Praktikum 5

Array, Vector dan Hash Map

Tujuan

Membuat dan menggunakan array satu dimensi dan multi dimensi.
Memahami konsep referensi array, vector serta hash map.

Dasar Teori

Array

Array adalah suatu kumpulan data pada suatu variabel. Cara mendeklarasikan suatu array adalah sebagai berikut:

```
tipe_array nama_array[];
tipe_array[] nama_array;
```

Contoh :

```
int nilai[];
char[] huruf;
```

Agar kita dapat memesan tempat di memori untuk menampung elemen-elemen array, kita perlu membuat array. Adapun caranya adalah dengan memakai *new* karena di dalam Java suatu array adalah dianggap suatu obyek. Format penulisannya adalah sebagai berikut:

```
nama_array = new tipe_array[total_elemen_array];
```

Contoh :

```
int nilai[];
nilai = new int[5];
```

Untuk dapat mengakses elemen array dapat dilakukan dengan menyebutkan elemen ke berapa dari array yang akan diakses, seperti berikut ini:

```
nama_array[elemen_array]
```

Kita juga dapat melakukan deklarasi dan pembuatan array hanya pada satu baris *statement*. Adapun format penulisannya adalah sebagai berikut:

```
tipe_array nama_array[] = new tipe_array[total_elemen_array];
```

Contoh :

```
int nilai[] = new int[5];
```

Inisialisasi array dapat dilakukan dengan format penulisan sebagai berikut:

```
tipe_array nama_array[] = {nilai_indeks_0, nilai_indeks_1, … ,
                           nilai_indeks_n};
```

Contoh :

```
int nilai[] = {70, 65, 85};
```

Kita dapat membuat array multi dimensi dengan cara menambahkan tanda [] sebanyak dimensi yang ingin dibuat. Sebagai contoh adalah sebagai berikut:

```
int x[][] = new int[3][4];
```

Baris *statement* diatas berarti kita ingin membuat array berdimensi 2, dengan 3 elemen di dimensi ke-1 dan 4 elemen di dimensi ke-2.

Untuk mengetahui panjang dari suatu array yang telah kita buat, kita dapat memakai properti *length*. Adapun format untuk menggunakan *length* adalah sebagai berikut:

var_array.length : total elemen array pada dimensi 1
var_array[i].length : total elemen array pada dimensi 2 untuk indeks ke-i pada dimensi 1
var_array[i][j].length : total elemen array pada dimensi 3 untuk indeks ke-i pada dimensi 1 dan indeks ke-j pada dimensi 2 dan seterusnya.

Isi dari suatu array dapat kita kopi pada array yang lain dengan memanfaatkan method arraycopy() pada class System. Format penulisannya sebagai berikut :

```
System.arraycopy(array1, p1, array2, p2, n);
```

dimana :

array1 = array asal/sumber pengkopian
array2 = array tujuan pengkopian
p1 = posisi indeks awal pengkopian pada array asal
p2 = posisi indeks awal pengkopian pada array tujuan
n = banyaknya elemen array yang akan dikopi

Suatu array juga dapat me-refer (merujuk) ke array yang lain, dengan kata lain merujuk pada alamat memori yang sama. Sebagai contoh adalah program berikut ini:

```
int nilai[] = {10, 20, 30};
int result[];
result = nilai;
```

Di baris ketiga, kita meng-*assign* array nilai ke array result. Akibatnya, array result akan me-*refer* (merujuk) pada array nilai, sehingga kedua array tersebut merujuk alamat memori yang sama.

Vector

Menggunakan Vector mirip dengan menggunakan ArrayList. Perbedaannya adalah nama method yang berbeda untuk melakukan tugas yang sama, atau nama metode yang berbeda untuk melakukan tugas yang sama.

Seperti ArrayList, suatu Vector mirip dengan array Object yang bisa berkembang jika diperlukan. Konstruktor **new** Vector() membuat vektor tanpa elemen.

Misalnya vec adalah suatu Vector. Maka :

- `vec.size()` adalah fungsi untuk mengembalikan jumlah elemen di dalam vektor.
- `vec.addElement(obj)` akan menambahkan Object obj di akhir vektor. Sama dengan metode add() pada ArrayList.
- `vec.removeElement(obj)` menghapus obj dari dalam vektor, kalau ada. Hanya objek pertama yang ditemui akan dihapus. Sama dengan remove(obj) pada kelas ArrayList
- `vec.removeElementAt(N)` menghapus elemen ke-N. N harus berada pada rentang 0 hingga vec.size() - 1. Sama dengan remove(N) pada ArrayList
- `vec.setSize(N)` akan mengubah ukuran vektor menjadi N. Jika di dalam vektor terdapat elemen yang jumlahnya lebih banyak dari N, maka elemen lainnya akan dihapus. Jika lebih sedikit, maka tempat kosong akan diisi dengan **null**. Kelas ArrayList tidak memiliki metode seperti ini.

Kelas Vector memiiki banyak metode lagi, akan tetapi ini adalah metode yang sering digunakan.

Hash Map

HashMap adalah class implementasi dari Map, Map itu sendiri adalah interface yang mempunyai fungsi untuk memetakan nilai dengan key unik. HashMap sangat bermanfaat sebagai memory record management, dimana tiap record dapat disimpann di sebuah Map, kemudian setiap Map diletakkan pada vector, list atau set yang masih turunan dari Collection. Demikian pula Hashmap sangat baik untuk menghandle resultset dari hasil query.

Contoh:

```
Map arecord= new HashMap();
arecord.put("id","1001");
arecord.put("nama","Adi");
arecord.put("alamat","Jl. Ahmad Yani 22");
```

A. Array

Percobaan 1:

Percobaan berikut ini menunjukkan pemakaian array, mengisi array dengan nilai numerik dan bagaimana menampilkan isi dari sebuah array (satu dimensi).

```
public class ArrayDemo {

    public static void main(String[] args) {
        int []anArray;
        anArray=new int[10];

        for (int i=0; i<anArray.length; i++){
            anArray[i] = i;
            System.out.println(anArray[i]+"");
        }
    System.out.println();
    }
}
```

Percobaan 2:

Percobaan berikut ini menunjukkan pemakaian array, mengisi array dengan karakter dan bagaimana menampilkan serta memanipulasi isi dari sebuah array (satu dimensi).

```
public class ArrayOfStringsDemo {

    public static void main (String[]args){
        String[] anArray={"String One", "String Two", "String Three"};

        for (int i=0; i<anArray.length; i++){
            System.out.println(anArray[i].toLowerCase());
        }
    }
}
```

Percobaan 3:

Percobaan berikut ini menunjukkan pemakaian objek dalam bentuk array , mengisi array dengan bermacam-macam tipe data dan bagaimana menampilkannya.

```
public class TestArrayObject {

    public static void main(String[] args){
        Object []a = new Object[4];
        a[0] = new String ("Hello, World!");
        a[1] = new Long (44);
        a[2] = new Float(1.5);

        for (int i=0; i<a.length; i++)
            System.out.println("a["+ i +"]=" +a[i]);
    }
}
```

Percobaan 4:

Percobaan berikut ini menunjukkan pemakaian array di dalam array (array 2 dimensi), bagaimana mengisi array dan menampilkannya.

```
public class ArrayOfArrayDemo {

    public static void main (String[]args){
        String[][]cartoons =
        {
            {"Flintstones", "Fred", "Wilma", "Pebbles", "Dino"},
            {"Rubbles", "Barney","Betty"},
            {"Jetsons", "George"},
            {"Scooby Doo gang", "Scooby Doo"}
        };

        for (int i=0; i<cartoons.length; i++){
            System.out.println(cartoons[i][0]+":");
            for (int j=1; j<cartoons[i].length; j++){
                System.out.println(cartoons [i][j]+"");
            }
            System.out.println();
        }
    }
}
```

Percobaan 5:

Percobaan berikut ini menunjukkan pemakaian array 2 dimensi yang direpresentasikan kedalam sebuah matriks yang berisi nilai numerik.

```
public class ArrayOfArrayDemo2 {

    public static void main (String[]args){
        int [][] aMatrix = new int [4][];

        for (int i=0; i<aMatrix.length; i++){
            aMatrix[i] = new int[5];
            for (int j=0; j<aMatrix[i].length; j++){
                aMatrix[i][j] = i+j;
            }
        }
        for (int i=0; i<aMatrix.length; i++){
            for (int j=0; j<aMatrix[i].length; j++){
                System.out.print(aMatrix[i][j]+"");
            }
            System.out.println();
        }
    }
}
```

Percobaan 6:

Percobaan berikut ini menunjukkan pemakaian fungsi arraycopy untuk menyalin isi dari sebuah array ke dalam array yang lainnya.

```
public class ArrayCopyDemo {

public static void main (String[] args){
        char[] copyFrom = {'d','e','c','a','f','f','e','i','n','a','t',
'e','d'};
        char[] copyTo=new char[7];
        char[] coba = new char[20];
        coba= copyFrom;
        coba[0]='t';

        System.arraycopy(copyFrom, 2, copyTo, 0, 7);
        System.out.println(new String(copyTo));
        System.out.println(new String(coba));
    }
}
```

B. Vector

Percobaan 7:

Percobaan berikut ini menunjukkan bagaimana cara membuat dan menginisialisasi vektor.

```
import java.util.Vector;

public class CreateVector {

    public static void main(String[] args) {
        //dengan inisial ukuran
        Vector v = new Vector(10);
        for(int i=0; i<=100;i++){
            System.out.println("Size:"+v.size());
            System.out.println("Capacity:"+v.capacity());
            v.add("hello");
        }

        //tanpa inisialisasi
        Vector w = new Vector();
        for(int i=0;i<=100;i++){
            System.out.println("Size:"+w.size());
            System.out.println("Capacity:"+w.capacity());
            v.add("hello");
        }
    }
}
```

Percobaan 8

Percobaan berikut ini menunjukkan bagaimana cara mengakses elemen pada vektor.

```
import java.util.Vector;
import java.util.ListIterator;

public class Sample {
    public static void main(String[] args) {
        Vector theVector = new Vector();

        for(int i=0; i<=10; i++){
```

```
            theVector.add(new Integer(i));
        }
        System.out.println("isi theVector:"+theVector.toString());
        System.out.println("Ukuran:"+theVector.size());

        System.out.println("Masing-masing isi elemen vector:");
        ListIterator iter= theVector.listIterator();

        while(iter.hasNext()){
            System.out.println(iter.next());
        }
    }
}
```

Percobaan 9:

Percobaan berikut ini menunjukkan bagaimana cara menyisipkan elemen pada vektor.

```
import java.util.Vector;

public class InsertElement {

    public static void main(String[] args){
        Vector v=new Vector();

        v.add(new Integer(20));
        v.add(new Long(10000000L));
        v.add(new Float(123.44f));
        v.add(new Double(123.12312312344));

        System.out.println(v);
        v.add(2,new String("Ini sisipan"));
        System.out.println(v);
        v.set(3,new String("Nilai Pengganti"));
        System.out.println(v);
    }
}
```

Percobaan 10:

Percobaan berikut ini menunjukkan bagaimana cara mencari elemen pada vektor.

```
import java.util.Vector;

public class FindVector {

    public static void main(String[] args){
        String data[]={"java","Source","and","Support",".","Java"};

        Vector v = new Vector();
        for(int i=0, n= data.length; i<n;i++){
            v.add(data[i]);
        }
        System.out.println(v);
        System.out.println("Contains Java?:"+v.contains("Java"));
        System.out.println("Contains Java2s?:"+v.contains("Java2s"));
        System.out.println("Where's and?:"+v.indexOf("and"));
        System.out.println("Where's source?:"+v.indexOf("Source"));
        System.out.println("Where's java from begin?: "+v.indexOf
```

```
                            ("Java"));
        System.out.println("Where's java from end?:"+v.indexOf("Java"));
    }
}
```

Percobaan 11:

Percobaan berikut ini menunjukkan bagaimana cara menghapus elemen pada vektor.

```
import java.util.Vector;

public class RemoveVector {

    private static int i;
    static boolean removeAll(Vector v, Object e){
        Vector v1=new Vector();
        v1.add(e);
        return v.removeAll(v1);
    }

    public static void main(String []args){
        Vector v= new Vector();
        for(int i=0,n=args.length; i<n ;i++){
            v.add(args[i]);
        }

        System.out.println(v);
        System.out.println(removeAll(v,args[0]));
        System.out.println(v);
    }
}
```

C. HashMap

Percobaan 12:

Percobaan berikut ini menunjukkan pemakaian Hash Map sederhana,

```
import java.util.HashMap;

public class JavaHashMapExample {

    public static void main (String[]args){

        HashMap hMap = new HashMap();

        hMap.put("One", new Integer(1));
        hMap.put("Two", new Integer(2));

        Object obj= hMap.get("One");
        System.out.println(obj);

        obj=hMap.get("Two");
        System.out.println(obj);
    }
}
```

Percobaan 13:

Percobaan berikut ini menunjukkan bagaimana cara mengakses elemen pada HashMap.

```
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;

public class IterateValueOfHashMap {

    public static void main(String[]args){
        HashMap hMap = new HashMap();

        hMap.put("1", "One");
        hMap.put("2", "Two");
        hMap.put("3", "Three");
        hMap.put("4", "Four");
        hMap.put("5", "Five");

        System.out.println("Size of hashMap after addition:"
+hMap.size());

        Object obj = hMap.remove("2");
        System.out.println("Size of hashMap after removal:"
+hMap.size());

        Collection c = hMap.values();
        Iterator itr = c.iterator();

        while(itr.hasNext())
             System.out.println(itr.next());
    }
}
```

Percobaan 14:

Percobaan berikut ini menunjukkan bagaimana cek key dan value elemen HashMap.

```
import java.util.HashMap;
import java.util.Iterator;

public class CheckHashMap {

    public static void main(String[]args){

        HashMap hashMap = new HashMap();

        hashMap.put("one", new Integer(1));
        hashMap.put("two", new Integer(2));
        hashMap.put("three", new Integer(3));

        if (hashMap.containsValue(new Integer (1))){
            System.out.println("HashMap contains 1 as value");
        }
        else{
           System.out.println("HashMap does not contain 1 as value");
        }

        if (hashMap.containsKey("One")){
            System.out.println("HashMap contains one as key");
```

```
        }
        else{
            System.out.println("HashMap does not ontain one as value");
        }
    }
}
```

Tugas

1. Mencari nilai rata-rata mata kuliah dari daftar nilai siswa

Diketahui daftar nilai siswa sebagai berikut:

NRP	Nama Mhs	RPL	BD	PBO
1	Ahmad	81	90	62
2	Adang	50	83	87
3	Dani	89	55	65
4	Edi	77	70	92

Buatlah program untuk menampilkan laporan sebagai berikut:

```
NRP    Rata-rata    Nilai
------------------------
1      77.67         AB
2      73.33         AB
3      69.67         B
4      79.67         AB
------------------------
```

Tambahkan juga method untuk INSERT, UPDATE, dan DELETE.

2. Mendeteksi bilangan prima

Buatlah suatu program untuk mendeteksi suatu bilangan itu termasuk bilangan prima

```
Masukkan bilangan? 8
8 bukan termasuk bilangan prima

Masukkan bilangan? 11
11 adalah bilangan prima
```

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);
    }
}
```

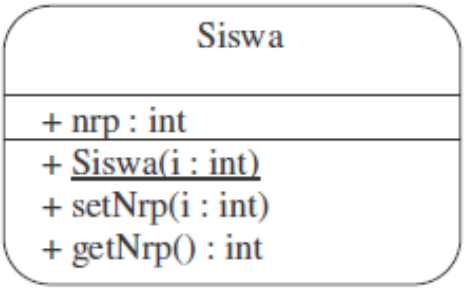
Percobaan 2:

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.

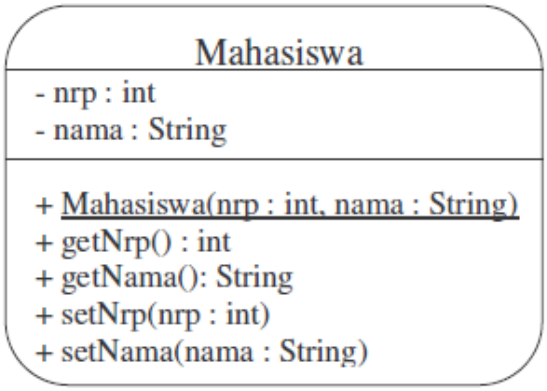



```
public class Siswa {
    public int nrp;

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

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

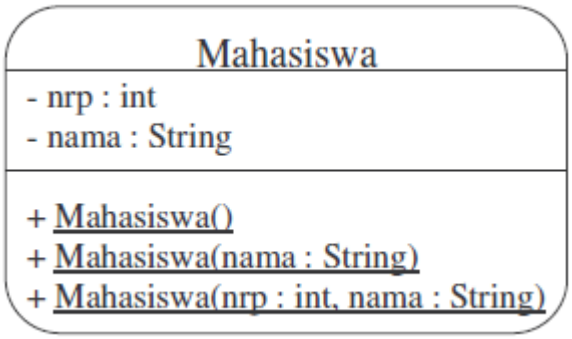
Percobaan 4:

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:

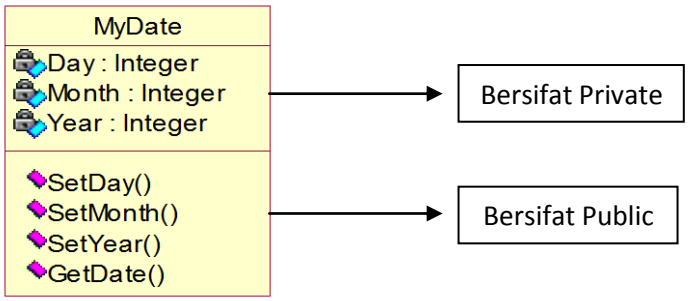
Percobaan berikut ini menunjukkan bagaimana menggunakan package dan import.

```
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 :

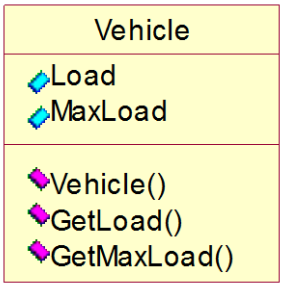
- Memasukkan tanggal (Day)

- 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");
    }
}
```

Praktikum 7

Inheritance (Pewarisan)

Tujuan

Memahami dan menerapkan konsep inheritance dalam pemrograman berorientasi objek.

Dasar Teori

Konsep inheritance ini mengadopsi dunia riil dimana suatu entitas/obyek dapat mempunyai entitas/obyek turunan. Dengan konsep inheritance, sebuah class dapat mempunyai class turunan. Suatu class yang mempunyai class turunan dinamakan parent class atau base class. Sedangkan class turunan itu sendiri seringkali disebut subclass atau child class. Suatu subclass dapat mewarisi apa-apa yang dipunyai oleh parent class-nya, sehingga member dari suatu subclass adalah terdiri dari apa-apa yang ia punyai dan juga apa-apa yang ia warisi dari class parent-nya. Kesimpulannya, boleh dikatakan bahwa suatu subclass adalah tidak lain hanya memperluas (extend) parent class-nya.

Di dalam Java untuk mendeklarasikan suatu class sebagai subclass dilakukan dengan cara menambahkan kata kunci extends setelah deklarasi nama class, kemudian diikuti dengan nama parent class-nya. Kata kunci extends tersebut memberitahu kompiler Java bahwa kita ingin melakukan perluasan class. Berikut adalah contoh deklarasi inheritance:

Contoh:

```
public class B extends A {
...
}
```

Contoh diatas memberitahukan kompiler Java bahwa kita ingin meng-extend class A ke class B. Dengan kata lain, class B adalah subclass (class turunan) dari class A, sedangkan class A adalah parent class dari class B.

Java hanya memperkenankan adanya single inheritance. Konsep single inheritance hanya memperbolehkan suatu sublass mempunyai satu parent class. Dengan konsep single inheritance ini, masalah pewarisan akan dapat diamati dengan mudah.

Dalam konsep dasar inheritance dikatakan bahwa suatu subclass adalah tidak lain hanya memperluas (extend) parent class-nya. Contoh :

```
public class Pegawai {
	public String nama;
	public double gaji;
}

public class Manajer extends Pegawai {
	public String departemen;
}
```

Pada saat class Manajer menurunkan atau memperluas (extend) class Pegawai, maka ia mewarisi data member yang dipunyai oleh class Pegawai. Dengan demikian, class Manajer mempunyai data member yang diwarisi oleh Pegawai (nama, gaji), ditambah dengan data member yang ia punyai (departemen).

Pengaksesan member yang ada di parent class dari subclass-nya tidak jauh berbeda dengan pengaksesan member subclass itu sendiri. Contoh:

Suatu parent class dapat tidak mewariskan sebagian member-nya kepada subclass-nya. Sejauh mana suatu member dapat diwariskan ke class lain, ataupun suatu member dapat diakses dari class lain, sangat berhubungan dengan access control (kontrol pengaksesan). Di dalam java, kontrol pengaksesan dapat digambarkan dalam tabel berikut ini:

Modifier	class yang sama	package yang sama	subclass	class manapun
private	√			
default	√	√		
protected	√	√	√	
public	√	√	√	√

Kata kunci *super* dipakai untuk merujuk pada member dari parent class, sebagaimana kata kunci *this* yang dipakai untuk merujuk pada member dari class itu sendiri. Adapun format penulisannya adalah sebagai berikut:

super.data_member : merujuk pada data member pada parent class
super.function_member() : merujuk pada function member pada parent class
super() : merujuk pada konstruktor pada parent class

Contoh:

```
public class Siswa {
	private int nrp;
	public setNrp(int nrp) {
		this.nrp=nrp;
	}
}
```

Percobaan 1:

Percobaan berikut ini menunjukkan penggunaan kata kunci “super”.

```
class Parent {
    public int x = 5;
}

class Child extends Parent {
    public int x = 10;
    public void Info(int x) {
    System.out.println("Nilai x sebagai parameter = " + x);
    System.out.println("Data member x di class Child = " + this.x);
    System.out.println("Data member x di class Parent = " +
super.x);
    }
}

public class NilaiX {
    public static void main(String args[]) {
        Child tes = new Child();
        tes.Info(20);
    }
}
```

Percobaan 2:

Percobaan berikut ini menunjukkan penggunaan kontrol akses terhadap atribut parent class. Mengapa terjadi error, dan bagaimana solusinya?

```
public class Pegawai {
   private String nama;
   public double gaji;
}

public class Manajer extends Pegawai {
   public String departemen;

   public void IsiData(String n, String d) {
      nama=n;
      departemen=d;
   }
}
```

Percobaan 3:

Percobaan berikut ini menunjukkan penggunaan konstruktor yang tidak diwariskan. Mengapa terjadi error, dan bagaimana solusinya?

```
public class Parent {
     // kosong
}

public class Child extends Parent {
    int x;
    public Child() {
        x = 5;
    }
```

```
}
```

Percobaan 4:

Percobaan berikut ini menunjukkan penggunaan kelas Employee dan subkelas Manager yang merupakan turunannya. Kelas TestManager digunakan untuk menguji kelas Manager.

```
class Employee {
    private static final double BASE_SALARY = 15000.00;
    private String Name = "";
    private double Salary = 0.0;
    private Date birthDate;

    public Employee() {}
    public Employee(String name, double salary, Date DoB){
        this.Name=name;
        this.Salary=salary;
        this.birthDate=DoB;
    }
    public Employee(String name,double salary){
        this(name,salary,null);
    }
    public Employee(String name, Date DoB){
        this(name,BASE_SALARY,DoB);
    }
    public Employee(String name){
        this(name,BASE_SALARY);
    }

    public String GetName(){ return Name;}
    public double GetSalary(){ return Salary; }
}

class Manager extends Employee {

    //tambahan attribrute untuk kelas manager
    private String department;

    public Manager(String name,double salary,String dept){
        super(name,salary);
        department=dept;
    }
    public Manager(String n,String dept){
        super(n);
        department=dept;
    }
    public Manager(String dept){
        super();
        department=dept;
    }
    public String GetDept(){
        return department;
    }
}

public class TestManager {

    public static void main(String[] args) {
        Manager Utama = new Manager("John",5000000,"Financial");
        System.out.println("Name:"+ Utama.GetName());
        System.out.println("Salary:"+ Utama.GetSalary());
```

```
        System.out.println("Department:"+ Utama.GetDept());

        Utama = new Manager("Michael","Accounting");
        System.out.println("Name:"+ Utama.GetName());
        System.out.println("Salary:"+ Utama.GetSalary());
        System.out.println("Department:"+ Utama.GetDept());
    }
}
```

Percobaan 5:

Percobaan berikut ini menunjukkan penggunaan kelas MoodyObject dengan subkelas HappyObject dan SadObject. Kelas MoodyTest digunakan untuk menguji kelas dan subkelas.

- SadObject berisi :
    - sad, method untuk menampilkan pesan, tipe public
- HappyObject berisi :
    - laugh, method untuk menampilkan pesan, tipe public
- MoodyObject berisi :
    - getMood, memberi nilai mood sekarang, tipe public, return type string
    - speak, menampilkan mood, tipe public

```
public class MoodyObject {

    protected String getMood(){
        return "moody";
    }
    public void speak(){
        System.out.println("I am"+getMood());
    }
    void laugh() {}
    void cry() {}
}

public class SadObject extends MoodyObject{
    protected String getMood(){
        return "sad";
    }
    public void cry(){
        System.out.println("Hoo hoo");
    }
}

public class HappyObject extends MoodyObject{
    protected String getMood(){
        return"happy";
    }
    public void laugh(){
    System.out.println("Hahaha");
    }
}

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

        MoodyObject m = new MoodyObject();

        //test perent class
        m.speak();
```

```
        //test inheritance class
        m = new HappyObject();
        m.speak();
        m.laugh();

        //test inheritance class
        m=new SadObject();
        m.speak();
        m.cry();
    }
}
```

Percobaan 6:

Percobaan berikut ini menunjukkan penggunaan kelas A dan dengan subkelas B. Simpan kedua kelas ini dalam 2 file yang berbeda (A.java dan B.java) dan dalam satu package. Perhatikan proses pemanggilan konstruktor dan pemanggilan variabel

```
class A {
    String var_a = "Variabel A";
    String var_b = "Variabel B";
    String var_c = "Variabel C";
    String var_d = "Variabel D";

    A(){
        System.out.println("Konstruktor A dijalankan");
    }
}

class B extends A{
    B(){
        System.out.println("Konstruktor B dijalankan ");
        var_a = "Var_a dari class B";
        var_b = "Var_a dari class B";
    }

    public static void main(String args[]){

        System.out.println("Objek A dibuat");
        A aa= new A();
        System.out.println("menampilkan nama variabel obyek aa");
        System.out.println(aa.var_a);
        System.out.println(aa.var_b);
        System.out.println(aa.var_c);
        System.out.println(aa.var_d);
        System.out.println("");

        System.out.println("Objek B dibuat");
        B bb= new B();
        System.out.println("menampilkan nama variabel obyek bb");
        System.out.println(bb.var_a);
        System.out.println(bb.var_b);
        System.out.println(bb.var_c);
        System.out.println(bb.var_d);
    }
}
```

Percobaan 7:

Percobaan berikut ini menunjukkan penggunaan Inheritance dan Overriding method pada kelas Bapak dan subkelas Anak. Terjadi override pada method show_variabel. Perhatikan perubahan nilai pada variabel a, b, dan c.

```
class Bapak {
    int a;
    int b;

    void show_variabel(){
        System.out.println("NIlai a="+ a);
        System.out.println("NIlai b="+ b);
    }
}

class Anak extends Bapak{
    int c;
    void show_variabel(){
        System.out.println("NIlai a="+ a);
        System.out.println("NIlai b="+ b);
        System.out.println("NIlai c="+ c);
    }
}

public class InheritExample {

    public static void main(String[] args) {

        Bapak objectBapak = new Bapak();
        Anak objectAnak = new Anak();

        objectBapak.a=1;
        objectBapak.b=1;
        System.out.println("Object Bapak (Superclass):");

        objectBapak.show_variabel();
        objectAnak.c=5;
        System.out.println("Object Anak (Superclass dari Bapak):");
        objectAnak.show_variabel();
    }
}
```

Kemudian lakukan modifikasi pada method show_variabel() pada class Anak. Gunakan super untuk menampilkan nilai a dan b (memanfaatkan method yang sudah ada pada superclass).

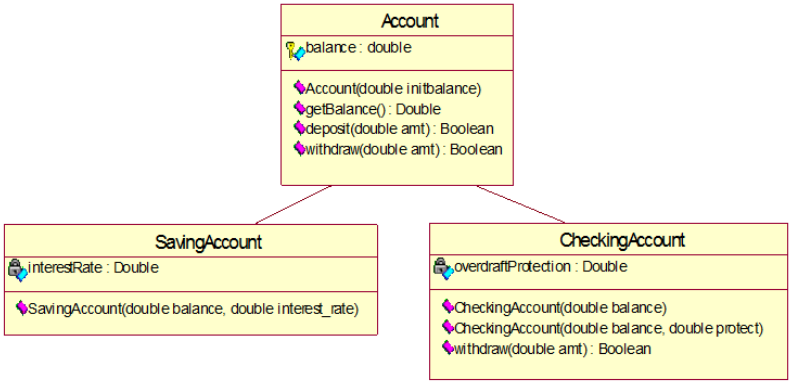
Percobaan 8:

Percobaan berikut ini menunjukkan penggunaan overriding method pada kelas Parent dan subkelas Baby, saat dilakukan pemanggilan konstruktor superclass dengan menggunakan super.

```
public class Parent {
    String parentName;
    Parent(){}

    Parent(String parentName){
        this.parentName = parentName;
        System.out.println("Konstruktor parent");
    }
```

```
}

class Baby extends Parent {
    String babyName;

    Baby(String babyName){
        super();
        this.babyName = babyName;
        System.out.println("Konstruktor Baby");
        System.out.println(babyName);
    }

    public void Cry(){
        System.out.println("Owek owek");
    }
}

public class TestBaby {

    public static void main(String args[]){
        Baby x = new Baby("Hafidza");
        x.Cry();
    }
}
```

Tugas: Pembuatan kelas Account dan subkelas SavingAccount, CheckingAccount



- Buat kelas Account sesuai dengan diagram UML untuk kelas Account sebelumnya, dengan definisi :
    - Atribut *balance* tipe double, dan sifat protected
    - Constructor *Account* untuk memberi nilai awal *balance*
    - Method *getBalance* untuk mendapatkan nilai *balance*
    - Method *deposit* untuk menambah nilai *balance*
    - Method *withdraw* untuk mengambil nilai *balance*

- Buat subkelas SavingAccount sesuai dengan diagram UML sebelumnya dengan definisi :
    - Kelas SavingAccount merupakan turunan kelas Account, gunakan keyword *extends*.
    - Atribut interestRate, tipe double, sifat private
    - Constructor *SavingAccount* dengan parameter *balance* dan *interest_rate*. Constructor ini harus passing parameter balance ke

parent constructor dengan menggunakan super(balance) dan mengeset nilai variabel interestRate dengan nilai interest_rate.

- Buat kelas CheckingAccount sesuai dengan diagram UML sebelumnya dengan definisi :
    - Kelas CheckingAccount merupakan turunan kelas Account, gunakan keyword *extends*.
    - Atribut overdraftProtection, tipe double, sifat private
    - Terdapat public constructor dengan dua parameter: *balance* and *protect*. Constructor ini harus passing parameter *balance* ke parent constructor dengan menggunakan super(balance) dan mengeset nilai variabel *overdraftProtection* dengan nilai *protect*.
    - Constructor dengan satu parameter yaitu balance. Constructor ini harus passing parameter balance ke lokal constructor dengan menggunakan this. Perhatikan bahwa constructor lain yang ada adalah constructor dengan dua parameter. Maka buat nilai protect default adalah -1.0 yang berarti bahwa pada account tidak terdapat *overdraftProtection*.
    - Saldo = balance + overdraftProtection
    - overdraftProtection = Saldo minimal, yaitu saldo yang diharapkan tidak boleh diambil pada suatu rekening, kecuali bila konsumen ingin menutup rekening.
    - Class CheckingAccount harus mengoverride method withdraw. Method withdraw harus melakukan cek terhadap saldo (balance) apakah jumlahnya cukup bila terjadi pengambilan sejumlah uang (amount). Cek yang dilakukan adalah sebagai berikut :
    - Jika balance – amount => 0.0 maka proses pengambilan diperbolehkan dan mengembalikan nilai true. Dan selanjutnya set balance = balance – amount;
    - Jika balance – amount < 0.0 maka lakukan cek sebagai berikut:
    - Jika tidak ada overdraftProtection (nilai = -1.0) atau overdraftProtection < overdraftNeeded (amount-balance) maka gagalkan proses pengambilan uang dengan mengembalikan nilai false.
    - Jika terdapat overdraftProtection atau overdraftProtection > overdraftNeeded (amount-balance) maka proses pengambilan uang berhasil dengan mengembalikan nilai true. Dan selanjutnya set balance = 0.0; overdraftProtection = overdraftProtection – overdraftNeeded;
    - Constructor dengan satu parameter yaitu balance. Constructor ini harus passing parameter balance ke lokal constructor dengan menggunakan this. Perhatikan bahwa constructor lain yang ada adalah constructor dengan dua parameter. Maka buat nilai protect default adalah -1.0 yang berarti bahwa pada account tidak terdapat *overdraftProtection*.
    - Saldo = balance + overdraftProtection, overdraftProtection adalah saldo minimal, yaitu saldo yang diharapkan tidak boleh diambil pada suatu rekening, kecuali bila konsumen ingin menutup rekening.

Proteksi Cerukan adalah *overdraft protection* yaitu fasititas kredit kepada nasabah penyimpan dana untuk menutupi cerukan; fasilitas kredit bank tersebut memungkinkan nasabah untuk menarik cek yang melebihi dana tersedia pada saldo akunnya sehingga kelebihan penarikan dana tersebut dikenakan bunga harian;

apabila kelebihan penarikan dana ditutup dengan fasilitas kreditnya, kelebihan penarikan itu tidak dikenakan bunga harian.

Praktikum 8

Polymorphism

Tujuan

Mengetahui

Dasar Teori

Mengetahui

Praktikum 9

Advanced Class (1) : Feature

Tujuan

Mengetahui

Dasar Teori

Mengetahui

Praktikum 10

Advanced Class (2) :

Abstract, Interface dan Inner Class

Tujuan

Mengetahui

Dasar Teori

Mengetahui

Praktikum 11

Exception Handling (Penanganan Kesalahan)

Tujuan

Mengetahui

Dasar Teori

Mengetahui

Praktikum 12

Collection

Tujuan

Mengetahui

Dasar Teori

Mengetahui

Praktikum 13

Graphical User Interface (1):

Layout Manager

Tujuan

Mengetahui

Dasar Teori

Mengetahui

Praktikum 14

Graphical User Interface (2):

Event Handling

Tujuan

Mengetahui

Dasar Teori

Mengetahui