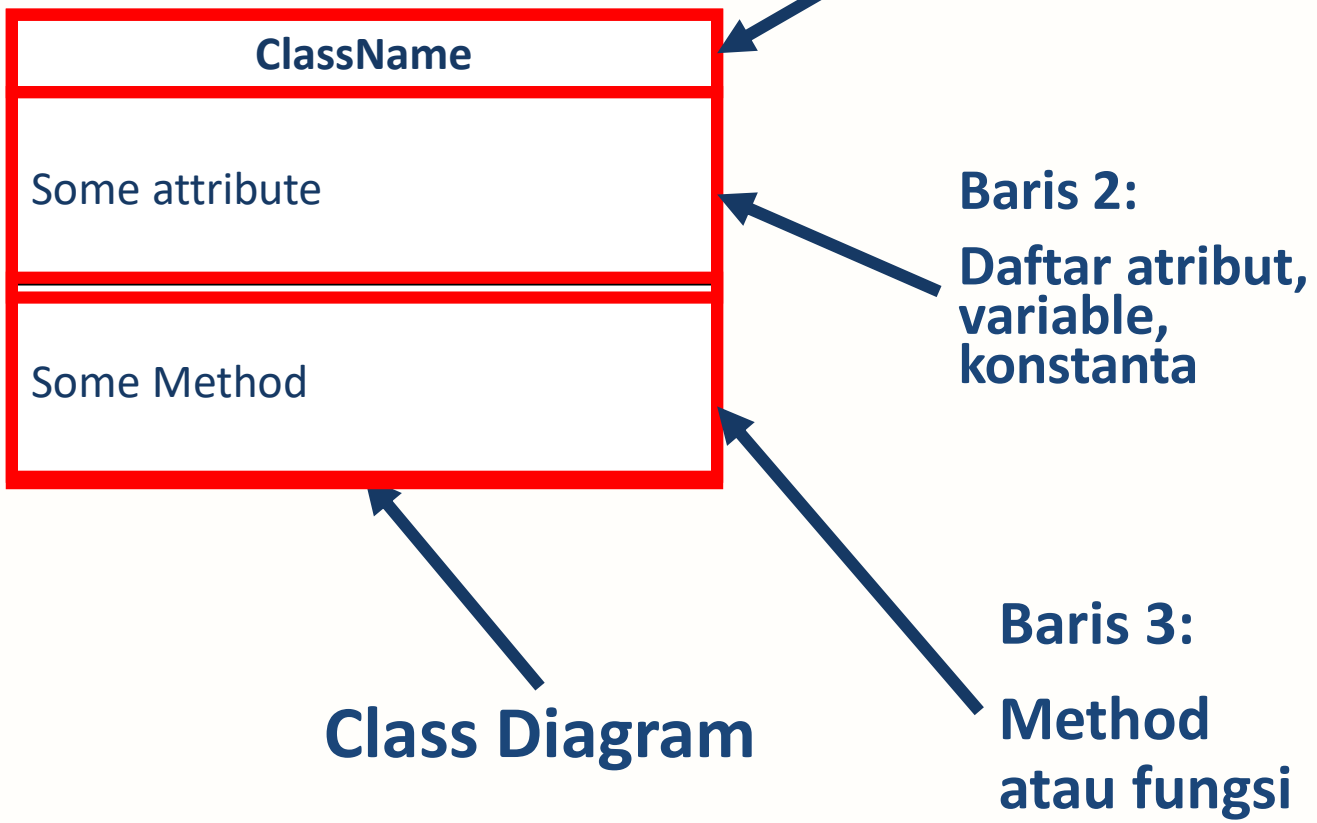
tidak mungkin terlepas dari

Class Diagram

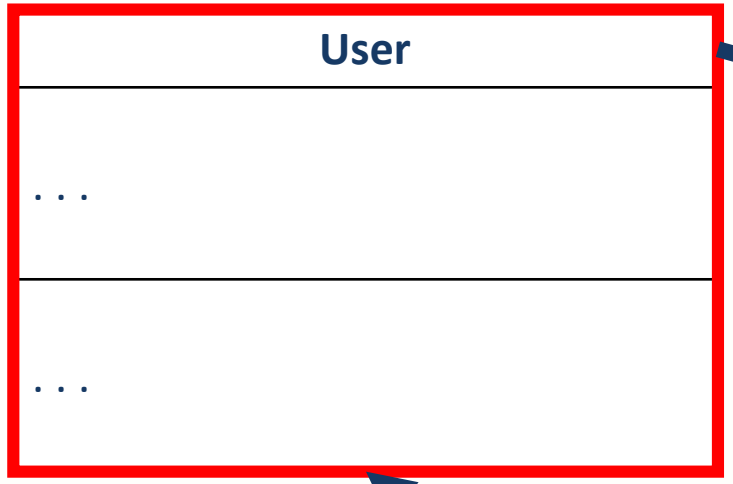
CLASS DIAGRAM

Apa itu **Class Diagram**?

Diagram yang menggambarkan **Class**



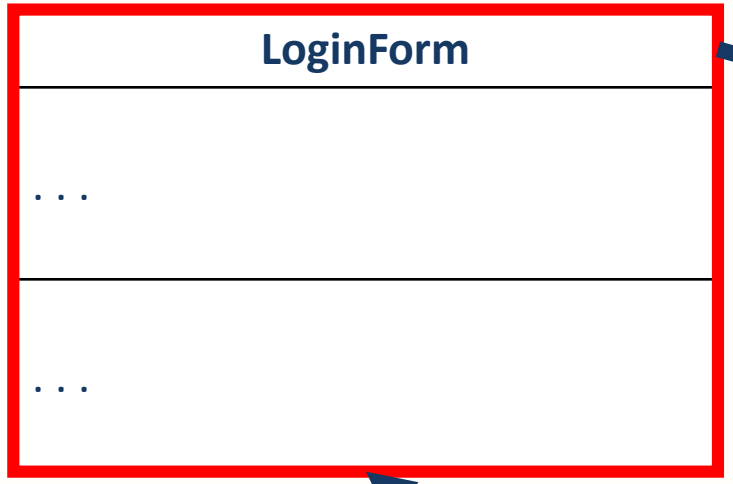
CLASS NAME



Jika di implementasikan dalam Bahasa java:

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

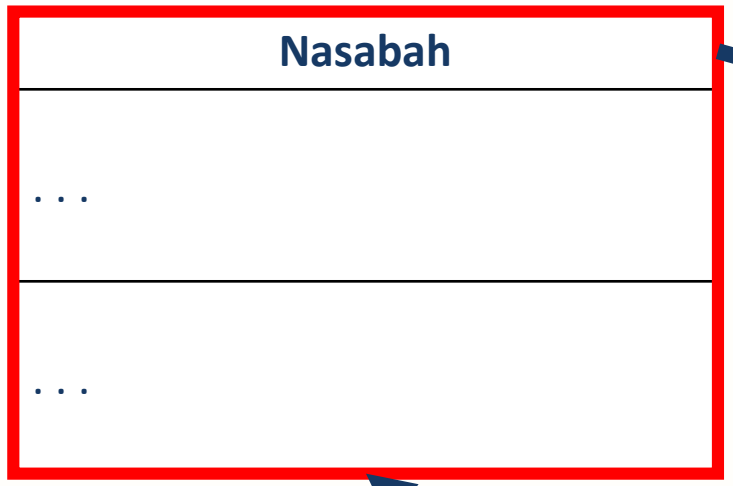
Ini adalah satu buah class dengan nama class : **User**



Jika di implementasikan dalam Bahasa java:

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

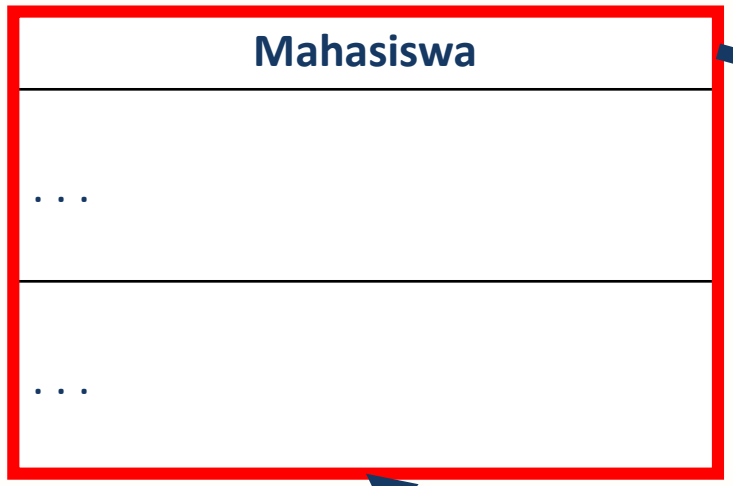
Ini adalah satu buah class dengan nama class : **LoginForm**



Jika di implementasikan dalam Bahasa java:

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

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

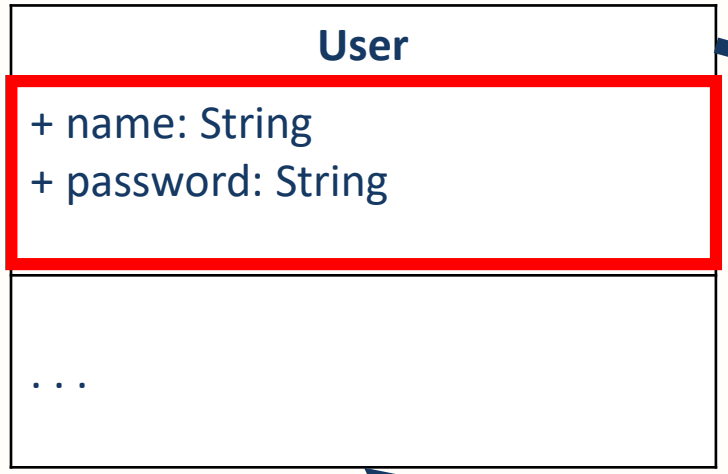


Jika di implementasikan dalam Bahasa java:

. . . ?

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

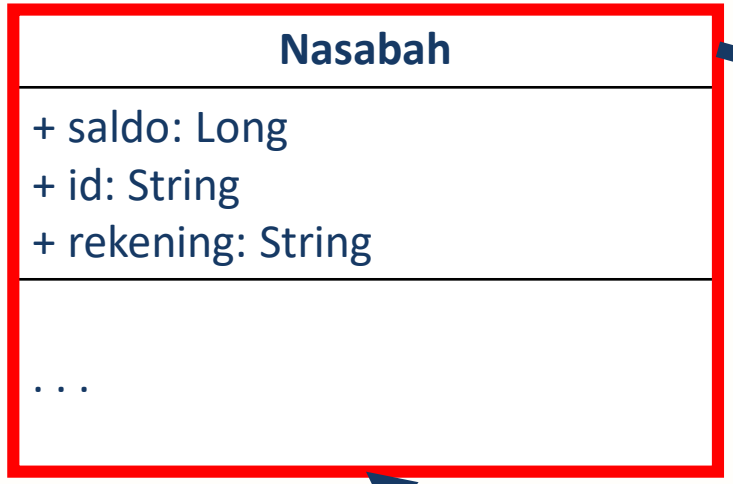
ATTRIBUTE



Dalam class User terdapat dua variable : name dan password

Jika di implementasikan dalam Bahasa java:

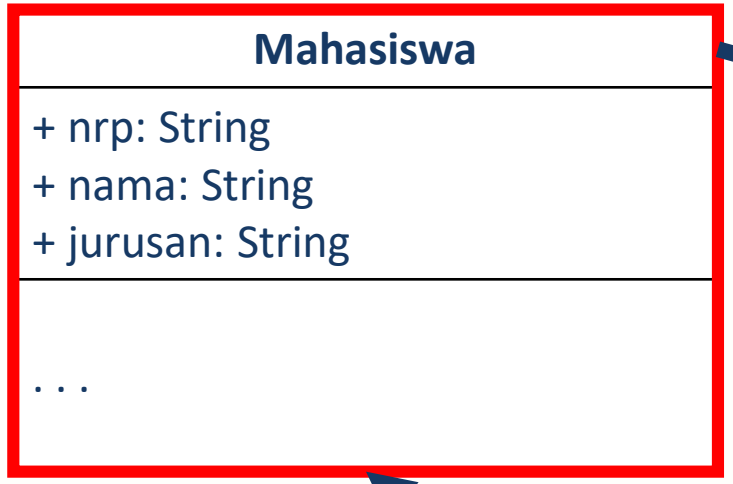
```
public class User{
    public String name;
    public String password;
}
```



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;
}
```

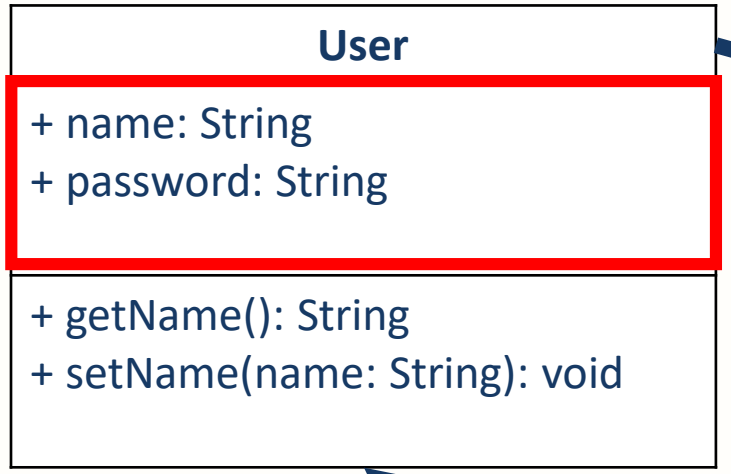


Jika di implementasikan dalam Bahasa java:

. . . ?

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

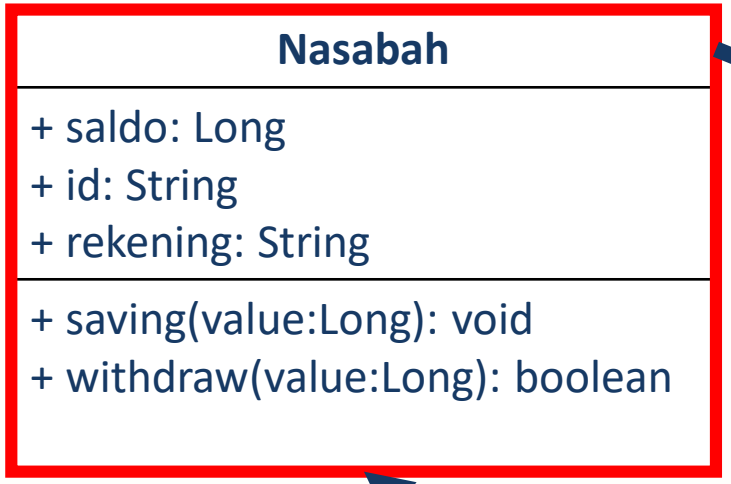
METHOD



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

Jika di implementasikan dalam Bahasa java:

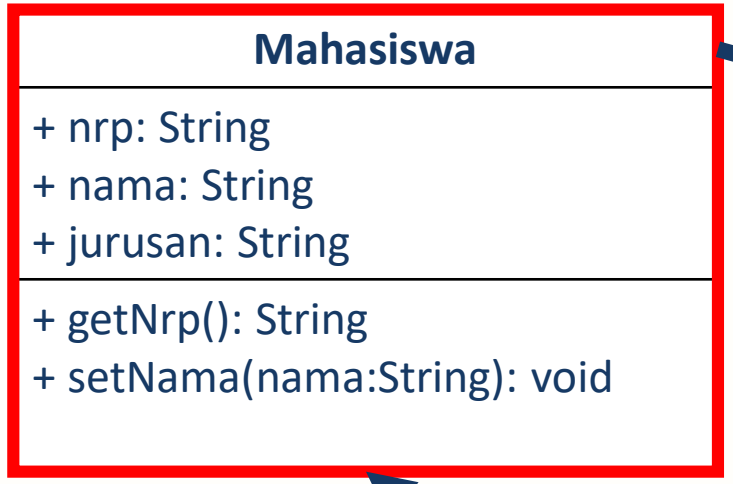
```
public class User{
    public String name;
    public String password;

    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 setNama()

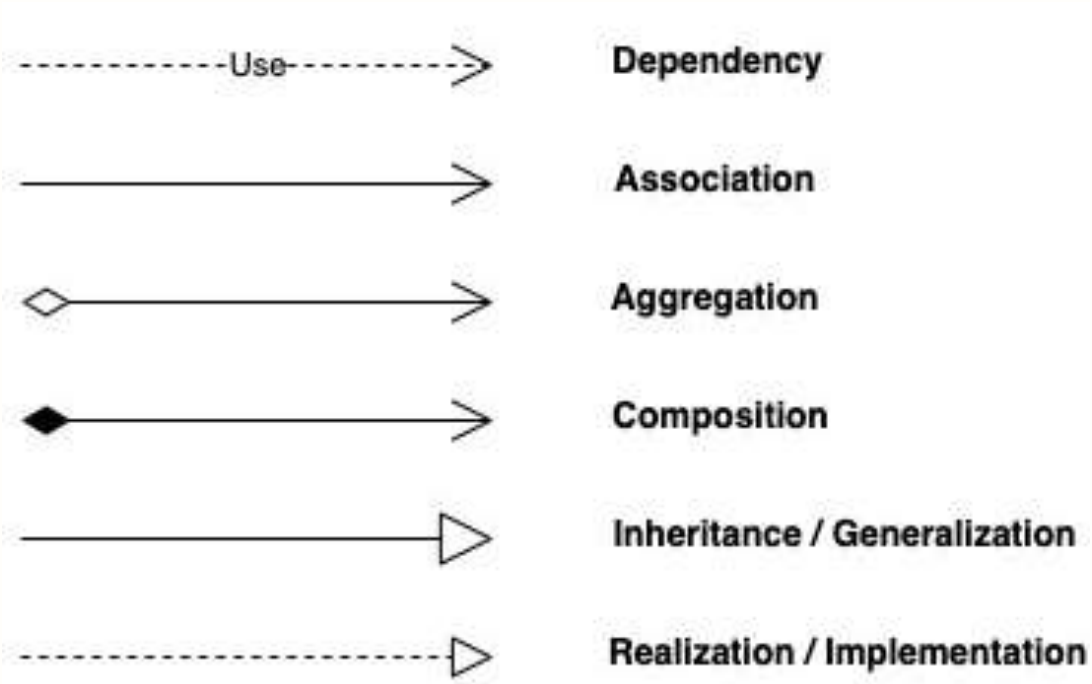
MODIFIER

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

Mahasiswa
+ nrp: String - nama: String # jurusan: String ~ wali: String
. . .

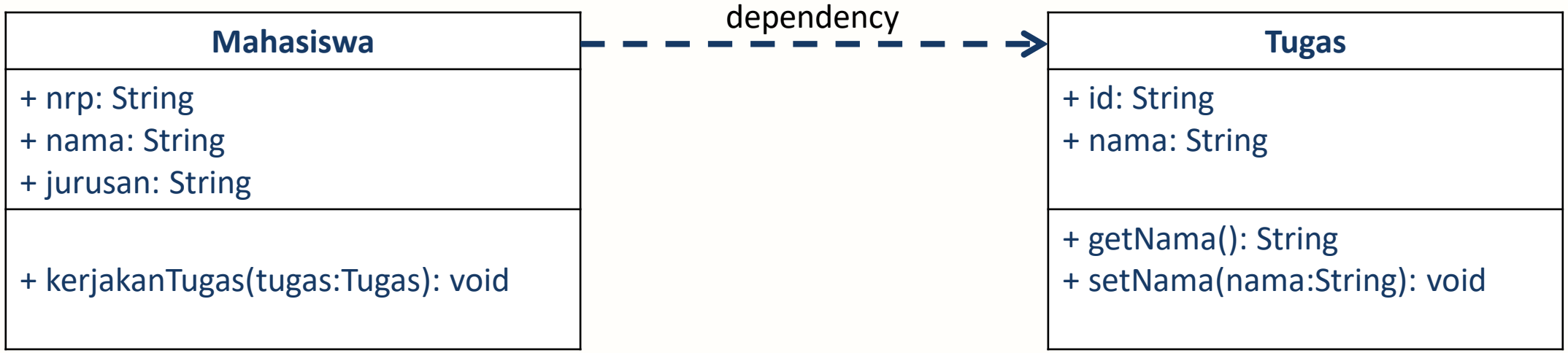
CLASS DIAGRAM RELATIONSHIP

CLASS DIAGRAM RELATIONSHIP



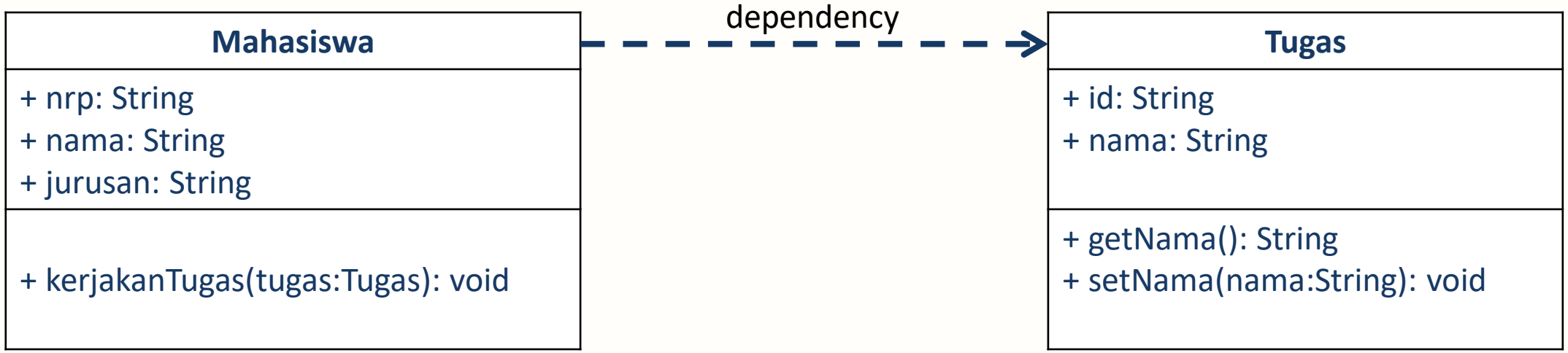
CLASS DIAGRAM RELATIONSHIP

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 getNama(){ … }
    public void setNama(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.

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;
    }
}
```

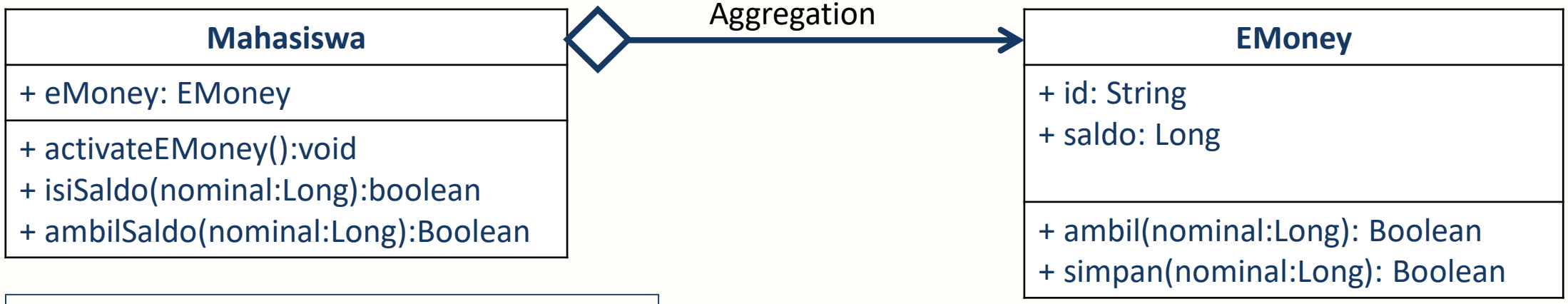
```
public class Jurusan{
    public String id;
    public String nama;

    public String getNama(){ … }
    public void setNama(String nama){ … }
}
```


```
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** dalm 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.

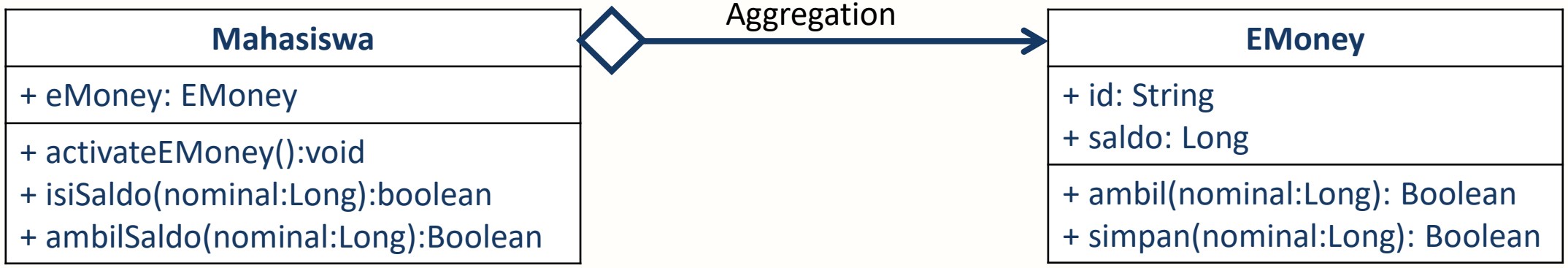
CLASS DIAGRAM RELATIONSHIP

Aggregation



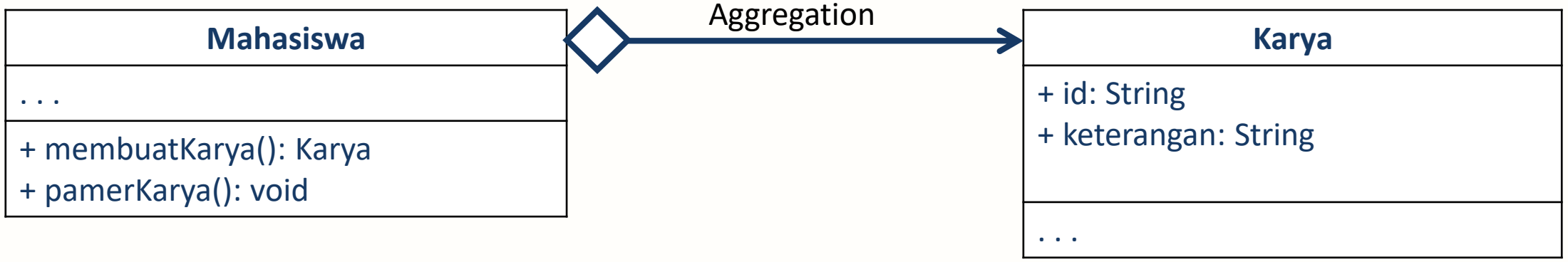

```
public class Mahasiswa{
    public Emoney eMoney;
    public void activateEMoney(){
        this.eMoney = new EMoney();
    }
    public Boolean isiSaldo(Long nominal){
        return this.eMoney.simpan(nominal);
    }
    public Boolean ambilSaldo(Long nominal){
        return this.eMoney.ambil(nominal);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

```
public class EMoney{
    public String id;
    public Long saldo;

    public boolean ambil(Long Nominal){ … }
    public boolean simpan(Long nominal){ … }
}
```




```
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();




```
public class Mahasiswa{

    public Karya membuatKarya(){
        Karya karya = new Karya();
        return karya;
    }
    public void pamerKarya(){
        cetak(new Karya());
    }
}
```

```
public class Karya{
    public String id;
    public String keterangan;

}
```

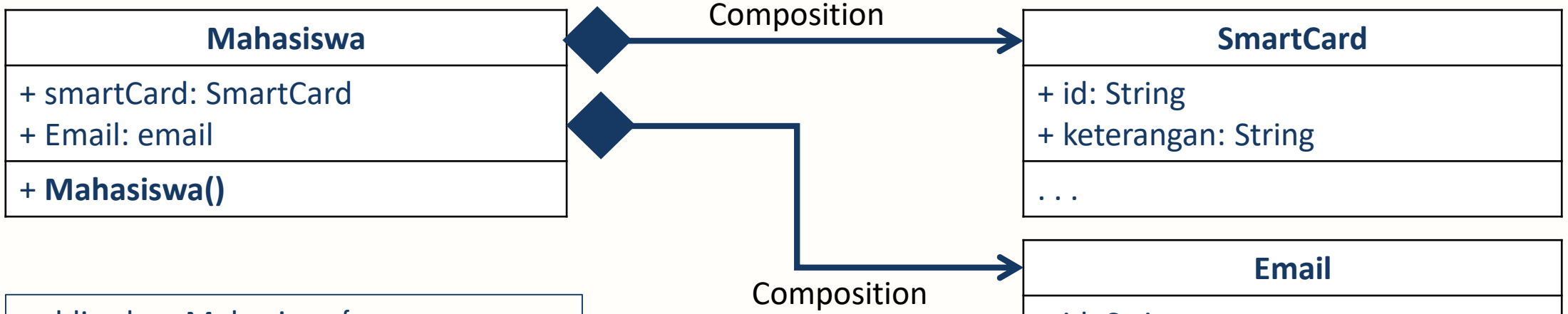

```
public class Mahasiswa{

    public Karya membuatKarya(){
        Karya karya = new Karya();
        return karya;
    }
    public void pamerKarya(){
        cetak(new Karya());
    }
}
```

Aggregation versi 2

- Karya menjadi sebuah object/variable local dalam method pada kelas Mahasiswa
- Karya menjadi **tidak variable global** dalam kelas Mahasiswa
- Kelas mahasiswa **melakukan instansiasi** object EMoney instansiasi → **new EMoney()**

CLASS DIAGRAM RELATIONSHIP

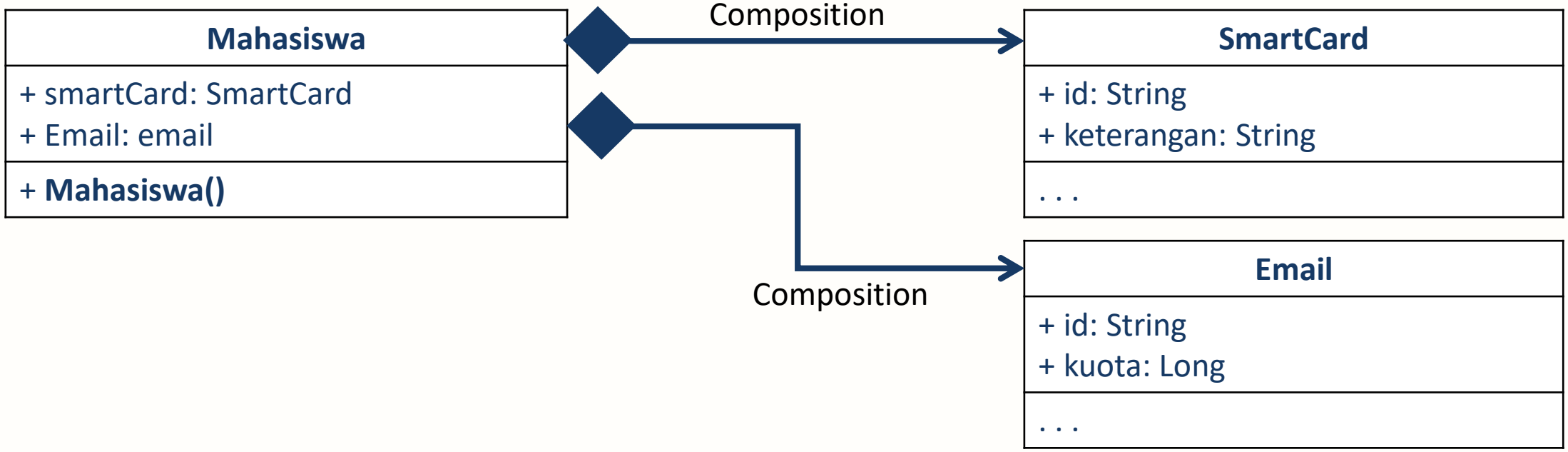
Composition




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

```
public class SmartCard{
    public String id;
    public String keterangan;
}
```

```
public class Email{
    public String id;
    public Long kuota;
}
```




```
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

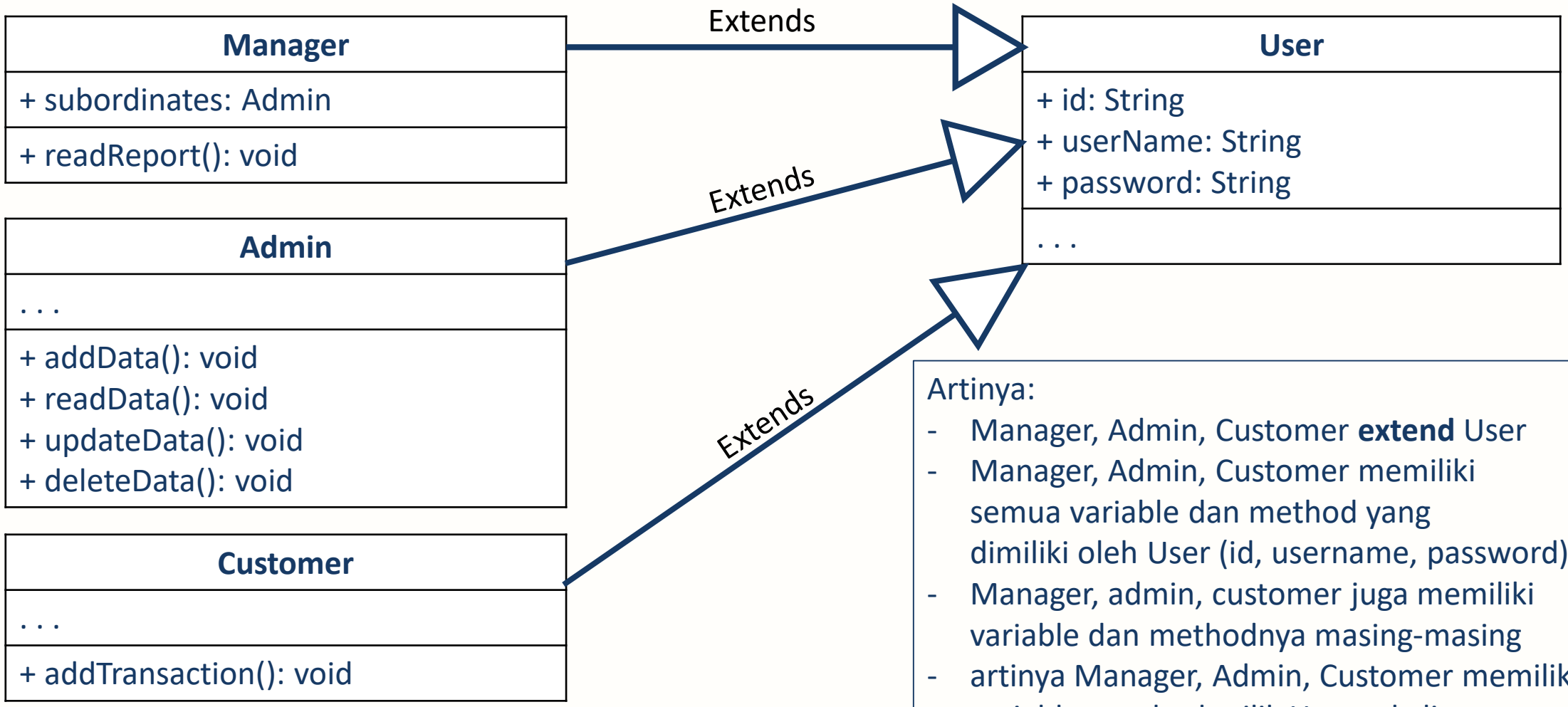
CLASS DIAGRAM RELATIONSHIP

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 paret 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 variablenya 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

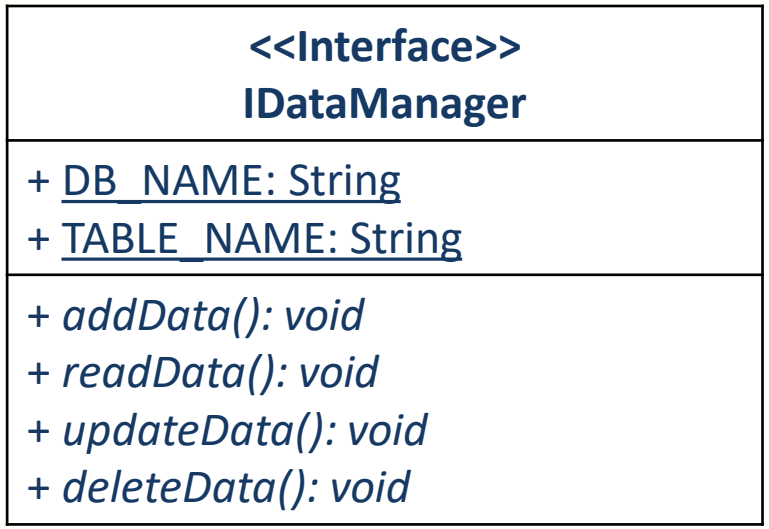
- Realization terjadi antara **Class** dengan **Interface.**
- Sama dengan class, **Interface dapat memiliki variables dan methods**.
- Namun Interface hanya bisa memiliki **method Abstract**.
- Method abstract adalah method kosong tanpa method body. Method ini hanya deklarasi saja. Contoh:

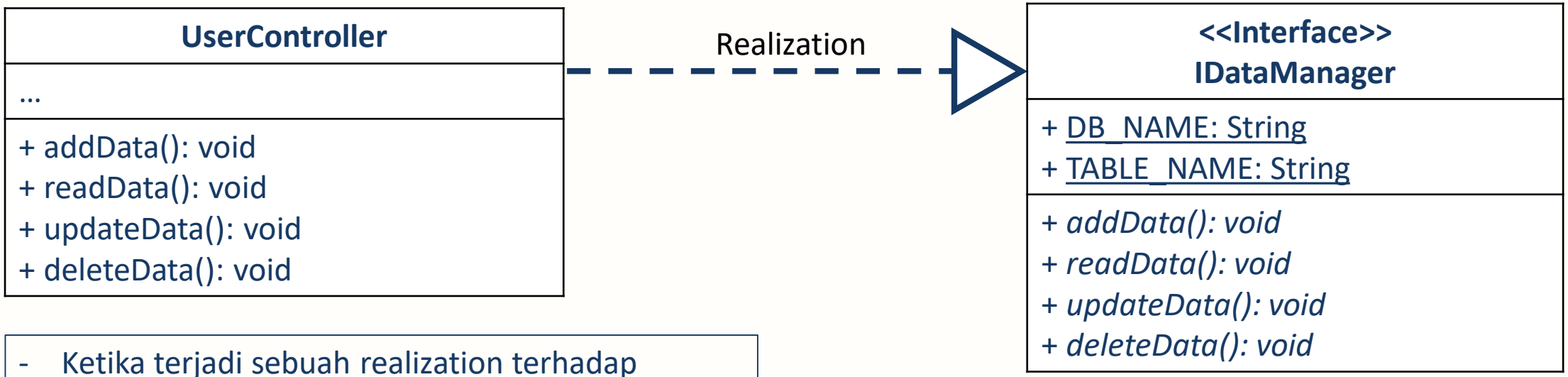
```
public int getData();
public void addData();
public Boolean edit(int data);
```

- Variable yang dideklarasikan dalam Interface harus **public, static & final**

Realization

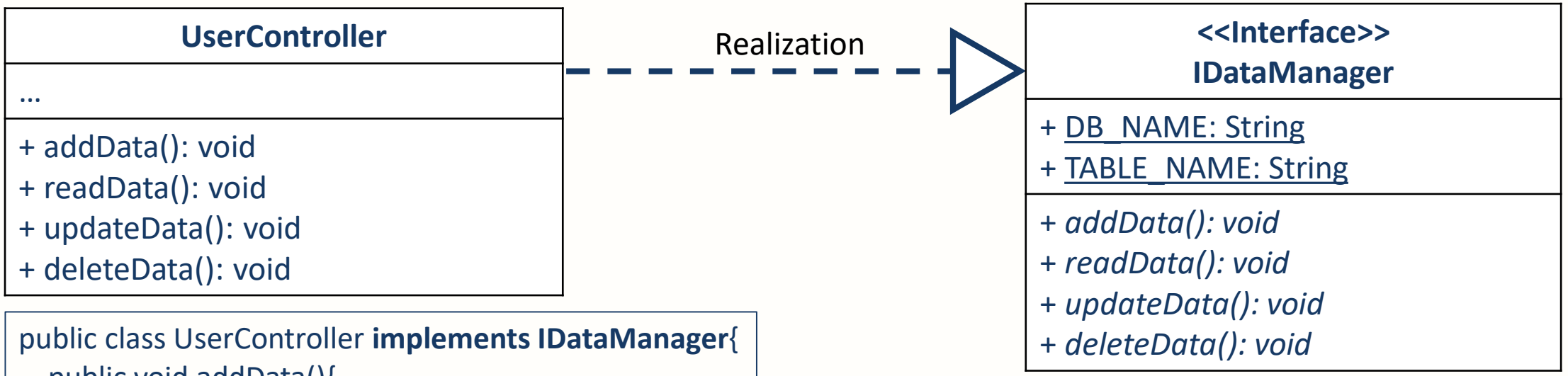
- Menggambarkan Interface dalam class diagram sedikit berbeda dengan class yaitu dengan menambahkan kata interface diatas nama interface
- variable static dituliskan dengan tanda underline/garis bawah
- Variable final dituliskan dengan huruf Capital semua. Dengan pemisah antar kata menggunakan underscore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





- 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 concreate/nyata.

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




```
public class UserController implements IDataManager{
    public void addData(){
        ....
    }
    public void addData(){
        ....
    }
    public void addData(){
        ....
    }
    public void addData(){
        ....
    }
}
```

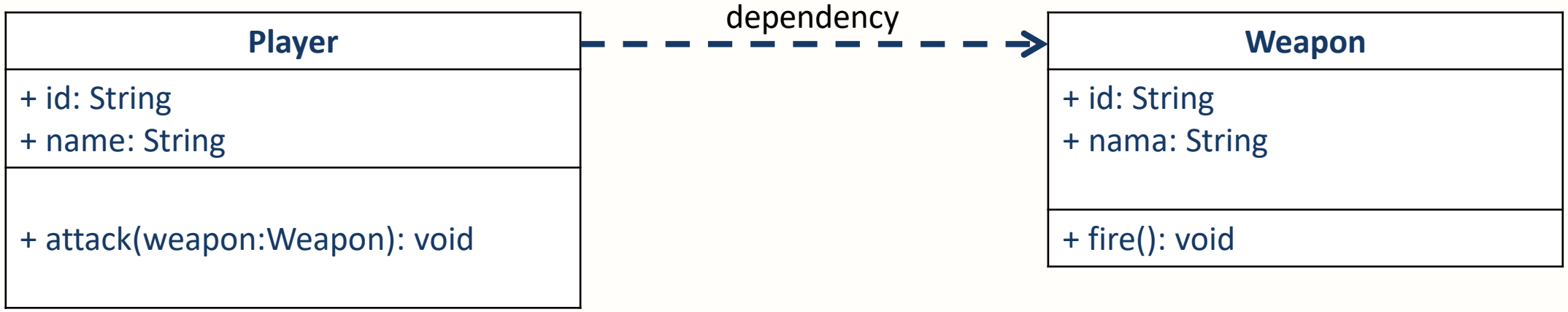
```
public Interface IDataManager{
    public static final String DB_NAME:
    public static final String TABLE_NAME;
    public void addData();
    public void addData();
    public void addData();
    public void addData();
}
```

bridge to the future

http://www.eepis-its.edu

CLASS DIAGRAM

LATIHAN

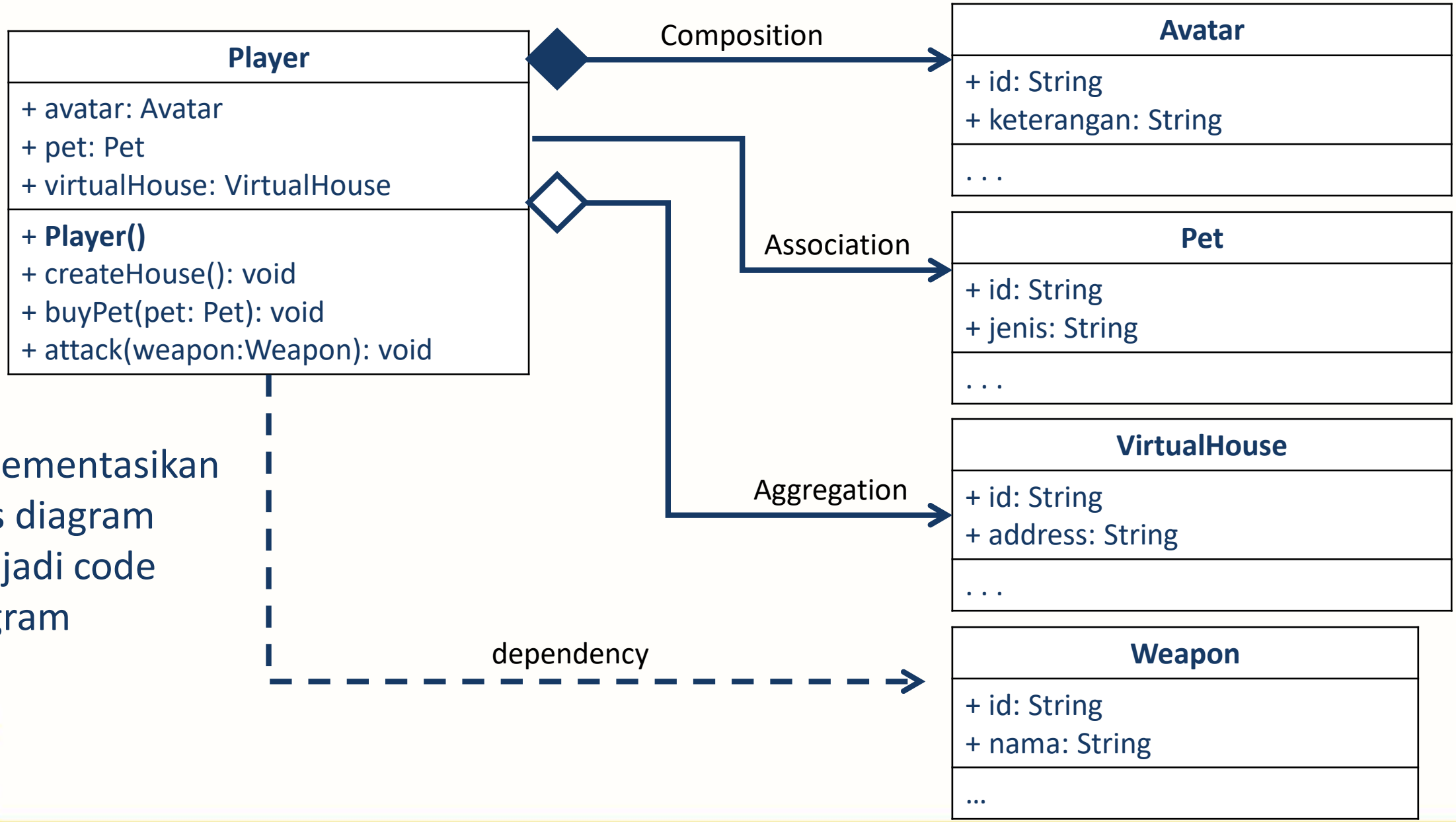


Implementasikan class diagram tersebut menjadi code program

Gambarkan class diagram dari code program berikut

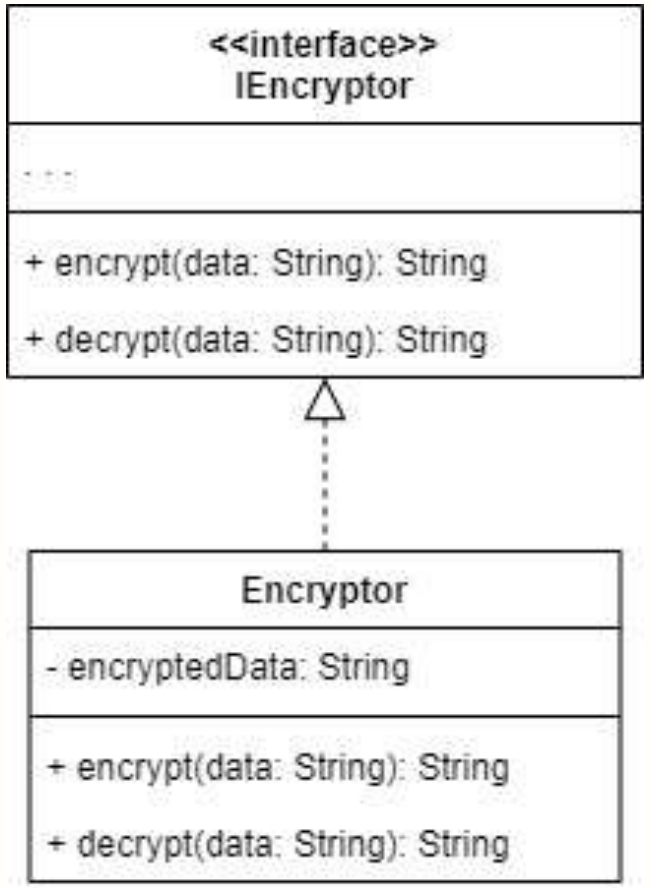
```
public class Car{
    public String id;
    public String type;

    public void drive(Gasoline gasoline){
        …
    }
}
```

```
public class Gasoline{
    public Long octane;
    public Long volume;

    public String runEngine(){ … }
}
```

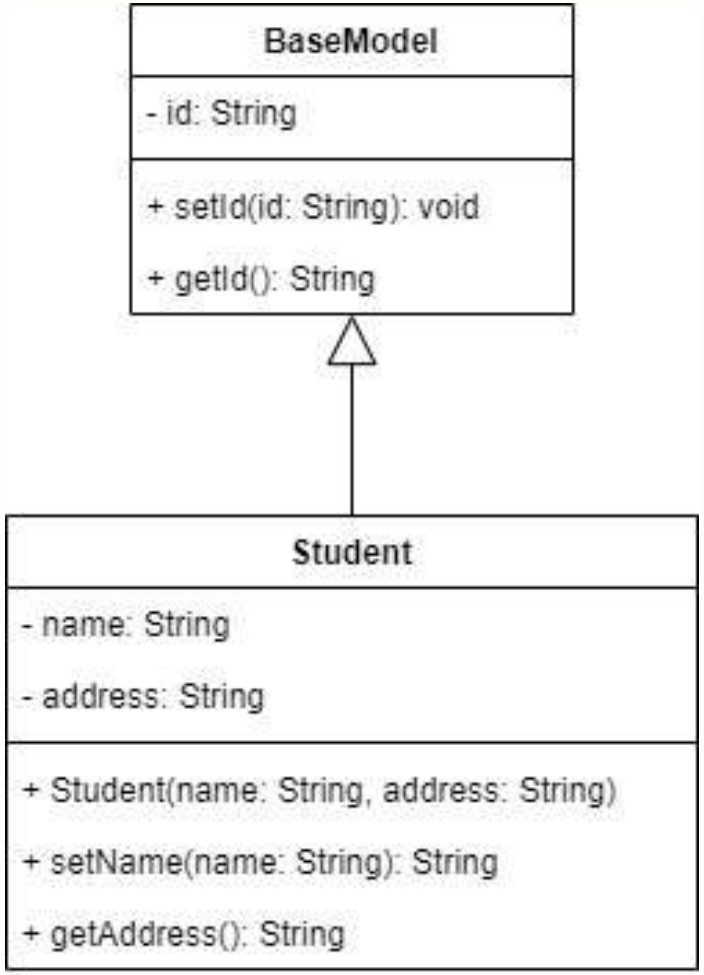


Implementasikan class diagram menjadi code program

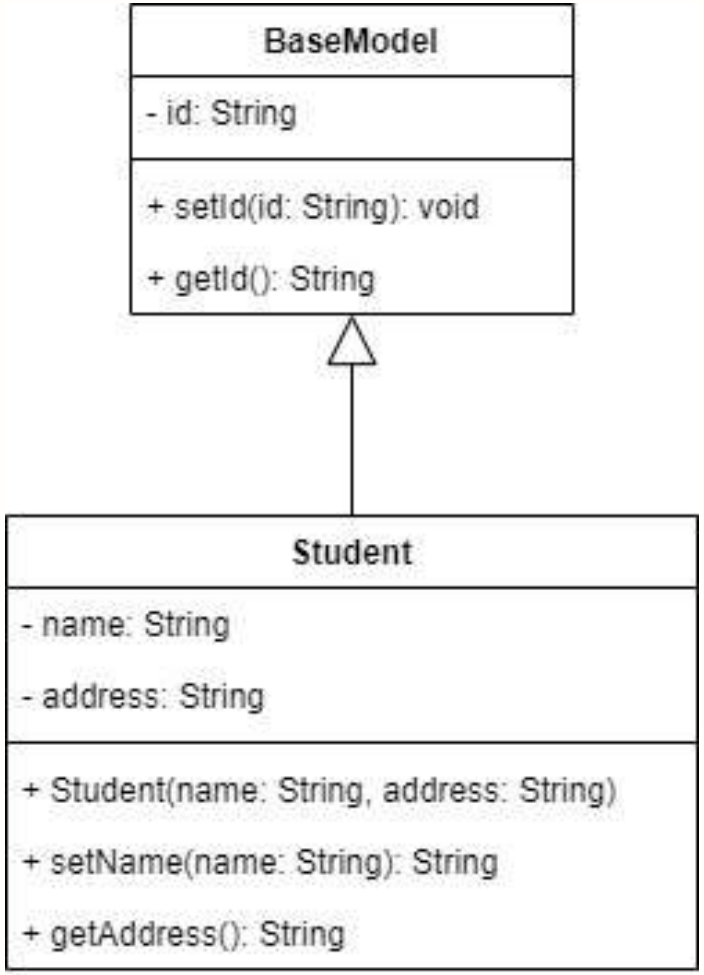
Implementasikan menjadi kode program



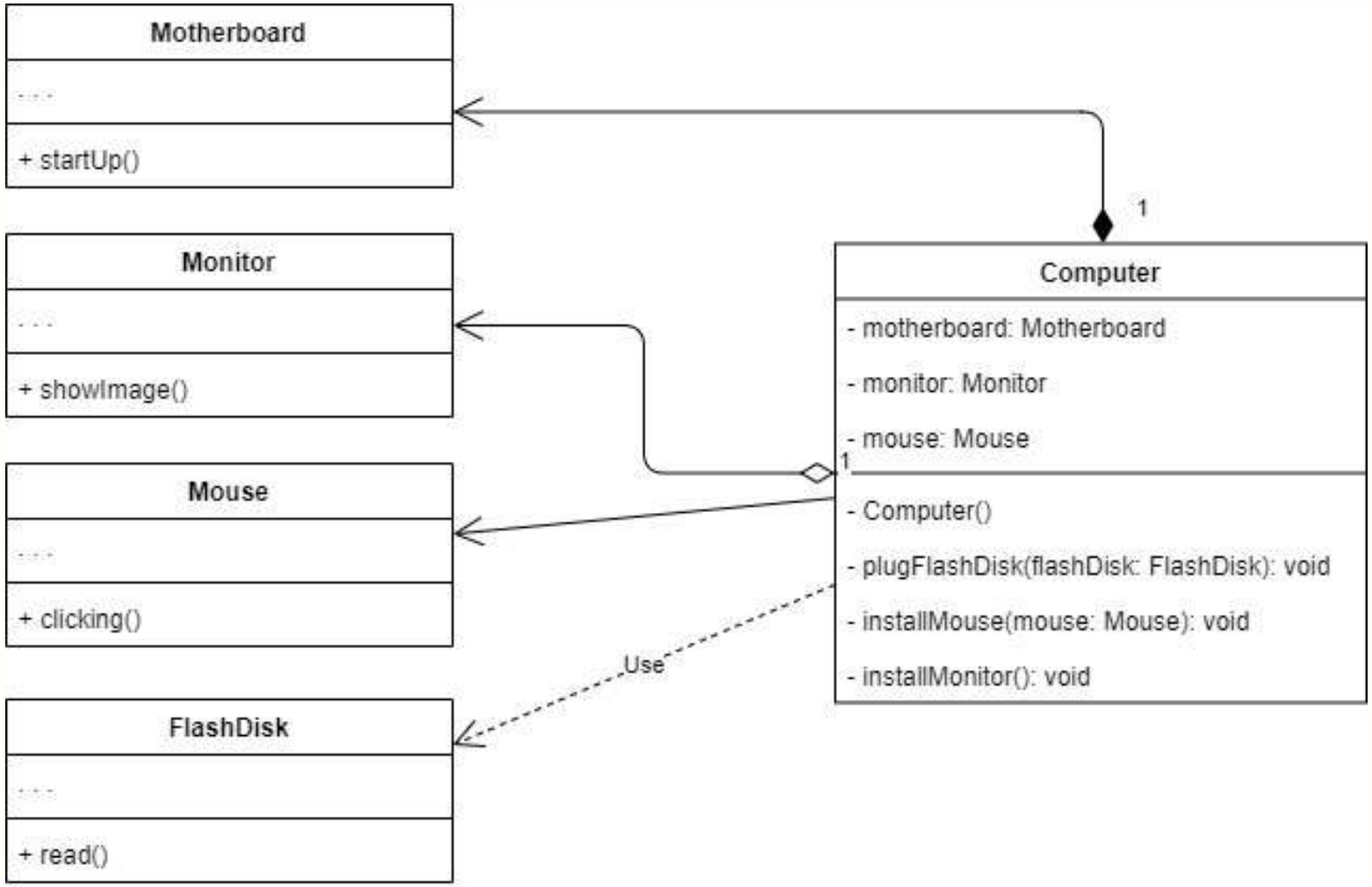
Implementasikan menjadi kode program



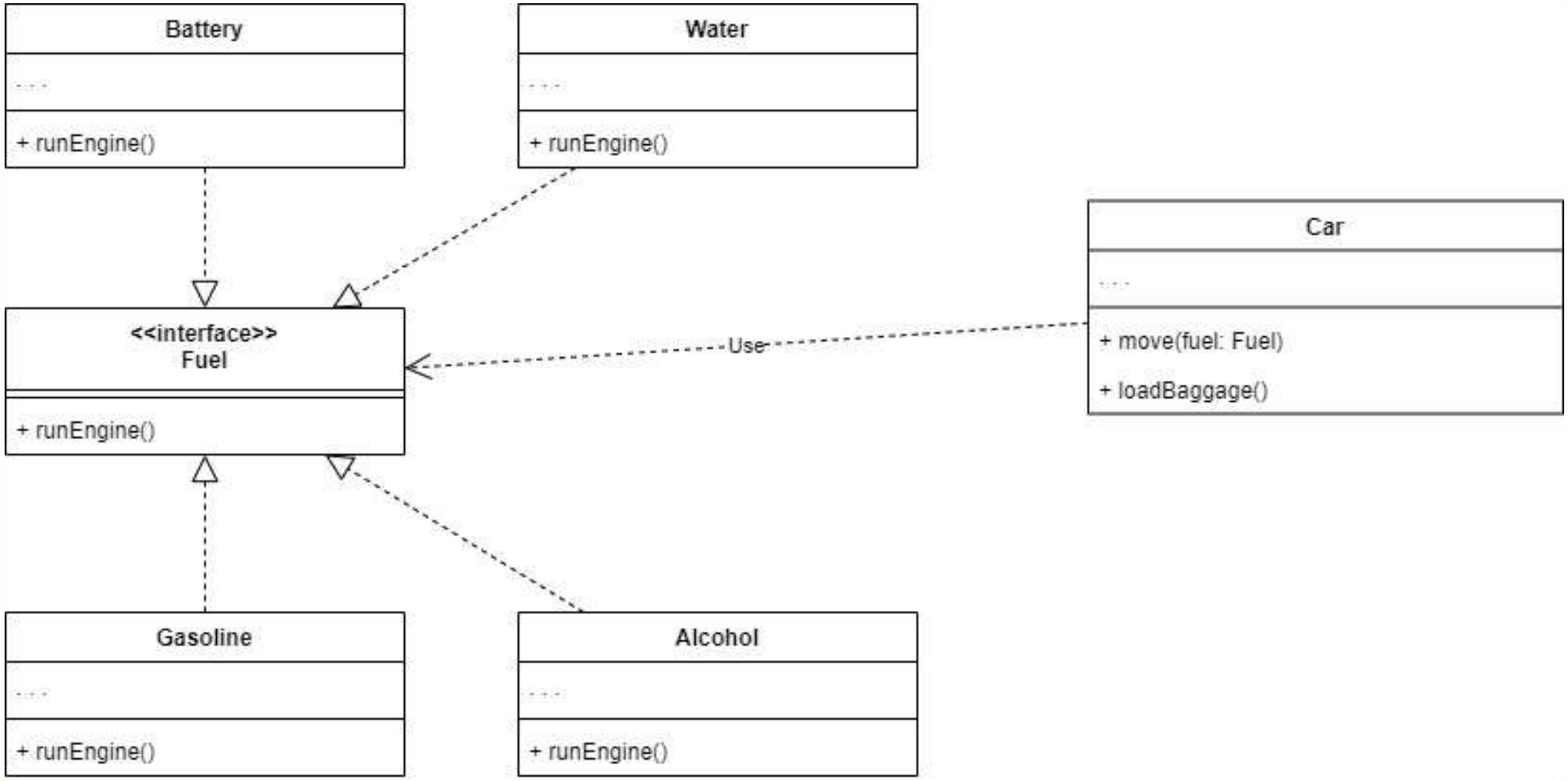
Implementasikan menjadi kode program



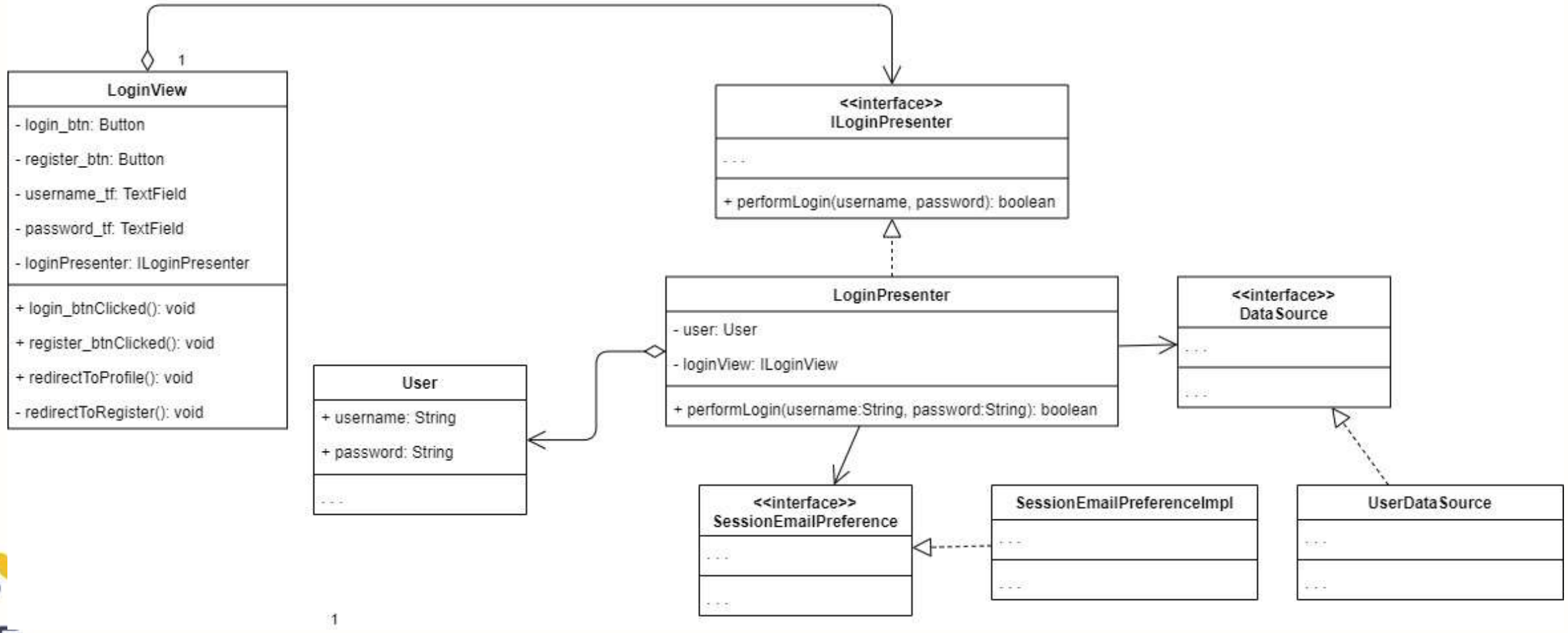
Implementasikan menjadi kode program



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.
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