Praktikum 6

Class Design (Encapsulation)

Tujuan

Mengetahui bagaimana cara mendeklarasikan suatu class beserta atribut dan metodenya, serta mengakses anggota dari suatu obyek.

Dasar Teori

Deklarasi class dapat dilakukan dengan sintaks sebagai berikut:

```
<modifier> class <nama_class> {
       [deklarasi_atribut]
       [deklarasi_konstruktor]
       [deklarasi_metode]
}
```

Contoh:

```
public class Siswa {
       …
}
```

Deklarasi atribut dapat dilakukan dengan sintaks sebagai berikut:

```
<modifier> <tipe> <nama_atribut> ;
```

Contoh:

```
public class Siswa {
       public int nrp;
       public String nama;
}
```

Deklarasi metode dapat dilakukan dengan sintaks sebagai berikut:

```
<modifier> <return_type> <nama_metode> ([daftar_argumen])
{
       [<statement>]
}
```

Contoh:

```
public class Siswa {
       public int nrp;
       public String nama;

       public void info() {
              System.out.println("Ini siswa PENS");
       }
}
```

Untuk dapat mengakses anggota-anggota dari suatu obyek, maka harus dibuat instance dari class tersebut terlebih dahulu. Berikut ini adalah contoh pengaksesan anggota-anggota dari class Siswa:

```
public class Siswa {
       public static void main(String args[]) {
              Siswa tekkom=new Siswa();
              tekkom.nrp=5;
              tekkom.nama=”Anis”;
              tekkom.info();
       }
}
```

Kita dapat menyembunyikan information dari suatu class sehingga anggota-anggota class tersebut tidak dapat diakses dari luar. Adapun caranya adalah cukup dengan memberikan akses kontrol private ketika mendeklarasikan suatu atribut atau method.

Contoh:

```
private int nrp;
```

Encapsulation (Enkapsulasi) adalah suatu cara untuk menyembunyikan implementasi detail dari suatu class. Enkapsulasi mempunyai dua hal mendasar, yaitu:

- information hiding
- menyediakan suatu perantara (method) untuk pengaksesan data

Contoh:

```
public class Siswa {
    private int nrp;
    public void setNrp(int n) {
        nrp=n;
    }
}
```

Contructor (konstruktor) adalah suatu method yang pertama kali dijalankan pada saat pembuatan suatu obyek. Konstruktor mempunyai ciri yaitu:

- mempunyai nama yang sama dengan nama class
- tidak mempunyai return type (seperti void, int, double, dan lain-lain)

Contoh:

```
public class Siswa {
    private int nrp;
    private String nama;
    public Siswa(int n, String m) {
        nrp  = nomor;
        nama = nama_lengkap;
    }
}
```

Suatu class dapat mempunyai lebih dari 1 konstruktor dengan syarat daftar parameternya tidak boleh ada yang sama.

Contoh:

```
public class Siswa {
     private int nrp;
     private String nama;
```

```
        public Siswa(String m) {
           nrp=0;
           nama="";
        }

        public Siswa(int n, String m) {
            nrp=n;
            nama=m;
        }
}
```

Package adalah suatu cara untuk memenej class-class yang kita buat. Package akan sangat bermanfaat jika class-class yang kita buat sangat banyak sehingga perlu dikelompokkan berdasarkan kategori tertentu. Yang perlu kita perhatikan pada saat deklarasikan package, bahwa class tersebut harus disimpan pada suatu direktori yang sama dengan nama package-nya.

Contoh:

```
package politeknik;
public class Siswa {
        …
}
```

Suatu class dapat meng-import class lainnya sesuai dengan nama package yang dipunyainya. Satu hal yang perlu kita ketahui, pada saat kita ingin meng-import suatu class dalam suatu package, pastikan letak package tersebut satu direktori dengan class yang ingin meng-import.

Contoh:

```
import politeknik.Siswa;
public class IsiData {
        …
}
```

Kata kunci *this* sangat berguna untuk menunjukkan suatu member dalam class-nya sendiri. This dapat digunakan baik untuk data member maupun untuk function member, serta dapat juga digunakan untuk konstruktor. Adapun format penulisannya adalah:

- this.data_member : merujuk pada data member
- this.function_member() : merujuk pada function member
- this() : merujuk pada konstruktor

Contoh:

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

    public Mahasiswa(int nrp, String nama) {
        this.nrp=nrp;
        this.nama=nama;
    }
}
```

Percobaan 1:

Percobaan berikut ini menunjukkan bagaimana mengakses anggota dari suatu class.

```
public class Siswa {
    int nrp;
    public void setNrp(int i) {
        nrp=i;
    }
}

public class Test {
    public static void main(String args[]) {
        Siswa anak=new Siswa();
        anak.setNrp(5);
        System.out.println(anak.nrp);
    }
}
```

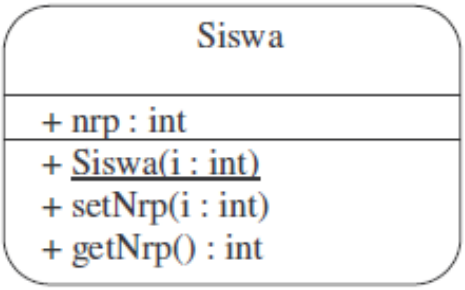
Percobaan 2:

Percobaan berikut ini menunjukkan bagaimana mengakses anggota dari suatu class.

```
public class Siswa {
    int nrp;
    String nama;
    public void setNrp(int i) {
        nrp=i;
    }
    public void setNama(String i) {
        nama=i;
    }
}
```

Percobaan 3:

Percobaan berikut ini mengimplementasikan enkapsulasi dalam pemrograman berorientasi objek, dari UML class diagram dibawah ini di turunkan kedalam bentuk program.

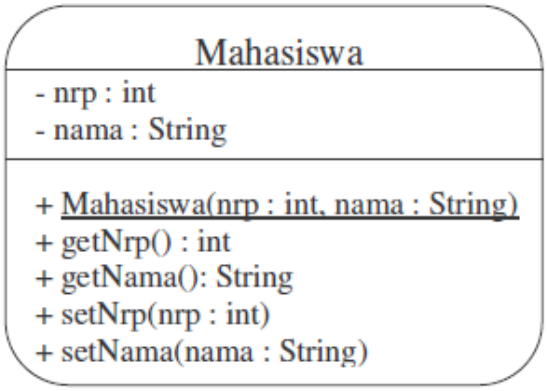



```
public class Siswa {
    public int nrp;

    public Siswa(int i) {
        nrp=i;
    }
```

```
    public void setNrp(int i) {
        nrp=i;
    }
    public int getNrp() {
        return nrp;
    }
}
```

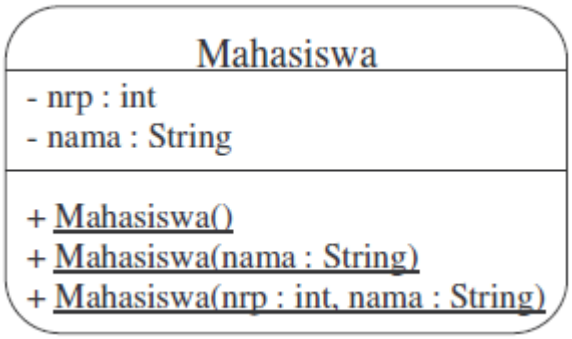
Percobaan 4:

Buatlah program untuk merepresentasikan UML dibawah ini.



Percobaan 5:

Percobaan berikut ini menunjukkan overloading constructor.



Dari class diagram tersebut, dapat diimplementasikan ke dalam program sebagai berikut:

```
public class Mahasiswa {
    private int nrp;
    private String nama;
    public Mahasiswa() {
        nrp=0;
        nama="";
    }

public Mahasiswa(String nama) {
```

```
        nrp=0;
        this.nama=nama;
}

public Mahasiswa(int nrp, String nama) {
       this.nrp=nrp;
       this.nama=nama;
    }
}
```

Percobaan 6:

Percobaan berikut ini menunjukkan bagaimana menggunakan kata kunci ***this***.

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

    public Mahasiswa(int nrp, String nama) {
        this.nrp=nrp;
        this.nama=nama;
    }
}
```

Percobaan 7:

Percobaan berikut ini menunjukkan bagaimana memakai kata kunci *this* pada overloading constructor.

```
public class Mahasiswa {
        private int nrp;
        private String nama;
        public Mahasiswa() {
            this(0,"");
        }
        Public Mahasiswa(String nama) {
            this(0,nama);
        }

        public Mahasiswa(int nrp, String nama) {
            this.nrp=nrp;
            this.nama=nama;
        }
}
```

Percobaan 8:

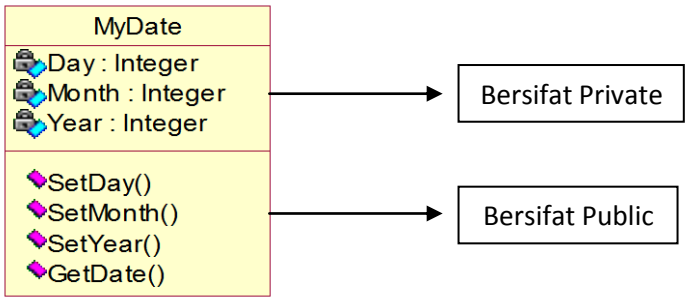
Percobaan berikut ini menunjukkan bagaimana menggunakan package dan import.

```
package sekolah;

public class Kelas {
    private int kodekelas;
```

```
    private String namakelas;
    private Mahasiswa mahasiswa;

    public Kelas(int kode, String nama) {
        this.kodekelas=kode;
        this.namakelas=nama;
    }

    public void setMhs(Mahasiswa mhs) {
        this.mahasiswa=mhs;
    }
}

public class Mahasiswa {
    private int nrp;
    private String nama;
    public Mahasiswa(int nrp, String nama) {
        this.nrp=nrp;
        this.nama=nama;
    }
}
```

Tugas 1: MyDate dan TestMyDate

Buatlah kelas MyDate dengan komposisi seperti pada gambar berikut:



Gambar 1. Class Diagram dari MyDate

- Atribut yang dimiliki adalah : Day, Month, Year : tipe integer bersifat private
- Constuctor yang bersifat public dengan parameter day, month, year untuk inisialisasi awal attribut MyDate(int day, int month, int year)
- Method yang dimiliki
- SetDay(int val) : return type bool, untuk set attribut Day
- SetMonth(int val) : return type bool, untuk set attribute Month
- SetYear(int val) : untuk set attribute Year
- GetDate() : return type string, untuk menampilkan tanggal, misal 20-2-2007
- Semua bersifat public

Buatlah kelas lain misal TestMyDate untuk memasukkan dan menampilkan data pada kelas MyDate :

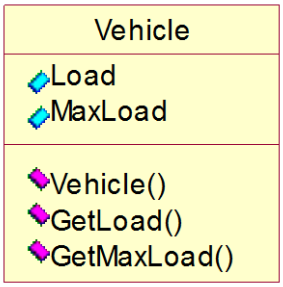
- Memasukkan tanggal (Day)

- Memasukkan bulan (Month)
- Memasukkan tahun (Year)
- Menampilkan tanggal yang dimasukkan (GetDate)

Buat perubahan pada kelas MyDate untuk mengatasi kesalahan input tanggal atau bulan Misal pada kelas TestMyDate berusaha memasukkan data melalui SetDay(32) tidak akan disimpan pada atrribut, dan digunakan nilai default.

Tugas 2: Vehicle dan TestVehicle

Buatlah kelas **Vehicle** yang mengimplementasikan sebuah kendaraan pengangkut barang.



Gambar 2. Class Diagram dari Vehicle

Kelas Vehicle terdiri dari dua buah atribut yang bertipe public: **load** (*the current weight of the vehicle's cargo*) dan **maxLoad** (*the vehicle's maximum cargo weight limit*). Tambahkan satu buah konstruktor yang bertipe public, yang digunakan untuk mengeset nilai atribut **maxLoad**.

```
Vehicle(double val){
       maxload = val;
}
```

Kemudian kita tambahkan dua buah methods yang bertipe public: **getLoad** (untuk mendapatkan nilai atribut **load**) dan **getMaxLoad** (untuk mendapatkan nilai atribut **maxLoad**). Semua data diasumsikan dalam satuan kilogram.

Buatlah kelas TestVehicle untuk menguji kelas Vehicle Perhatikan bahwa pada TestVehicle, dibuat Vehicle dengan kapasitas maksimum 10.000 kg. Tetapi pada program selanjutnya terdapat penambahan boxes yang melebihi kapasitas (10.050 kg). Kenapa demikian? Karena tidak ada pengecekan kapasitas maksimum sehingga vehicle nya kelebihan kapasitas. Untuk menyelesaikan masalah diatas, maka kelas Vehicle diubah dengan menyembunyikan data internal (**load** dan **maxLoad**) dan menyediakan method, **addBox**, sebagai fasilitas pengecekan terhadap **maxLoad** supaya tidak terjadi kelebihan kapasitas.

Lakukan modifikasi terhadap atribut **load** dan **maxLoad** → jadikan bertipe private.

Tambahkan method **addBox**. Method ini mempunyai satu argumen yaitu **weight** dalam satuan kilogram. Method addBox harus melakukan pengecekan terhadap penambahan box agar jangan sampai melebihi kapasitas maksimum.

Tambahkan method **subBox**. Method ini mempunyai satu argumen yaitu **weight** dalam satuan kilogram. Method addBox harus melakukan pengecekan terhadap pengurangan box agar jangan sampai kurang dari kapasitas minimum.

Bila terjadi pelanggaran terhadap kapasitas maksimum, maka penambahan box di tolak dan mengembalikan nilai **false**; jika tidak terjadi pelanggaran terhadap batas maksimum maka weight dari box diterima dan ditambahkan pada vehicle dan mengembalikan nilai **true.** Gunakan statement if-else untuk melakukan pengecekan terhadap kapasitas maksimum

Kompile Vehicle dan TestVehicle. Jalankan TestVehicle. Pada pengujian kelas Vehicle baru, lakukan penambahan box sampai terjadi kelebihan kapasitas maksimal sehingga method **addBox** mengembalikan nilai **false**, dalam arti bahwa terjadi penolakan terhadap penambahan box.

Vehicle.java

```
class Vehicle {
double load, maxload;
private double weight;

    public double getLoad(){
        return load;
    }

    public void GetMaxLoad(double value){
        maxload=value;
    }

    public boolean addBox(double weight){
        load=load+weight;
        if(load<=maxload){
            return true;
        }
        else{
            load=load-weight;
             return false;
        }
    }

    public boolean subBox(double weight){
        load=load-weight;
        if(load>=0){
            return true;
        }
        else{
            load=load+weight;
            return false;
        }
    }

    Vehicle(double value){
         maxload=value;
    }
}
```

TestVehicle.java

```
public class TestVehicle{
    public static void main(String[] args){

        System.out.println("Create a vehicle with a 10,000kg max load");

        Vehicle vehicle=new Vehicle(10000.0);

        System.out.println("Add box #1(500kg) :" + vehicle.addBox(500.0));
        System.out.println("Add box #2(250kg) :" + vehicle.addBox(250.0));
        System.out.println("Add box #3(5000kg) :" + vehicle.addBox(5000.0));
        System.out.println("Add box #4(4000kg) :" + vehicle.addBox(4000.0));
        System.out.println("Add box #5(300kg) :" + vehicle.addBox(300.0));

        System.out.println("Vehicle load is "+ vehicle.getLoad()+"kg\n");
    }
}
```