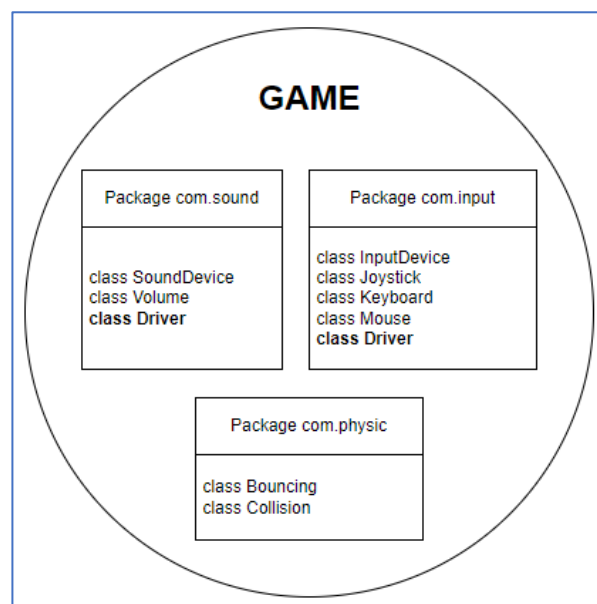


**MATA KULIAH** : PEMROGRAMAN BERORIENTASI OBJEK  
**SESI PERTEMUAN** : VI (ENAM)  
**MATERI** : PACKAGE, ABSTRACT & INTERFACE  
**DOSEN** : ALUN SUJJADA, S.KOM., M.T

### 6.1 Package

Pada pembuatan program menggunakan JAVA, disediakan sebuah fasilitas atau fitur yang sangat berguna sekali terlebih jika program yang akan dibuat dirasa kompleks dan melibatkan banyak file yang terdiri dari *class* atau *interface*. Salah satu fitur tersebut adalah *package* atau paket yaitu sebuah folder yang digunakan sebagai dasar meletakkan file *class* atau *interface*. *Package* berfungsi untuk mengelompokkan beberapa *class* atau *interface* dalam satu unit yang mempunyai kesamaan kategori. Selain itu kegunaan *package* adalah untuk melakukan manajemen *file* dalam jumlah banyak untuk menghindari konflik pada penamaan. Gambar 6.1 berikut ini adalah ilustrasi tentang *package*



Gambar 6.1. Ilustrasi *package*

Misal ketika ingin membuat *game* maka file *class* atau *interface* diorganisir kedalam *package* agar mudah dalam memanajemen *file* serta menghindari konflik akibat kesamaan nama *file*. Seperti contoh pada gambar 6.1, dalam sebuah *game* masing-masing *class* yang mempunyai kesamaan kategori dikelompokkan dalam masing-masing *package* yaitu :

1. *Package* **com.sound** untuk *class* yang bertugas menangani interaksi *user* dengan *sound hardware*.
2. *Package* **com.input** untuk *class* yang bertugas menangani *input* sewaktu menjalankan *game*. *User* dapat memilih mau menggunakan *joystick*, *keyboard* atau *mouse*.
3. *Package* **com.physic** digunakan untuk pergerakan benda yang berhubungan dengan rumus-rumus fisika seperti memantul atau *bouncing*, *collision detection* yaitu tumbukan antar objek.

Ketika *user* akan berinteraksi dengan *sound* maka diperlukan sebuah *class* **Driver** sebagai penghubungnya, begitu juga ketika akan berinteraksi dengan *input*. Terdapat 2 *class* dengan nama yang sama yaitu *class* **Driver**. Ketika hal tersebut sudah menggunakan organisasi *file* menggunakan *package* maka hal tersebut dapat berjalan sesuai dengan alur dan fungsinya, tetapi ketika tidak menggunakan *package* maka ketika *class* tersebut dipanggil menggunakan *keyword* **import** akan terjadi *error*.

## 6.2 Membuat *Package* dari Awal

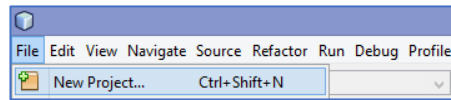
Pembuatan *package* menggunakan *keyword* *package* diikuti dengan nama *package* yang dituliskan pada bagian paling atas dari sebuah *file*. Secara konvensi atau aturan kesepakatan bersama bahwa pembuatan *package* diawali dengan *keyword* *com*, tetapi hal tersebut bukanlah menjadi sebuah kewajiban atau keharusan. Kode 6.1 berikut ini adalah contoh kode pembuatan *package*.

```
package com.Lingkaran
```

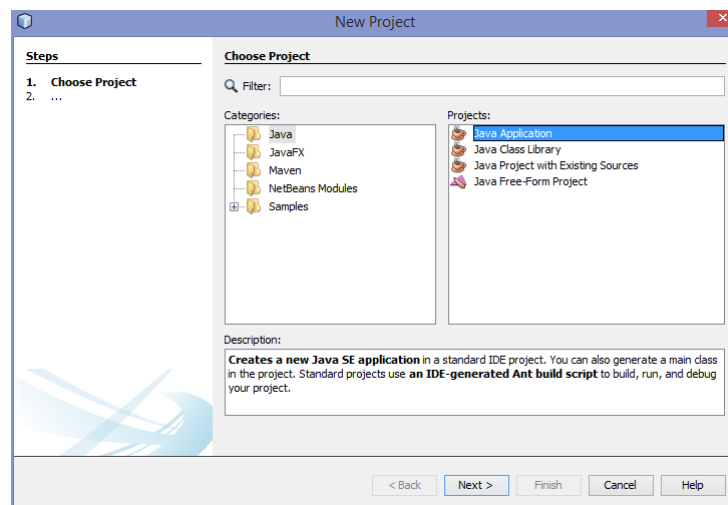
Kode 6.1 Pembuatan *package*

Berikut ini adalah tahapan pembuatan *package* dari awal menggunakan **IDE Netbeans** yaitu:

1. Buat *project* dengan memilih menu **File** → **New Project** seperti pada gambar 6.2 berikut ini:

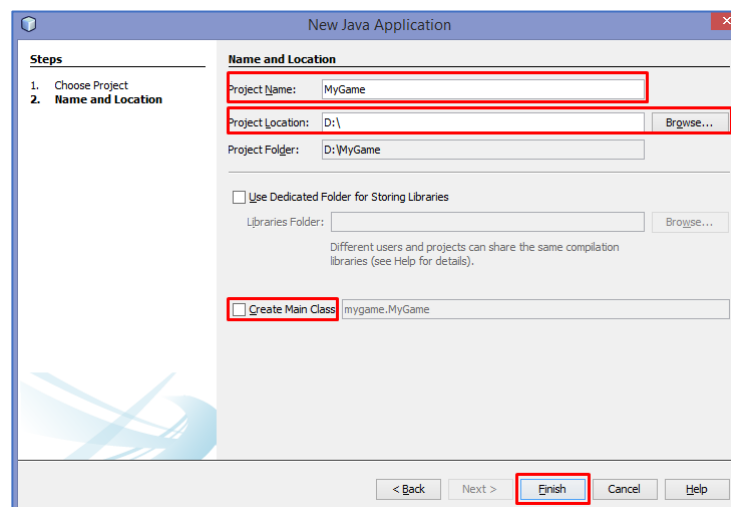
Gambar 6.2. Pembuatan *project*

- Setelah di klik maka akan tampil *window box* dan isilah **Categories** dengan Java dan **Projects** dengan Java Application seperti pada gambar 6.3 berikut ini:



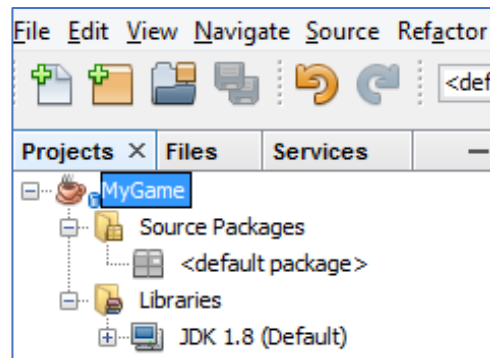
Gambar 6.3 Pemilihan Categories dan Projects

- Klik tombol **Next >** maka akan muncul pengisian parameter dari *project* yang akan dibuat seperti nama *project* dan lokasi *path* penyimpanan *file* seperti pada gambar 6.4 berikut ini:

Gambar 6.4 Pengisian parameter *project*

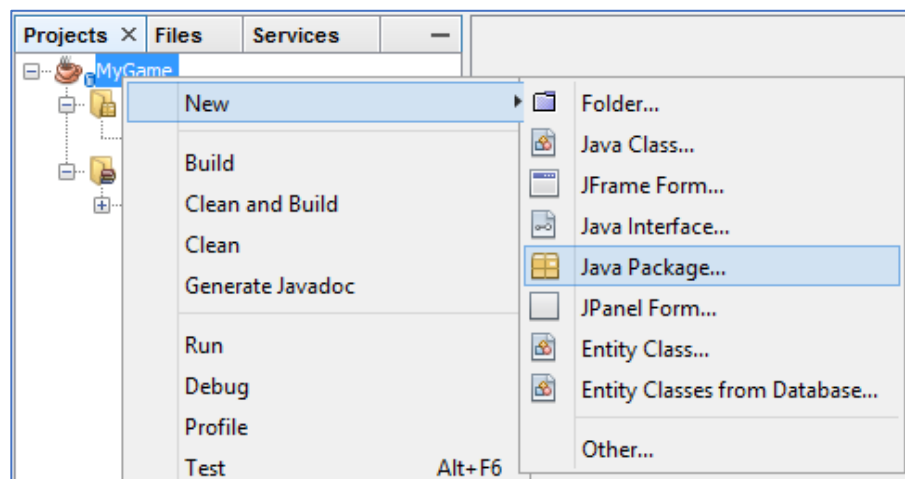
Isikan **Project Name** atau nama *project* , **Project Location** atau lokasi *path* penyimpanan *file* dan hilangkan centang (*uncheck*) untuk **Create Main Class**. Lanjutkan dengan klik tombol **Finish**

4. Pada tampilan awal program akan muncul hirarki **Projects** seperti pada gambar 6.5 berikut ini:



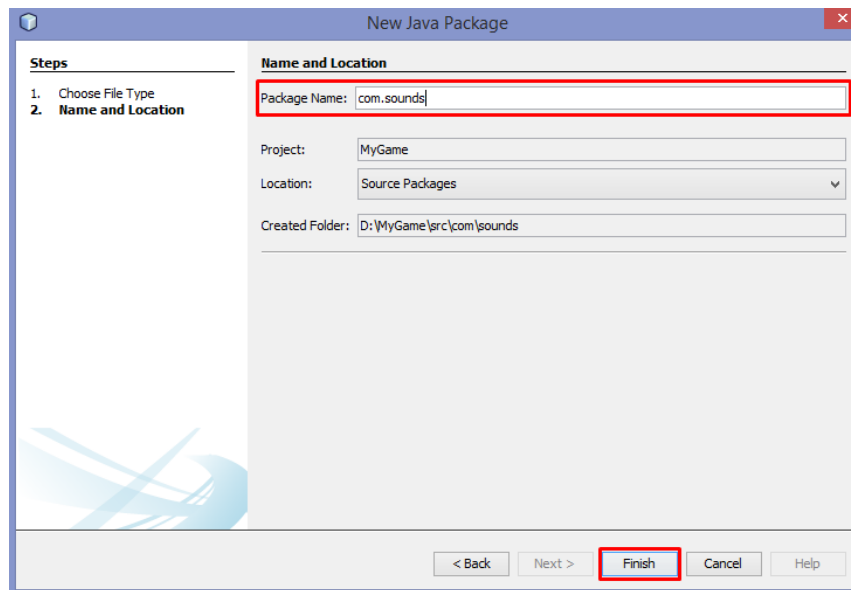
Gambar 6.5 Hirarki Projects

5. Klik kanan pada **Projects MyGame** kemudian pilih menu **New → Java Package** seperti pada gambar 6.6 berikut ini:



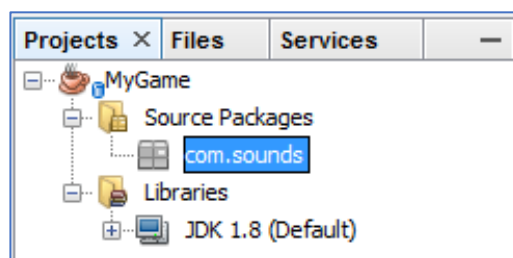
Gambar 6.6 Pemilihan menu Java Package

6. Setelah itu isikan nama *package* pada *form* isian berikutnya dan klik tombol **Finish** seperti pada gambar 6.7 berikut ini:



Gambar 6.7 Pengisian nama *package*

7. Jika berhasil maka *project* yang dibuat akan memiliki hirarki **Project** seperti pada gambar 6.8 berikut ini:

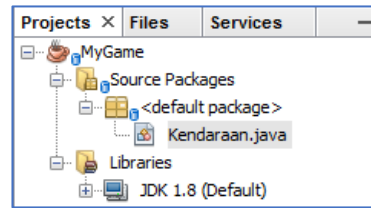


Gambar 6.8 Tampilan Hirarki Project dengan *package*

8. Ulangi langkah-langkah tersebut dari awal, jika ingin membuat *package* baru.

### 6.3 Membuat *Package* di Akhir

Langkah pembuatan *package* pada sub bab 6.2 tersebut adalah langkah yang dikerjakan dari awal jika perencanaan sudah bagus. Akan tetapi kadang kala terjadi kode program sudah dibuat terlebih dahulu kemudian baru *file class* ingin dikelompokkan dalam sebuah *package*. Misalkan terdapat file *Kendaraan.java* yang berada pada *source package* seperti pada gambar 6.9 berikut ini:



Gambar 6.8 Class Kendaraan di source package

Misal file **Kendaraan.java** tersebut ingin diubah kedalam *package* `com.alat.transportasi`, maka tinggal tambahkan ***package com.alat.transportasi*** dibagian awal file seperti pada kode 6.2 berikut ini:

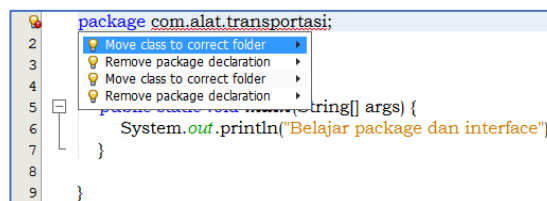
```

1. package com.alat.transportasi;
2.
3. public class Kendaraan {
4.
5.     public static void main(String[] args) {
6.         System.out.println("Belajar package dan interface");
7.     }
8.
9. }

```

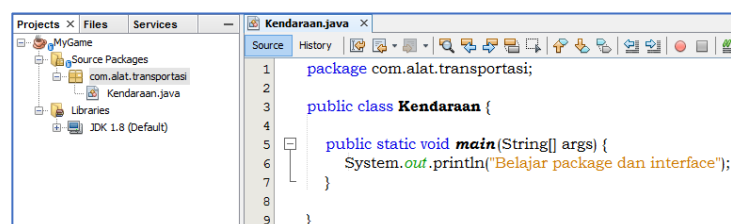
Kode 6.2 File Kendaraan.java

Karena letak file tersebut tidak sesuai dengan folder *package* maka akan muncul saran berupa notifikasi dengan gambar *yellow bulb*, lakukan klik dan pilih menu **move class to correct folder** seperti pada gambar 6.10 berikut ini:



Gambar 6.9 Menu saran dari Netbeans

Hasil dari pemilihan menu tersebut maka Netbeans otomatis memindahkan file `Kendaraan.java` dari *source package* ke folder *package com.alat.transportasi* yang sudah dibuat secara otomatis seperti pada gambar 6.11 berikut ini:

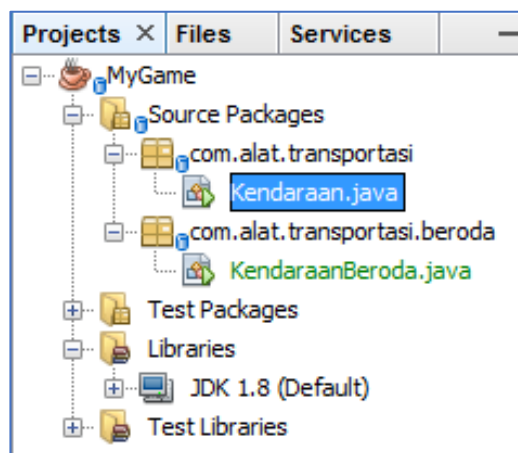


Gambar 6.10 Hasil pemindahan file Kendaraan.java

## 6.4 Mengimport Packages

*File class* yang berada dalam sebuah *package* dapat digunakan oleh *class* lain yang berada diluar *package* dengan melakukan cara *import package*. Secara aturan maka *import* dituliskan dibawah *package*. Simaklah gambar 6.12 berikut ini agar lebih jelas dan mengerti. Misal terdapat 2 *package* yaitu

1. **com.alat.transportasi** yang mempunyai 1 (satu) *class* *Kendaraan*.
2. **com.alat.transportasi.beroda** yang mempunyai 1 (satu) *class* *KendaraanBeroda*.



Gambar 6.11 Hierarki 2 (dua) package

Maka untuk menggunakan *file* **KendaraanBeroda** pada *file* **Kendaraan** adalah seperti pada kode 6.3 berikut ini:

```
1. package com.alat.transportasi;
2.
3. import com.alat.transportasi.beroda.*;
4.
5. public class Kendaraan {
6.
7.     public static void main(String[] args) {
8.         System.out.println("Belajar package dan interface");
9.     }
10.
11. }
```

Kode 6.3 Import package

Atau jika ingin menggunakan *file* yang lebih spesifik dapat menggunakan *import* beserta nama *class* yang ingin digunakan, seperti pada kode 6.4 berikut ini:

```
1. package com.alat.transportasi;
2.
3. import com.alat.transportasi.beroda.KendaraanBeroda;
4.
5. public class Kendaraan {
6.
7.     public static void main(String[] args) {
8.         System.out.println("Belajar package dan interface");
9.     }
10. }
```

Kode 6.4 *Import spesifik file*

Sebagai contoh, bila Anda ingin menggunakan class `Color` dalam package `awt`, Anda harus menuliskan import package seperti kode 6.5 berikut ini:

```
1. import java.awt.Color;
2. import java.awt.*;
3. public class DrawingCanvas{
4.
5. }
```

Kode 6.5 Import Java AWT

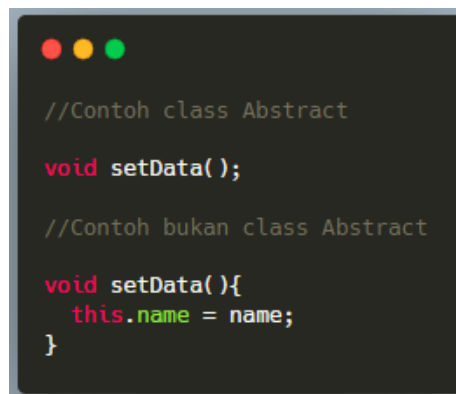
## 8.5 Abstract

Salah satu contoh yang mewakili konsep *abstract* yaitu class “Kendaraan”. Hal tersebut terjadi karena class “Kendaraan” belum dapat menentukan secara spesifik benda yang mewakilinya. Kendaraan bisa saja adalah sebuah motor, pesawat atau mobil dan lain sebagainya. Tetapi semuanya itu termasuk dalam kategori “Kendaraan”. Setiap “Kendaraan” pasti bisa berjalan, namun ketika disebut “Kendaraan Berjalan” akan muncul pertanyaan “Bagaimana cara jalannya?”, “Terbang atau jalan di darat atau laut?”. Tetapi jika disebut “Mobil berjalan”, maka setiap orang pasti sudah tau maksudnya. Mobil adalah benda yang konkrit sedangkan “Kendaraan” adalah benda yang masih abstrak.

### 8.5 Class Abstract

Class abstrak adalah class yang masih dalam bentuk abstract, karena bentuknya masih abstrak, maka class tersebut tidak bisa dibuat langsung menjadi objek. Sebuah class agar dapat disebut class abstract setidaknya memiliki satu atau lebih method abstract. Method abstract adalah method yang tidak memiliki implementasi atau tidak ada bentuk konkritnya. Gambar 6.13 berikut ini adalah contoh dari method abstract dan bukan abstract





```
//Contoh class Abstract  
  
void setData();  
  
//Contoh bukan class Abstract  
  
void setData(){  
    this.name = name;  
}
```

Gambar 6.13 Contoh method abstract dan non abstract

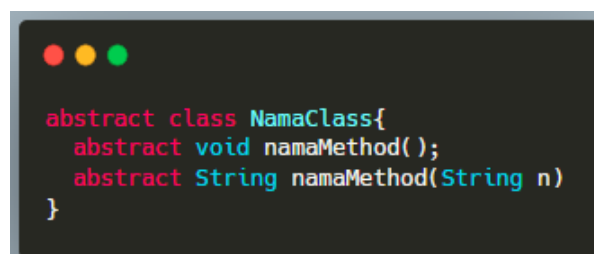
### 8.5 Penggunaan Class Abstract

Class abstract memang belum dapat digunakan secara langsung, sehingga untuk menggunakannya maka harus dibuat terlebih dahulu bentuk konkritnya. Cara untuk membuat class abstract menjadi bentuk konkrit adalah dengan mengimplementasikan method yang ada didalamnya. Hal tersebut dapat dilakukan dengan menggunakan cara pewarisan (*inheritance*). Class Abstract dibuat sebagai parent class dari class yang lain. Child class membuat bentuk konkrit dari class abstract.

Mengapa dibutuhkan sebuah class abstract, padahal hal tersebut dapat dilakukan menggunakan class biasa saja?. Pernyataan tersebut adalah benar, akan tetapi pada sebuah kondisi tertentu parent class tidak ingin dibuat sebagai *object*, karena methodnya belum jelas akan diimplementasikan seperti apa? maka sebaiknya *class* tersebut dibuat menjadi *abstract*.

### 8.6 Cara Membuat Class Abstract

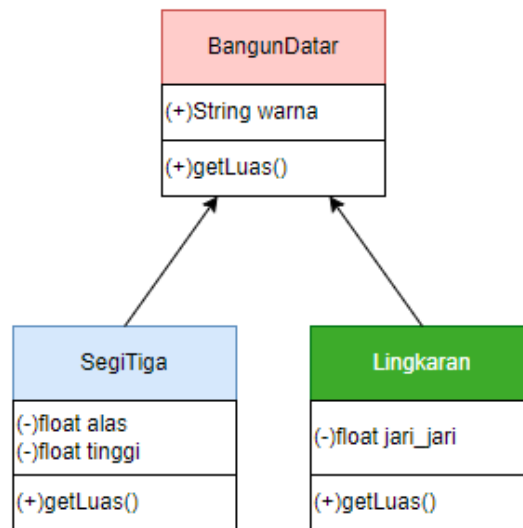
Cara membuat class abstract adalah menambahkan kata kunci *abstract* pada class dan method. Gambar 6.14 berikut ini adalah contoh dari class abstract:



```
abstract class NamaClass{  
    abstract void namaMethod();  
    abstract String namaMethod(String n)  
}
```

Gambar 6.14 Membuat class abstract

Berikut ini adalah contoh program dari class abstract



Gambar 6.15 Diagram Class

Class **BangunDatar** adalah class abstrak, karena mempunyai method abstrak `getLuas()`. Jika akan membuat objek dengan class **BangunDatar** dan memanggil method `getLuas()`, maka class **BangunDatar** akan bingung. **BangunDatar** atau bentuk bangun datar yang mau dihitung luasnya seperti apa? bagaimana rumusnya? oleh karena itu, class ini dijadikan sebagai abstrak. Sedangkan seperti yang diketahui secara umum bahwa setiap bangun datar pasti mempunyai luas. Kode 6.6 berikut ini adalah contoh implementasi penyelesaian masalah bangun datar.

```
1. public abstract class BangunDatar {
2.     String warna;
3.
4.     String getWarna() {
5.         return warna;
6.     }
7.
8.     void setWarna(String warna) {
9.         this.warna = warna;
10.    }
11.
12.    abstract float getLuas();
13. }
```

Kode 6.6 Class BangunDatar

Kemudian dibuat class SegiTiga dan Lingkaran seperti kode program 6.7 dan 6.8 berikut ini:

```
1. public class SegiTiga extends BangunDatar {
2.
3.     private float alas;
4.     private float tinggi;
5.
6.     public SegiTiga(float alas, float tinggi) {
7.         this.alas = alas;
8.         this.tinggi = tinggi;
9.     }
10.
11.     @Override
12.     float getLuas() {
13.         return (float)0.5 * alas * tinggi;
14.     }
15. }
```

Kode 6.7 Class SegiTiga

```
1. public class Lingkaran extends BangunDatar {
2.
3.     float jari_jari;
4.
5.     public Lingkaran(float jari_jari) {
6.         this.jari_jari = jari_jari;
7.     }
8.
9.     @Override
10.    float getLuas() {
11.        return (float) Math.PI * jari_jari * jari_jari;
12.    }
13. }
```

Kode 6.8 Class Lingkaran

Proses terakhir adalah membuat class Main untuk mengimplementasikan pembuatan objek dengan class yang telah dibuat sebelumnya.

```
1. public class Main {
2.
3.     public static void main(String[] args) {
4.
5.         BangunDatar segitiga = new SegiTiga(12, 20);
6.         BangunDatar lingkaran = new Lingkaran(60);
7.
8.         System.out.println("Luas dari bangun datar segitiga : " + segitiga.getLuas());
9.         System.out.println("Luas dari bangun datar lingkaran : " + lingkaran.getLuas());
10.    }
11. }
```

Kode 6.9 Class Main

Pada class Main terdiri dari objek yaitu objek segitiga dan lingkaran. Kedua objek ini memiliki tipe yang sama yaitu *BangunDatar*, tetapi dibuat dari class yang berbeda.

### 8.7 Interface

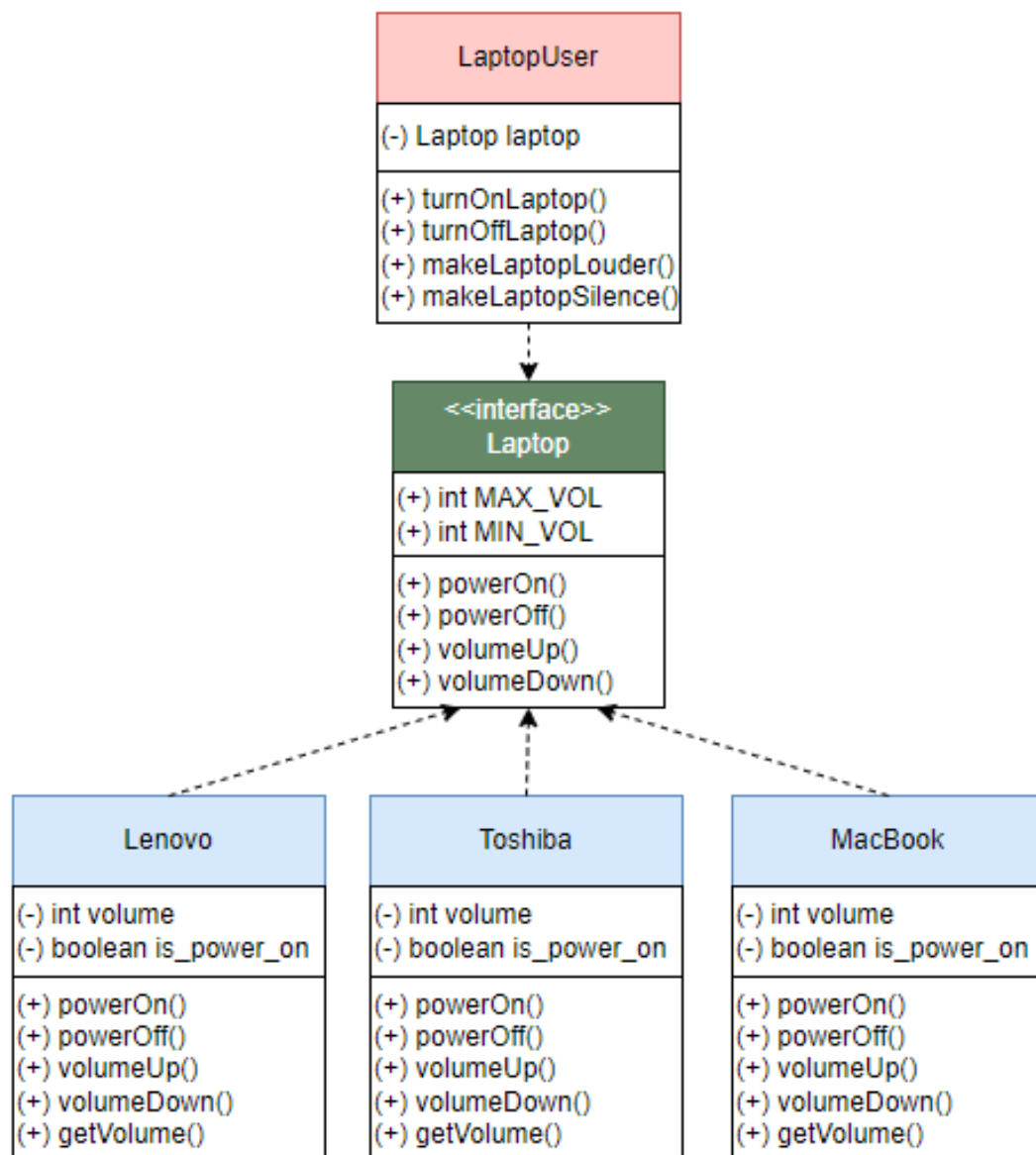
*Interface* adalah sebuah cara dari JAVA untuk mengakomodir konsep *multiple inheritance* yang dianut oleh bahasa pemrograman lain seperti C, C++ dan lain sebagainya. Interface adalah *blue print* dari sebuah *class* yang berupa *method* kosong atau nama *method* tanpa implementasi program. Implementasi program dituliskan pada *class* yang menggunakan *interface*. Berdasarkan definisi *interface* adalah antarmuka, seperti yang sudah sering terdengar misalnya GUI (*Graphical User Interface*). *Interface* pada JAVA lebih tepatnya diartikan sebagai “penghubung” antara sesuatu yang masih bersifat abstrak dengan sesuatu yang telah memiliki nilai nyata. Perhatikan gambar 6.15 berikut ini:



Gambar 6.15 Ilustrasi Interface

Laptop pada gambar 6.15 mempunyai tombol-tombol keyboard. Tombol itulah yang dimaksud dengan *interface*, sedangkan *user* tidak akan tahu seberapa kompleks dan abstrak tampilan sistem yang ada pada layar monitor. Saat tombol *keyboard* tersebut ditekan, maka hasil yang didapatkan dari masing-masing merk akan berbeda. Misal ketika ditekan tombol F1 pada laptop merk A akan menonaktifkan suara (*volume*) sedangkan pada laptop merk B akan menambah

tingkat kecerahan. Hal ini bergantung pada implementasi yang diberikan pada tiap-tiap tombol tersebut. Beda merk akan beda pula cara memperlakukannya. Sebagai contoh digunakan 2 objek yaitu Laptop dan User sebagai pengguna laptop seperti contoh *class diagram* pada gambar 6.16



Gambar 6.16 Class Diagram Laptop

Kode program 6.10, 6.11, 6.12, 6.13 berikut ini adalah implementasi dari class diagram

```
1. public interface Laptop {  
2.  
3.     int MAX_VOL=100;  
4.     int MIN_VOL=0;  
5.  
6.     void powerOn();  
7.     void powerOff();  
8.     void volumeUp();  
9.     void volumeDown();  
10. }
```

Kode 6.10 Interface Laptop

```
1. public class LaptopUser {  
2.  
3.     private Laptop laptop;  
4.  
5.     public LaptopUser(Laptop laptop) {  
6.         this.laptop = laptop;  
7.     }  
8.  
9.     void turnOnLaptop(){  
10.         this.laptop.powerOn();  
11.     }  
12.  
13.     void turnOffLaptop(){  
14.         this.laptop.powerOff();  
15.     }  
16.  
17.     void makeLaptopLouder(){  
18.         this.laptop.volumeUp();  
19.     }  
20.  
21.     void makeLaptopSilence(){  
22.         this.laptop.volumeDown();  
23.     }  
24. }
```

Kode 6.11 Class LaptopUser

Kemudian dibuat implementasi dari interface Laptop yaitu class Lenovo seperti pada kode 6.12 berikut ini:

```
1. public class Lenovo implements Laptop {
2.
3.     private int volume;
4.     boolean is_power_on;
5.
6.     public Lenovo(){
7.         this.volume = 50;
8.     }
9.
10.    @Override
11.    public void powerOn() {
12.        is_power_on = true;
13.        System.out.println("Laptop is On");
14.        System.out.println("Lenovo ThinkPad");
15.    }
16.
17.    @Override
18.    public void powerOff() {
19.        is_power_on = false;
20.        System.out.println("Shutdown is process ...");
21.    }
22.
23.
24.    @Override
25.    public void volumeUp() {
26.        if(is_power_on){
27.            if(this.volume==MAX_VOL){
28.                System.out.println("Volume is Full ");
29.            }
30.            else{
31.                this.volume += 10;
32.                System.out.println("Volume is :" + this.getVolume());
33.            }
34.        }
35.    }
36.
37.    @Override
38.    public void volumeDown() {
39.        if(is_power_on){
40.            if(this.volume==MIN_VOL){
41.                System.out.println("Volume is 0% ");
42.            }
43.            else{
44.                this.volume -= 10;
45.                System.out.println("Volume is :" + this.getVolume());
46.            }
47.        }
48.    }
49.
50.    public int getVolume(){
51.        return this.volume;
52.    }
53. }
```

Kode 6.22 Class Lenovo

Kemudian dibuat class Main untuk mengimplementasikan pembuatan objek seperti pada kode 6.13 berikut ini:

```
1. import java.util.Scanner;
2.
3. public class Main {
4.
5.     public static void main(String[] args) {
6.         Laptop thinkpad = new Lenovo();
7.         LaptopUser andri = new LaptopUser(thinkpad);
8.
9.         andri.turnOnLaptop();
10.        andri.makeLaptopLouder();
11.        andri.makeLaptopLouder();
12.        andri.makeLaptopSilence();
13.        andri.turnOffLaptop();
14.    }
15. }
```

Kode 6.33 Class Main

## 6.8 Perbedaan Abstract dan Interface

Class abstract dan interface sama-sama digunakan untuk membuat abstraksi. Keduanya hampir memiliki sifat yang sama, namun demikian terdapat beberapa perbedaan yaitu :

1. Pada class abstract dapat membuat properti atau variabel sedangkan pada interface hanya dapat membuat konstanta saja
2. Pada class abstract dapat mengimplementasikan kode method seperti class biasa, sedangkan pada interface harus menggunakan default.
3. Pada class abstract dapat memiliki member private dan protected, sedangkan interface harus public semua
4. Class abstract diimplementasikan dengan pewarisan (extends) sedangkan interface menggunakan implements