ABSTRACT CLASS DAN INTERFACE

1. Kompetensi

Setelah menempuh materi percobaan ini, mahasiswa mampu mengenal:

1. Konsep abstract class
2. Abstract method
3. Notasi UML untuk abstract class
4. Konsep interface
5. Beda interface dengan abstract class
6. Cara membuat class yang implements ke interface
7. Notasi UML untuk interface dan relasi implements

2. Pendahuluan

Abstract class memiliki banyak kemiripan dengan interface. Yaitu sama-sama tidak bisa di-instansiasi dan keduanya dapat memiliki method yang dideklarasikan tanpa atau dengan implementasi.

Pada abstract class:

- Kita bisa membuat property yang tidak static dan final.
- Kita bisa membuat method yang public, protected dan private

Pada interface:

- Semua property secara otomatis public, static dan final.
- Semua method adalah public

Abstract class hanya bisa diwariskan satu kali (single inheritance) sedangkan interface bisa diimplementasikan lebih dari satu.

Kapan menggunakan abstract class atau interface?

Gunakan abstract class jika:

- Kita ingin berbagi code/mewariskan code yang sama dengan class-class yang berhubungan.
- Kita ingin membuat beberapa class dengan method dan property yang hampir sama.
- Kita ingin mendeklarasikan property yang tidak static dan tidak final. Dengan cara ini kita bisa mendefinisikan method dengan implementasi yang berbeda-beda di tiap class yang mewariskan abstract class tersebut.

Gunakan interface jika:

- Kita ingin mengimplementasikan interface ke beberapa class yang tidak berhubungan secara hirarki.
- Kita ingin menspesifikasikan behavior dari beberapa object, namun tidak mau tau bagaimana implementasi dari behavior tersebut.
- Kita ingin memanfaatkan multiple inheritance

3. Abstract Class

Abstract class adalah sebuah class yang dideklarasikan sebagai *abstract*. Didalamnya bisa terdapat method biasa, dan bisa terdapat abstract method. Abstract class tidak bisa di-instansiasi (dibuat object nya). Untuk membuat object dari abstract class tersebut, pertama-tama harus diturunkan terlebih dulu (diwariskan). Cara membuat abstract class:

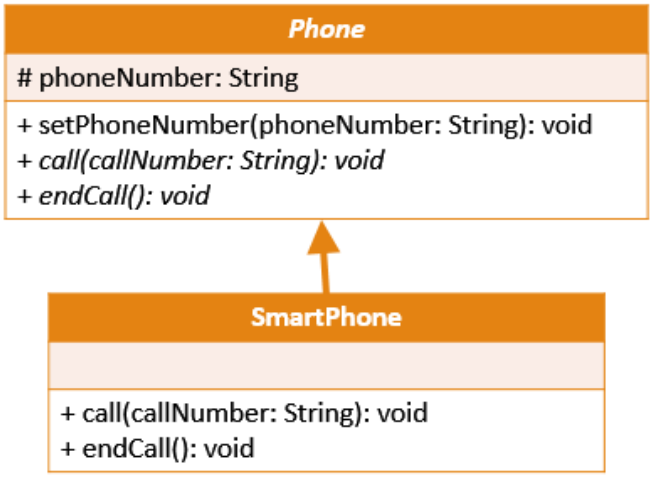
```
3   abstract class Phone
4   {
5       protected String phoneNumber;
6   }
```

Abstract method adalah method yang hanya dideklarasikan dan tanpa implementasi (diakhiri dengan titik koma di akhir deklarasi, dan tidak ada kurung kurawal). Contoh abstract method:

```
3   abstract class Phone
4   {
5       protected String phoneNumber;
6
7       abstract void call(String callNumber);
8       abstract void endCall();
9   }
```

Method abstract tanpa implementasi

Ketika sebuah class mewarisi sebuah abstract class, maka class tersebut harus mengimplentasikan ulang semua abstract method yang ada di parent-nya. Jika tidak, maka class tersebut juga harus dideklarasikan sebagai abstract. Perhatikan contoh berikut ini:



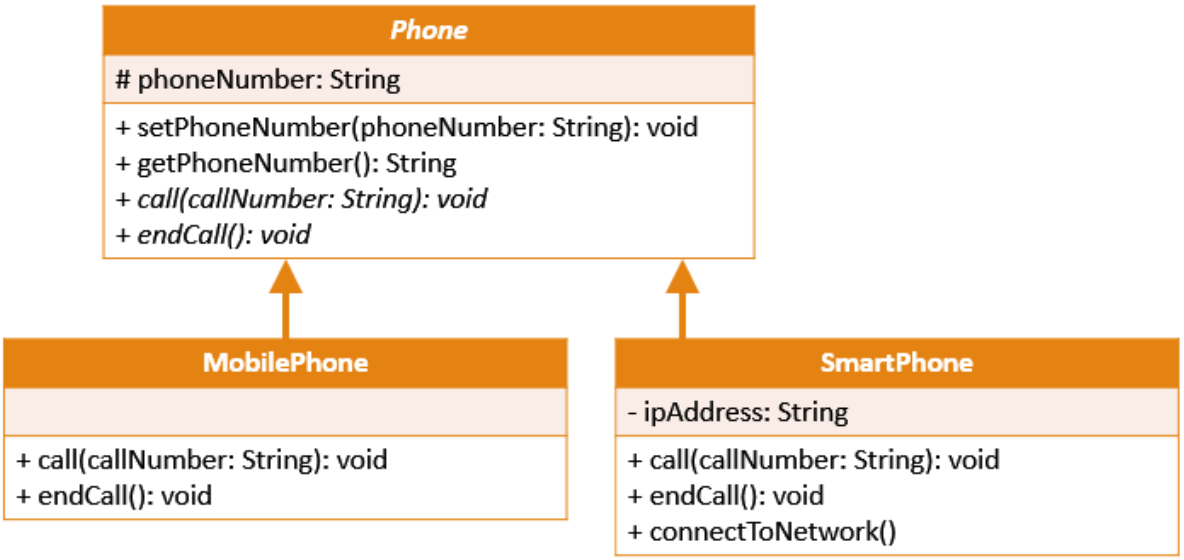
Pada class diagram diatas, class **Phone** adalah abstract class, maka nama class nya harus ditulis miring. Didalamnya terdapat method *call()* dan *endCall()* yang merupakan abstract method, maka harus ditulis miring juga. Sedangkan method *setPhoneNumber()* bukan abstract method, maka ditulis tegak.

Untuk meng-instansiasi class **Phone** tersebut maka dibuatlah class turunannya yaitu **SmartPhone**, yang didalamnya mengimplementasikan ulang method-method abstract yaitu *call()* dan *endCall()*. Sedangkan method *setPhoneNumber()* bukan method abstract, jadi tidak perlu diimplementasikan ulang.

4. Percobaan 1

4.1 Tahap Percobaan

1. Perhatikan class diagram dibawah ini:



2. Buatlah abstract class Phone seperti dibawah ini:

```
4   abstract class Phone
5   {
6       protected String phoneNumber;
7
8       public void setPhoneNumber(String phoneNumber)
9       {
10          this.phoneNumber = phoneNumber;
11      }
12
13      public String getPhoneNumber()
14      {
15          return phoneNumber;
16      }
17
18      abstract void call(String callNumber);
19      abstract void endCall();
20  }
```

Method abstract tanpa implementasi

3. Kemudian buatlah class MobilePhone yang menurunkan class abstract Phone:

```
3   public class MobilePhone extends Phone
4   {
6       public void call(String callNumber)
7       {
8           System.out.println("MobilePhone melakukan panggilan ke: " + callNumber);
9       }
11
12      public void endCall()
13      {
14          System.out.println("Mobilephone mengakhiri panggilan");
15      }
16  }
```

Implementasi method dari class abstract Phone

4. Kemudian class SmartPhone yang juga menurunkan class abstract Phone:

```
3   public class SmartPhone extends Phone
4   {
5       public String ipAddress;
6
8       public void call(String callNumber)
9       {
10          System.out.println("Smarphone melakukan panggilan ke: " + callNumber);
11      }
13      public void endCall()
14      {
15          System.out.println("Smarphone mengakhiri panggilan");
16      }
17
18      public void connectToNetwork()
19      {
20          System.out.println("Smarphone melakukan koneksi ke jaringan");
21      }
22  }
```

Implementasi method dari class abstract Phone

5. Kemudian class TestPhone:

```
3   public class TestPhone
4   {
5       public static void main(String[] args)
6       {
7           MobilePhone noqia = new MobilePhone();
8           SmartPhone samsunk = new SmartPhone();
9
10          noqia.setPhoneNumber("081234567");
11          samsunk.setPhoneNumber("08567891011");
12
13          noqia.call("0833445566");
14          noqia.endCall();
15
16          checkPhoneNumber(samsunk);
17      }
18
19      public static void checkPhoneNumber(Phone phone)
20      {
21          System.out.println("Nomor dari telepon ini adalah: "
22                  + phone.getPhoneNumber());
23      }
24  }
```

6. Compile class TestPhone, cocokkan output anda:

```
run:
MobilePhone melakukan panggilan ke: 0833445566
Mobilephone mengakhiri panggilan
Nomor dari telepon ini adalah: 08567891011
BUILD SUCCESSFUL (total time: 0 seconds)
```

4.2 Pertanyaan

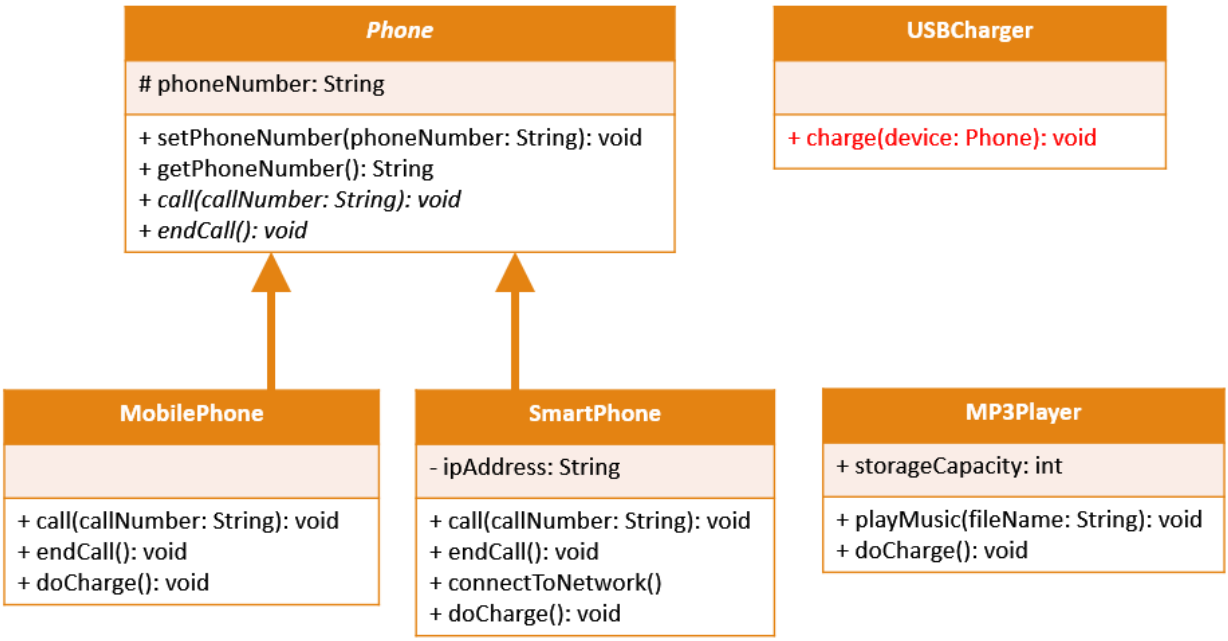
1. Dari class Phone, SmartPhone, MobilePhone, manakah yang merupakan abstract class?
2. Mana sajakah yang merupakan abstract method?
3. Apa perbedaan dari abstract method dan method biasa?
4. Pada TestPhone, terdapat method *checkPhoneNumber()* yang menerima parameter bertipe **Phone** (baris ke 19 pada gambar). Namun pada saat method tersebut dipanggil, object *samsunk* yang notabene bertipe **SmartPhone** bisa dimasukkan sebagai parameter method tersebut (baris ke 16). Mengapa demikian?
5. Cobalah membuat object dari class Phone pada TestPhone, seperti dibawah ini:

```
        Phone sonny = new Phone();
```

6. Percobaan nomor 5 diatas akan menghasilkan error. Mengapa demikian?
7. Coba hapus method *call()* dan *endCall()* pada class **SmartPhone**. Akan terjadi error, mengapa demikian?
8. Apa yang dapat anda simpulkan dari percobaan diatas?

5. Interface

Interface digunakan sebagai jembatan antar object agar bisa saling berkomunikasi tanpa melihat hirarki pewarisan dari object tersebut. Untuk memudahkan memahami interface dan manfaatnya, coba amati class diagram berikut ini:



Pada class diagram diatas, dapat kita lihat bahwa class **MP3Player** tidak ada hubungan hirarki dengan class **Phone, MobilePhone** maupun **SmartPhone**. Akibatnya, class **MP3Player** tidak dapat dimasukkan sebagai parameter di method *charge()* yang ada di class **USBCharger**. Sedangkan class **MobilePhone** dan **SmartPhone** dapat dimasukkan ke method *charge()*, karena merupakan turunan dari class **Phone**. Padahal harapannya, semua class bisa dimasukkan ke method *charge()*. Bagaimana solusinya? Kita bisa gunakan **interface**.

Cara membuat interface adalah dengan menggunakan keyword **interface**:

```
3    public interface ChargeAble
4    {
5
6    }
```

Berbeda dengan abstract class, pada interface tidak boleh ada method yang ada implementasinya. Method yang ada didalam interface hanya deklarasi saja (diakhiri dengan titik koma, tanpa ada kurung kurawal). Seperti contoh berikut ini:

```
3    public interface ChargeAble
4    {
5        public void doCharge();
6    }
```

Deklarasi method tanpa implementasi

Interface dapat diimplementasikan oleh class lain. Caranya adalah dengan menggunakan keyword **implements**. Sebagai contoh, class **MP3Player** yang mengimplementasikan interface **ChargeAble:**

```
3   public class MP3Player implements ChargeAble
4   {
5
6   }
```

Semua method yang ada pada interface, **harus** diimplementasikan ulang pada class yang mengimplementasikannya. Misal pada interface **ChargeAble** terdapat method *doCharge()*, maka class **MP3Player** yang notabene meng-implements interface **ChargeAble**, harus mengimplementasikan ulang method *doCharge()* tersebut. Contoh:

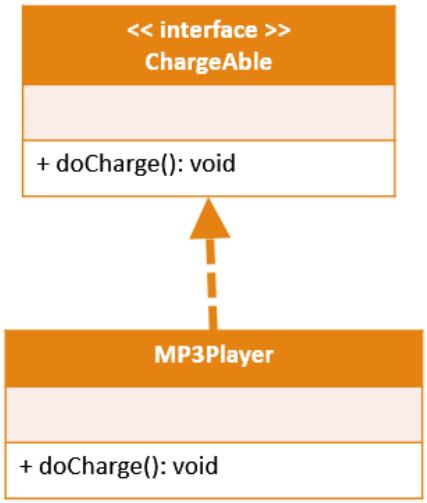
```
4   public interface ChargeAble
4   {
6       public void doCharge();
6   }
```

Deklarasi method tanpa implementasi

```
3   public class MP3Player implements ChargeAble
4   {
6       public void doCharge()
6       {
7           System.out.println("MP3Player is charging.");
8       }
9   }
```

Implementasi method dari interface ChargeAble

Adapun class diagram dari contoh diatas adalah sebagai berikut:



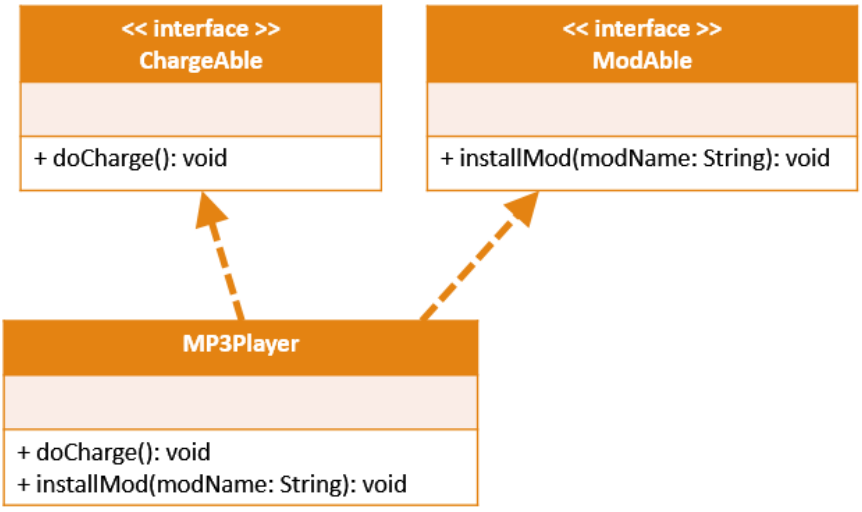
Pada interface, ditambahkan tanda **<< interface >>** diatas nama interface-nya. Kemudian class yang meng-implements interface tersebut diberi tanda panah dengan garis putus-putus yang mengarah ke interface.

6. Multiple Interface

Kelebihan dari interface adalah satu class dapat meng-implements lebih dari satu interface. Oleh karena itu interface dikatakan sebagai solusi jika kita ingin menerapkan multiple inheritance. Penulisan implements yang lebih dari satu interface dipisahkan dengan koma (,)

Misal, class **MP3Player** yang meng-implement **ChargeAble** dan **ModAble**:

```
3    public class MP3Player implements ChargeAble, ModAble
4    {
5        // ...
6    }
```



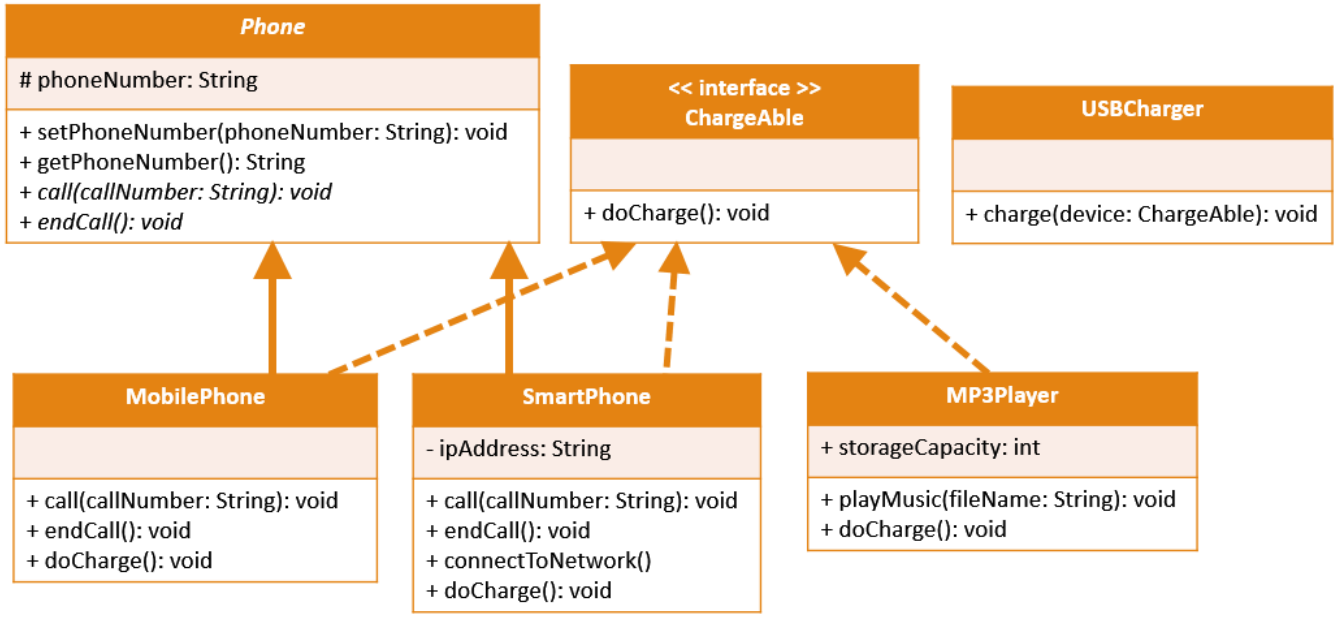
Maka class **MP3Player** harus mengimplementasikan semua method yang ada di **ChargeAble** dan **ModAble**.

Dengan demikian, kita bisa menerapkan interface pada contoh kasus ini. Untuk lebih lebih lengkapnya ikuti percobaan berikut ini.

7. Percobaan 2

7.1 Langkah Percobaan

1. Amati class diagram dibawah ini



Dapat kita amati pada class diagram diatas, class **MobilePhone, SmartPhone** dan **MP3Player**, semuanya mengimplements interface **ChargeAble**. Class-class tersebut juga mengimplentasikan ulang method *doCharge()*. Kemudian pada class **USBCharger**, method *charge()* sekarang menerima parameter yang bertipe **ChargeAble**, dengan demikian semua class yang meng-implements **ChargeAble** dapat di-charge oleh **USBCharger**.

2. Buatlah interface **ChargeAble** seperti berikut ini:

```
4   public interface ChargeAble
5   {
6       public void doCharge();
7   }
```

3. Kemudian buat class **USBCharger** seperti berikut ini:

```
3   public class USBCharger
4   {
5       public void charge(ChargeAble device)
6       {
7           device.doCharge();
8       }
9   }
```

4. Buatlah class **MP3Player** yang meng-implements interface **ChargeAble** seperti berikut ini:

```java
public class MP3Player implements ChargeAble
{
    public int storageCapacity;

    public void playMusic(String fileName)
    {
        System.out.println("Playing music: " + fileName);
    }

    public void doCharge()
    {
        System.out.println("MP3Player is charging.");
    }
}
```

5. Kemudian ubah class **MobilePhone** dan **SmartPhone** tambahkan implements ChargeAble. dan method *doCharge()* seperti berikut ini:

```java
public class SmartPhone extends Phone implements ChargeAble
{
    public String ipAddress;

    public void call(String callNumber)
    {
        System.out.println("Smarphone melakukan panggilan ke: " + callNumber);
    }

    public void endCall()
    {
        System.out.println("Smarphone mengakhiri panggilan");
    }

    public void connectToNetwork()
    {
        System.out.println("Smarphone melakukan koneksi ke jaringan");
    }

    public void doCharge()
    {
        System.out.println("Smartphone is charging.");
    }
}
```

```
3   public class MobilePhone extends Phone implements ChargeAble
4   {
5       public void call(String callNumber)
6       {
7           System.out.println("MobilePhone melakukan panggilan ke: " + callNumber);
8       }
9
10      public void endCall()
11      {
12          System.out.println("Mobilephone mengakhiri panggilan");
13      }
14
15      public void doCharge()
16      {
17          System.out.println("MobilePhone is charging.");
18      }
19  }
```

6. Kemudian TestPhone:

```
3   public class TestPhone
4   {
5       public static void main(String[] args)
6       {
7           MobilePhone noqia = new MobilePhone();
8           SmartPhone samsunk = new SmartPhone();
9           MP3Player iphod = new MP3Player();
10          USBCharger charger = new USBCharger();
11
12          charger.charge(noqia);
13          charger.charge(samsunk);
14          charger.charge(iphod);
15      }
16  }
```

7. Cocokkan output anda:

```
run:
MobilePhone is charging.
Smartphone is charging.
MP3Player is charging.
BUILD SUCCESSFUL (total time: 0 seconds)
```

7.2 Pertanyaan

1. Apa yang dimaksud dengan interface?
2. Apa perbedaan antara interface dengan abstract class?
3. Dari class-class diatas, yang manakah yang merupakan interface?
4. Apa maksud dari keyword **implements**?
5. Coba tambahkan method *checkVoltage()* pada **ChargeAble** seperti berikut ini:

```
7       public void checkVoltage()
        {
9           System.out.println("5 volt");
10      }
11  }
```

Compile program anda. Apa yang terjadi? Jelaskan alasannya.

6. Perhatikan pada class **USBCharger**, method *charge()* menerima parameter bertipe **ChargeAble**. Namun pada **TestPhone**, kita memanggil method *charge()* tersebut dengan memasukkan parameter object **noqia:**

```
charger.charge(noqia);
```

Namun hal ini dapat dilakukan, padahal notabene object **noqia** bertipe **MobilePhone**. Mengapa demikian?

7. Mengapa terdapat method *doCharge()* pada class **SmartPhone**?
8. Apabila method *doCharge()* pada class **SmartPhone** dihapus, apa yang terjadi? Jelaskan alasannya.
9. Tambahkan method *stopCharge()* pada interface **ChargeAble**:

```
    public interface ChargeAble
4   {
        public void doCharge();
        public void stopCharge();
7
8   }
```

Kemudian tambahkan method *stop()* pada class **USBCharger:**

```
3   public class USBCharger
4   {
5       public void charge(ChargeAble device)
6       {
7           device.doCharge();
8       }
9
10      public void stop(ChargeAble device)
11      {
12          device.stopCharge();
13      }
14  }
```

Lakukan perubahan seperlunya pada class-class lain yang sekiranya perlu dilakukan perubahan. Kemudian uji dengan **TestPhone** berikut ini:

```java
public class TestPhone
{
    public static void main(String[] args)
    {
        MobilePhone noqia = new MobilePhone();
        SmartPhone samsunk = new SmartPhone();
        MP3Player iphod = new MP3Player();
        USBCharger charger = new USBCharger();

        charger.charge(noqia);
        charger.charge(iphod);

        charger.stop(noqia);
        charger.stop(iphod);
    }
}
```

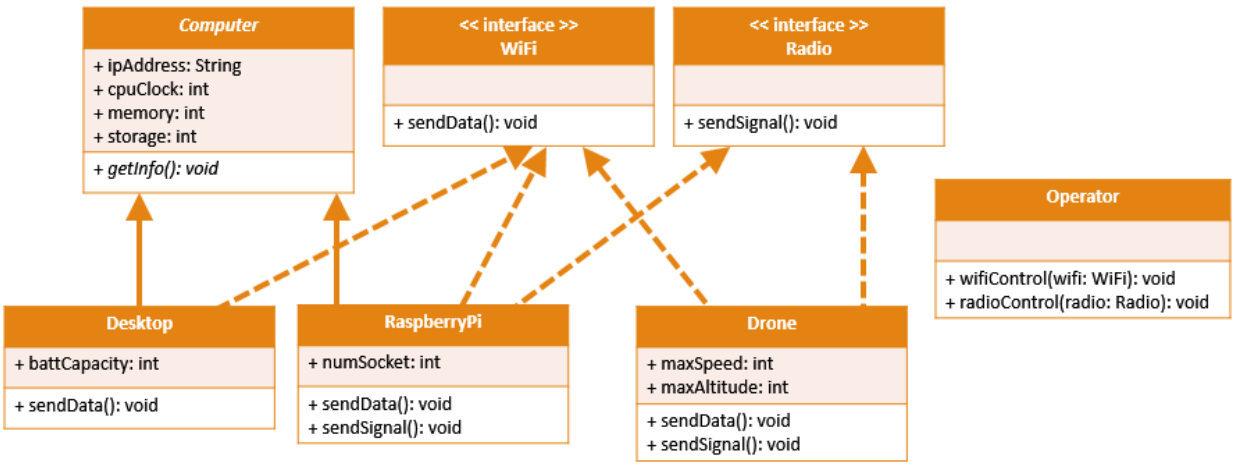
Output yang diharapkan:

```
run:
MobilePhone is charging.
MP3Player is charging.
MobilePhone has stopped charging.
MP3Player has stopped charging.
BUILD SUCCESSFUL (total time: 0 seconds)
```

10. Apa yang dapat anda simpulkan dari percobaan ini?

8. Tugas

1. Perhatikan class diagram dibawah ini:



Buatlah kode program dari class diagram diatas. Perhatikan bahwa class **RaspberryPi** dan **Drone** meng-implement lebih dari satu interface (Multiple Interface). Uji dengan **TestSignal** berikut ini dan cocokkan hasilnya:

```
public class TestSignal
{
    public static void main(String[] args)
    {
        Dekstop aceer = new Dekstop();
        RaspberryPi rp = new RaspberryPi();
        Drone dji = new Drone();
        Operator op = new Operator();

        op.wifiControl(aceer);
        op.wifiControl(rp);
        op.wifiControl(dji);
        op.radioControl(rp);
        op.radioControl(dji);
    }
}
```

Hasil yang diharapkan:

```
Controlling desktop via wifi...
Controlling raspberry via wifi...
Controlling drone via wifi...
Controlling raspberry via radio...
controlling drone via radio...
```

2. Sebuah game action adventure, didalamnya terdapat karakter, kendaraan, bangunan dan monster. Semuanya (karakter, kendaraan dan bangunan) dapat dihancurkan oleh monster. Desainlah programnya, dengan menerapkan konsep abstract class dan interface. Anda bebas menentukan atribut dan method pada masing-masing class dan interface nya dan juga test class nya. Buat class diagramnya dan kode programnya.