MODUL I
MENGENAL UNIT/KELAS/OBJEK

1. TUJUAN

- Membuat dan menjelaskan bagian-bagian kelas
- Membuat/menginstansiasi objek dari suatu kelas
- Mengunakan objek

2. TEORI DASAR

Sebuah sistem yang dibangun berdasarkan metoda berorientasi objek adalah sebuah sistem yang komponennya di enkapsulasi menjadi kelompok data dan fungsi, yang dapat mewarisi atribut dan sifat dari komponen lainnya, dan komponen-komponen tersebut saling berinteraksi satu sama lain.

Bahasa Pemrograman Yang berorientasi OBJEK memiliki kemampuan dalam pengelolaan program yang lebih diarahkan pada pembentukan objek. Dengan menerapkan konsep ini program akan lebih mudah untuk dikembangkan karena sifatnya yang lebih modular.

Dalam konsep object oriented akan kita temukan kata object dan class, class merupakan pola / template yang menggambarkan kumpulan object yang mempunyai sifat yang sama, perilaku, atau disebut dengan himpunan object sejenis. Sementara object adalah implementasi dari class. Tabel berikut contoh ilustrasi kelas dan objek.

Tabel : Contoh class car dan object-object nya

Kelas Mobil		Objek Mobil A	Objek Mobil B
Instan Variabel	nomor Plat	ABC 111	XYZ 123
	Warna	Biru	Merah
	Manufaktur	Mitsubishi	Toyota
	Kecepatan	50 km/h	100 km/h
Instan Metode	method akselerasi		
	method belok		
	method rem		

Enkapsulation

Encapsulation adalah proses menyembunyikan detil implementasi sebuah objek atau pembungkusan attribut (field atau variabel) dan tingkah laku (metode) di dalam sebuah kelas.

Manfaat dari proses enkapsulasi adalah :

Modularitas : Kode sumber dari sebuah objek dapat dikelola secara independen dari kode sumber objek yang lain.

Information Hiding : Karena kita dapat menentukan hak akses sebuah variabel/method dari objek, dengan demikian kita bisa menyembunyikan informasi yang tidak perlu diketahui objek lain.

Membuat kelas

Merupakan deklarasi atau identitas dari suatu Class. Penamaan suatu class mempunyai aturan umum :

- tidak menggunakan spasi atau menggunakan spasi dengan diganti "_"
- Apabila nama kelas terdiri atas 2 kata maka huruf pertama dari kedua kata tersebut
- Nama class harus dimulai dengan huruf KAPITAL
- Tidak menggunakan kata yang telah dipakai oleh compiler (reserve Word)
- Pertimbangkan nama yang tepat untuk class. Jangan gunakan nama acak dan singkat seperti XYZ.
- Nama file dari class harus sama dengan nama public class

Class Body merupakan bagian dari kelas yang mendeklarasikan kode-kode program java. Class Body tersusun atas:

- Konstruktor
- Variabel
- Method (Fungsi-fungsi atau prosedur)

Konstruktor adalah suatu method yang mempunyai nama sama dengan nama class dimana method itu dibuat. Fungsi konstruktor adalah sebagai inisiasi awal dari terbentuknya sebuah objek.

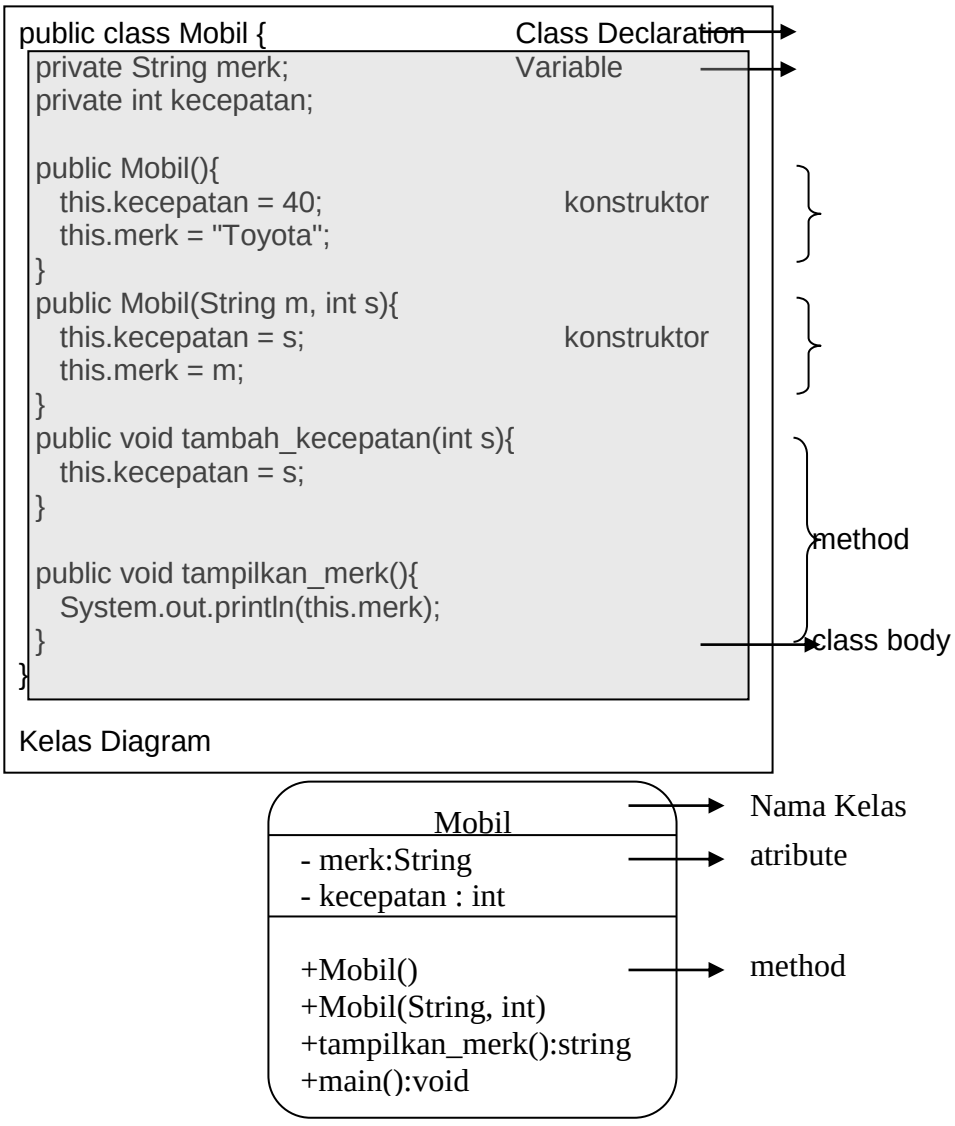
Petunjuk Penulisan Program :

1. Deklarasikan seluruh instance variable pada awal penulisan class
2. Deklarasikan variable per baris
3. Penulisan instance variable, termasuk juga variabel lain harus dimulai dengan huruf kecil
4. Gunakan tipe data yang tepat pada setiap variabel
5. Deklarasikan instance variable sebagai private supaya hanya method pada class itu sendiri yang dapat mengaksesnya.

Program 1.

```
public class Sederhana {
public static void main(String[] args) {      method main
  System.out.println("Hello …. Ini Program Pertamaku");
 }
}
```

Program 2.



Membuat Objek

Objek merupakan bagian dinamis dari suatu class. Class hanyalah sebuah prototype penggunaan class tersebut tidak bias langsung tetapi harus dibuatkan objek terlebih dahulu. Sebagai contoh kita ambil kelas Mobil diatas kita akan membuat objek Mobil dengan nama mobilA dan mobilB, mobilA dengan merk karimun sedangkan mobilB merk kijang, maka pembuatan objeknya / *instansiasi object* dari kelas Mobil adalah:

```
public class Mobil {
```

```
  private String merk;
  private int kecepatan;

  public Mobil(){
     this.kecepatan = 60;
     this.merk = "Karimun";
  }
  public Mobil(String m, int s){
     this.kecepatan = s;
     this.merk = m;
  }
  public void tambah_kecepatan(int s){
     this.kecepatan = s;
  }
  public void tampilkan_merk(){
     System.out.println(this.merk);
  }

  public static void main(String[] args) {
    Mobil mobilA = new Mobil();
    Mobil mobilB = new Mobil("Kijang",60);
    Mobil mobilC = new Mobil("Avanza",40);
    Mobil mobilD = new Mobil("Jazz",80);
  }
}
```

Praktik 3. Mengunakan objek dari kelas

```
  public static void main(String[] args) {
    Mobil mobilA = new Mobil();
    Mobil mobilB = new Mobil("Kijang",60);
    Mobil mobilC = new Mobil("Avanza",40);
    Mobil mobilD = new Mobil("Jazz",80);
    mobilA.tampilkan_merk();
    mobilB.tampilkan_merk();
    mobilC.tampilkan_merk();
    mobilD.tampilkan_merk();
  }
```

3. LATIHAN

1. Pada Program mobil tambahkan method menampilkan kecepatan mobil
2. Tampilkan kecepatan masing-masing objek mobil yang telah diinstansiasi
3. Buatlah objek baru dari kelas mobil dan tampilkan merk dan kecepatannya
4. Terdapat program sebagai berikut

```
import java.util.Scanner;
public class Person {
private String nama;
private String alamat;
private int umur;
public Person(){
    this.nama = "";
```

```
    this.alamat = "";
    this.umur = 0;
  }
  public Person(String n, String a, int u){
    this.nama = n;
    this.alamat = a;
    this.umur = u;
  }
  public void info() {
     System.out.println("Nama = "+this.nama);
     System.out.println("Alamat = "+this.alamat);
     System.out.println("Umur = "+this.umur);
   }
  public static void main(String[] args) {
    Scanner masuk = new Scanner(System.in);
    Person siA = new Person();
    String nm;
    System.out.print("Masukkan Nama   :");
    nm = masuk.nextLine();
    String alamat;
    System.out.print("Masukkan Alamat :");
    alamat = masuk.nextLine();
    int umur;
    System.out.print("Masukkan Nama :");
    umur = masuk.nextInt();
    Person siB = new Person(nm,alamat,umur);
    siA.info();
    siB.info();
   }
}
```

Buat program diatas

(Dibuat pada laporan)

- Identifikasi atributenya?
- Konstruktornya ?
- method yang ada pada program tersebut?
- Objek yang dibuat
- Gambarkan diagram kelasnya

MODUL II
METHOD

1. TUJUAN

- Menjelaskan struktur method
- menyebutkan dan membuat berbagai method

2. TEORI DASAR

(*Object Oriented Programming*) OOP berputar pada konsep dari obyek yang merupakan elemen dasar dari program Anda. Ketika kita membandingkan dengan dunia nyata, kita dapat menemukan beberapa obyek disekitar kita seperti mobil, singa, manusia dan seterusnya. Obyek ini dikarakterisasi oleh atribut dan tingkah lakunya.

Objek	Atribut	Tingkah Laku
Mobil	Tipe dari transmisi Manufaktur warna	Berbelok Mengerem Mempercepat
Singa	Berat Warna Lapat atau tidak lapar Jinak atau liar	Mengeram Tidur Berburu

Dengan deskripsi ini, obyek pada dunia nyata dapat secara mudah asumsikan sebagai obyek perangkat lunak menggunakan atribut sebagai data dan tingkah laku sebagai method. Data dan method dapat digunakan dalam pemrograman game atau perangkat lunak interaktif untuk membuat simulasi obyek pada dunia nyata.

Sebuah method adalah bagian-bagian kode yang dapat dipanggil oleh program utama atau dari method lainnya untuk menjalankan fungsi yang spesifik.
Berikut adalah karakteristik dari method :

1. dapat mengembalikan satu nilai atau tidak sama sekali
2. dapat diterima beberapa parameter yang dibutuhkan atau tidak ada parameter sama sekali. Parameter bisa juga disebut sebagai argumen dari fungsi
3. setelah method telah selesai dieksekusi, dia akan kembali pada method yang memanggilnya.

Untuk memanggil sebuah instance method, kita dapat menuliskan :

nameOfObject.nameOfMethod(parameters);

mari kita mengambil dua contoh method yang ditemukan dalam class String.

Tabel : Contoh Metode dari kelas String

Deklarasi metode	Definisi
public char charAt(int index)	Mengambil karakter pada indeks tertentu
public boolean equalsIgnoreCase (String anotherString)	Membandingkan antar String, tidak case sensitive

Deklarasi *Methods*

1. *Accessor Methods*

   Digunakan untuk membaca nilai variabel pada *class*, baik berupa *instance* maupun *static*. Sebuah *accessor method* umumnya dimulai dengan penulisan

   get<namaInstanceVariable>.

   Method ini juga mempunyai sebuah *return value*.

2. *Mutator Methods*

   method yang dapat memberi atau mengubah nilai variable dalam *class*, baik itu berupa *instance* maupun *static*. *Method* semacam ini disebut dengan *mutator methods*. Sebuah *mutator method* umumnya tertulis

   set<namaInstanceVariabel>.

3. *Multiple Return Statements*

   Anda dapat mempunyai banyak *return values* pada sebuah *method* selama mereka tidak pada blok program yang sama. Anda juga dapat menggunakan konstanta disamping variabel sebagai *return value*.

4. Method Static

   Method Static adalah method yang dapat dipakai tanpa harus menginisialisasi suatu class (maksudnya tanpa menggunakan variabel terlebih dahulu). Method static hanya dimiliki oleh class dan tidak dapat digunakan oleh instance (atau objek) dari suatu class. Method static dibedakan dari method yang dapat instance di dalam suatu class oleh kata kunci static. Untuk memanggil method static, ketik :

   Classname.staticMethodName(params);

5. Reference *this*

   Reference *this* digunakan untuk mengakses instance variable yang dibiaskan oleh parameter.

6. *Overloading Methods*

Overloading method mengijinkan sebuah *method* dengan nama yang sama namun memiliki parameter yang berbeda sehingga mempunyai implementasi dan *return value* yang berbeda pula.

overloaded method memiliki *property* sebagai berikut :

1. Nama yang sama
2. Parameter yang berbeda
3. Nilai kembalian (*return*) bisa sama ataupun berbeda

7. Constructor

Constructor adalah *method* dimana seluruh inisialisasi *object* ditempatkan. Berikut ini adalah *property* dari Constructor :

1. Constructor memiliki nama yang sama dengan class
2. Constructor tidak memiliki return value
3. Constructor tidak dapat dipanggil secara langsung, namun harus dipanggil dengan menggunakan operator ***new*** pada pembentukan sebuah *class*.

Contoh Kode Membuat *Accessor Methods*

```
public class Method1 {
private int umur = 18;
private String nama="Galih";
/*metode Asesor*/
public int  getUmur(){
  return umur;
  }
public String getNama(){
  return nama;
  }
//metode main
public static void main(String args[]){
  Method1 m1 = new Method1();
  System.out.println(m1.getUmur());
  System.out.println(m1.getNama());
  }
}
```

Contoh Kode Membuat *Mutator Methods*

```
public class Method2 {
private int umur = 18;
private String nama="Galih";
/*metode Mutator*/
public void setUmur(int temp){
   umur = temp;
```

```
}
/*metode Acesor*/
public int  getUmur(){
  return umur;
  }
/*metode Mutator*/
public void setNama(String temp){
   nama = temp;
}
/*metode Acesor*/
public String getNama(){
  return nama;
  }
//metode main
public static void main(String args[]){
  Method2 m1 = new Method2();
  System.out.println(m1.getUmur());
  System.out.println(m1.getNama());
  m1.setUmur(20);
  m1.setNama("Rizky");
  System.out.println(m1.getUmur());
  System.out.println(m1.getNama());
 }
}
```

Contoh Kode *Multiple Return Statements*

```
public class Method3 {
public String getNumberInWords(int num) {
  String defaultNum = "Zero";
  if (num == 1) {
    return "one";
  }
  else if (num == 2) {
    return "two";
  }
  return defaultNum;
}
public static void main(String args[]) {
  Method3 m1 = new Method3();
  System.out.println(m1.getNumberInWords(2));
  }
}
```

Contoh Kode Membuat Method Static

```
public class Method4 {

public int luasKotak(int p, int l){
   return p * l;
  }
public static void main(String args[]){
  System.out.println(luasKotak(10,5));
}
}
```

→ Jalankan Program Method4
→ Modifikasi program seperti di bawah

```
public class Method4 {
public static int luasKotak(int p, int l){
   return p * l;
  }
public static void main(String args[]){
  System.out.println(luasKotak(10,5));
}
}
```

Contoh Kode Penggunaan Reference *this*

```
public class Method5 {
private int umur;
private String nama;
/*metode Mutator*/
public void setUmur(int umur){
   this.umur = umur;
}
/*metode Mutator*/
public void setNama(String nama){
   this.nama = nama;
}
public void info(){
  System.out.println("Nama : "+nama);
  System.out.println("Umur : "+umur);
  }
//metode main
public static void main(String args[]){
  Method5 m5 = new Method5();
  m5.info();
  m5.setNama("Nana");
  m5.setUmur(18);
  m5.info();
 }
}
```

Contoh Kode Membuat *Overloading Methods*

```
public class Method6 {
public double luas(double p, double l){
   return 0.5* p * l;
  }
public int luas(int p, int l){
   return p * l;
  }
public int luas(int s){
   return s * s;
  }
public static void main(String args[]){
  Method6 m6= new Method6();
  System.out.println("Luas segitiga");
```

```
  System.out.println(m6.luas(2.5,3.5));
  System.out.println("Luas persegi panjang");
  System.out.println(m6.luas(5,3));
  System.out.println("Luas bujursangkar");
  System.out.println(m6.luas(5));
 }
}
```

Contoh Kode Constructor

```
public class Method7 {
private int umur;
private String nama;
//konstruktor
public Method7(){
   umur = 0;
   nama = "";
}
//konstruktor  -->untuk inisiasi
public Method7(int umur, String nama){
   this.umur = umur;
   this.nama = nama;
}
public void info(){
  System.out.println("Nama : "+nama);
  System.out.println("Umur : "+umur);
  }
//metode main
public static void main(String args[]){
  Method7 m1 = new Method7();
  m1.info();
  Method7 m2 = new Method7(20,"Surya");
  m2.info();
 }
}
```

4. Latihan

Buatlah Program kelas mahasiswa yang mempunyai

Atribute : nomhs, nama, jurusan, ipkum

Methode : info untuk menampilkan data mahasiswa

Lengkapi program dengan konstruktor, metode asesor dan metode mutator

MODUL 3
PACKAGE

1. TUJUAN

Mahasiswa mampu dan memahami cara penggunaan dan membuat Package

2. TEORI DASAR

Seperti yang diketahui, Java adalah pemrograman dengan menggunakan banyak kelas. Kelas-kelas tersebut dikelompokkan ke dalam kategori tertentu yang berhubungan disebut dengan *packages* (paket) . Misalnya saja paket javax.swing.* berarti semua kelas yang berhubungan dengan javax swing berada dalam direktori tersebut.

Langkah-langkah untuk membuat kelas yang *reusable* adalah sebagai berkut:

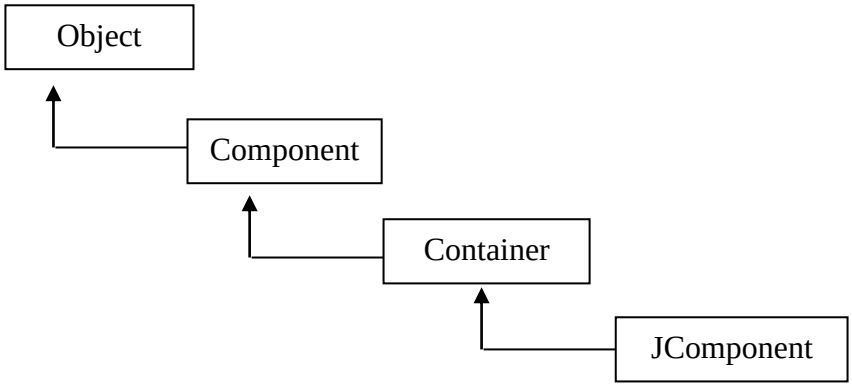
1. Deklarasikan *public class*. Jika kelas tidak bersifat public, dia hanya bisa dipergunakan oleh kelas lain di dalam paket yang sama.
2. Pilih nama paket, dan tambahkan sebuah *package declaration* ke file *source code* untuk deklarasi kelas *reusable*. Hanya bisa ada satu deklarasi paket saja di dalam sebuah file *source code* Java dan dia harus mendahului semua deklarasi dan pernyataan lain di dalam file tersebut.
3. Kompilasi kelas tersebut sehingga dia ditempatkan di dalam struktur direktori paket yang sesuai.
4. Import *reusable class* ke dalam sebuah program, dan gunakan kelas tersebut.

Fitur ini menyediakan mekanisme untuk mengatur class dan interface dalam jumlah banyak dan menghindari konflik pada penamaan.

Menggunakan Salah satu Paket Miliknya Java : Komponen Swing

Komponen	Deskripsi
Jlabel	Area dimana teks yang tidak bisa diedit atau sebuah icon dapat ditampilkan
JTextField	Area dimana user menginputkan data dari keyboard. Area ini juga bisa menampilkan informasi
JButton	Area yang memicu sebuah event ketika diklik dengan mouse
JCheckBox	Komponen GUI yang bisa dipilih atau tidak dipilih
JComboBox	Sebuah drop-down list dari item dari mana user dapat membuat pilihan dengan meng-klik sebuah item dalam list atau mungkin dengan menuliskan dalam box
JList	Area yang berisi sebuah list dari item dari mana user dapat membuat pilihan dengan

	meng-klik pada beberapa elemen dalam list. Beberapa elemen dapat dipilih
JPanel	Sebuah kontainer dalam mana komponen dapat ditempatkan atau diorganisasi



Gambar 1. Superclass untuk banyak komponen Swing

Gambar 1 memperlihatkan sebuah hirarki inheritance yang berisi class yang mendeklarasikan atribut dan lingkungan yang berhubungan dengan banyak komponen Swing. Class object adalah superclass dari hirarki class Java. Class Component (package java.awt) adalah subclass dari Object, class Container (package java.awt) adalah subclass dari Component, dan class Jcomponent (package javax.swing) adalah subclass dari Container.

Struktur Package

```
package nm_package;
import registration.processing.*;
import java.util.List;
import java.lang.*; //imported by default
class NmClass {
  /* details of NmClass */
}
```

3. LATIHAN

1. Membuat sebuah paket sederhana

Tuliskan program berikut dalam folder mahasiswa yang berada di dalam direktori kerja anda. Jika belum ada buat dulu.

```
package mahasiswa;
public class paket
{
        public static void isiPaket(){
         System.out.println("ini hasil import");
```

```
    }
}
```

Simpan dengan nama paket.java pada direktori mahasiswa.

4. Menggunakan paket hasil buatan sendiri. Cobalah program berikut. Simpan dalam direktori kerja anda (di luar folder mahasiswa).

```
import mahasiswa.*;
public class cobaPaket
{
  public static void main(String args[])
  {
        paket coba=new paket();
        coba.isiPaket();
  }
 }
```

5. Masukkan method-method tentang mahasiswa yang anda buat pada modul sebelumnya di dalam paket mahasiswa. Panggil method tersebut dari class cobaPaket.
6. Menggunakan paket miliknya java. Cobalah program berikut yang menggunakan paket bawaannya java.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LabelTest extends JFrame {
private JLabel label;

public LabelTest()
{
 super( "Mencoba JLabel" );
 Container container = getContentPane();
 container.setLayout( new FlowLayout() );
 label = new JLabel( "Label dengan text" );
 label.setToolTipText( "Ini adalah label1" );
 container.add( label );
 setSize( 500, 400 );
 setVisible( true );
}

public static void main( String args[] )
{
 LabelTest aplikasi = new LabelTest();
 aplikasi.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

}
```

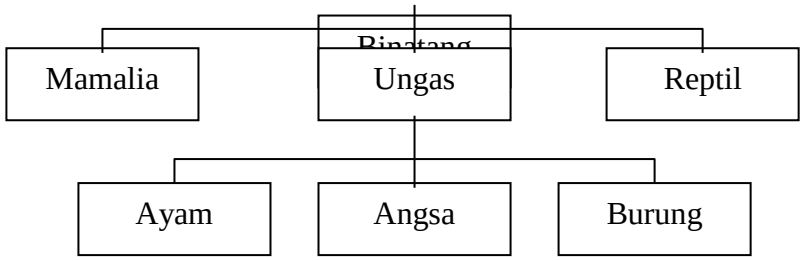
MODUL IV
PEWARISAN

1. TUJUAN

- Mahasiswa Mampu mendefinisikan *superclasses* dan *subclasses*
- Mahasiswa Mampu membuat *superclasses* dan *subclasses*
- Mahasiswa Mampu menjelaskan dan menggunakan acces modifier
- Mahasiswa Mampu menjelaskan *Overriding* method dari *superclasses*

2. TEORI DASAR

Sebagai manusia kita sebenarnya terbiasa untuk melihat objek yang berada disekitar kita tersusun secara hierarki berdasarkan class-nya masing-masing. Dari sini kemudian timbul suatu konsep tentang pewarisan yang merupakan suatu proses dimana suatu class diturunkan dari class lainnya sehingga ia mendapatkkan ciri atau sifat dari class tersebut. Perhatikan contoh hirarki berikut ini:



Dari hirarki diatas dapat dilihat bahwa, semakin ke bawah, class akan semakin bersifat spesifik. Class Ungas memiliki seluruh sifat yang dimiliki oleh binatang, demikian halnya juga Ayam, Angsa dan Burung memiliki seluruh sifat yang diturunkan dari class Ungas. Dengan konsep ini, karakteristik yang dimiliki oleh class binatang cukup didefinisikan didefinisikan dalam class binatang saja. Class Ungsa tidak perlu mendefinisikan ulang apa yang telah dimiliki oleh class binatang, karena sebagai class turunannya, ia akan mendapatkan karakteristik dari class binatang secara otomatis. Demikian juga dengan class anjing, kucing dan monyet, hanya perlu mendefinisikan karakteristik yang spesifik dimiliki oleh class-nya masing-masing. Dengan memanfaatkan konsep pewarisan ini dalam pemrograman, maka hanya perlu mendefinisikan karakteristik yang lebih umum akan didapatkan dari class darimana ia diturunkan.

Beberapa class di atas class utama dalam hirarki class dikenal sebagai superclass. Sementara beberapa class di bawah class pokok dalam hirarki class dikenal sebagai subclass dari class tersebut.

Pewarisan adalah keuntungan besar dalam pemrograman berbasis object karena suatu sifat atau method didefinisikan dalam *superclass*, sifat ini secara otomatis diwariskan dari semua *subclasses*. Jadi, Anda dapat menuliskan kode method hanya sekali dan mereka dapat digunakan oleh semua subclass. Subclass hanya perlu mengimplementasikan perbedaannya sendiri dan induknya.

Access modifiers digunakan membuat, mengatur *properties* dan *class methods*, untuk mengimplementasikan beberapa macam larangan untuk mengakses data. macam *access modifiers* di JAVA, yaitu : *public, private, protected* dan *default*.
3 tipe akses pertama tertulis secara ekplisit pada kode untuk mengindikasikan tipe akses, sedangkan yang keempat yang merupakan tipe *default*, tidak diperlukan penulisan *keyword*
atas tipe.

Tabel. *Access Modifiers*

	Private	**default/package**	**protected**	**Public**
Same class	Yes	Yes	Yes	Yes
Same package		yes	Yes	Yes
Different package (subclass)			Yes	Yes
Different package				Yes

Contoh Mendefinisikan Superclass dan Subclass
Untuk memperoleh suatu class, kita menggunakan kata kunci **extend**.
Membuat class anak atau subkelas:

- Menggunakan *extends* dalam mendeklarasikan class
- Sintaks:

  class <childClassName> extends <parentClassName>
- Sebuah class hanya dapat meng-*extend* satu class induk

Buat class induk/superclass dibawah ini

```
public class Person {
protected String nama;
protected String alamat;
```

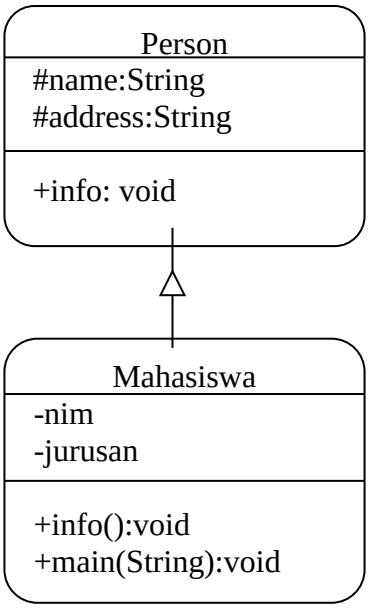
```
        public Person(){
                System.out.println("Inside Person:Constructor");
                nama = "";
                alamat = "";
                }
        public Person( String nama, String alamat ){
                this.nama = nama;
                this.alamat = alamat;
                }
        public String getNama(){
                return nama;
                }
        public String getAlamat(){
                return alamat;
                }
        public void setNama( String nama ){
                this.nama = nama;
                }
        public void setAlamat( String almt ){
                this.address = almt;
                }
        }
```

Contoh Kode Buat subclass/ Class Child dibawah ini

```
        public class Mahasiswa extends Person{
        private int nim;
        private String jurusan;
        public Mahasiswa(){
         }

        public Mahasiswa(int nim, String nm,
         String jurusan, String almt){
           this.nim = nim;
           this.jurusan = jurusan;
           nama = nm;
           alamat = almt;
           }

        public static void main(String[] args) {
           Mahasiswa mhs = new Mahasiswa();
           System.out.println("Nim     = "+ mhs.nim);
           System.out.println("Nama    = "+ mhs.nama);
           System.out.println("Jurusan = "+ mhs.jurusan);
           System.out.println("Alamat  = "+ mhs.alamat);
           Mahasiswa mahasiswa1 = new Mahasiswa(1,"lulu","SI","Janti");
           System.out.println("Nim     = "+ mhs1.nim);
           System.out.println("Nama    = "+ mhs1.nama);
           System.out.println("Jurusan = "+ mhs1.jurusan);
           System.out.println("Alamat  = "+ mhs1.alamat);
         }
  }
```



Contoh Kode Membuat Method Overriding

Untuk beberapa pertimbangan, terkadang class asal perlu mempunyai implementasi berbeda dari method yang khusus dari *superclass* tersebut. Oleh karena itulah, method overriding digunakan jika kelas child perlu mempunyai implementasi berbeda dari method yang khusus dari superclass

Modifikasi Program Class Person dengan penambahan methode info sebagai berikut :

```
public class Person {
protected String nama;
protected String alamat;
.
.
.
public void setAddress( String add ){
this.address = add;
}
public void info(){
System.out.println("Nama    = "+ nama);
System.out.println("Alamat  = "+ alamat);
}
public static void main(String[] args) {
Person  p1 = new Person("Agus","Janti");
p1.info();
}
}
```

Compile dan Jalankan program di atas

Modifikasi Program Class Mahasiswa dengan penambahan method info sebagai berikut :

```
public class Mahasiswa extends Person{
private int nim;
private String jurusan;
public Mahasiswa(){
 }

public Mahasiswa(int nim, String nama,
 String jurusan, String alamat){
   this.nim = nim;
   this.jurusan = jurusan;
   super.nama = nama;
   super.alamat = alamat;
   }
public void info(){
   System.out.println("Nim     = "+ nim);
   System.out.println("Nama    = "+ name);
   System.out.println("Jurusan = "+ jurusan);
   System.out.println("Alamat  = "+ alamat);
  }

public static void main(String[] args) {
   Mahasiswa mhs = new
    Mahasiswa(1,"lulu","SI","Janti");
 mhs.info();
}
}
```

Contoh Kode Urutan Eksekusi Konstruktor

Subkelas memiliki semua yang dimiliki oleh superkelas maka subkelas juga memiliki konstruktor yang dimiliki oleh superkelasnya.

```
//class Pewarisan.Java
class A {
A() {
    System.out.println("konstruktor class A dieksekusi...");
 }
}

class B extends A {
B() {
    System.out.println("konstruktor class B dieksekusi...");
 }
}

class C extends B {
C() {
    System.out.println("konstruktor class C dieksekusi...");
```

```
 }
}

public class Pewarisan {
    public static void main(String[] args) {
        C subOb = new C();
    }
}
```

4. LATIHAN

1. Pada program Person ganti akses modifier untuk variabel name dan address menjadi private, Compile kelas Person kemudian compile dan kelas mahasiswa, apa yang terjadi jelaskan pada laporan
2. Bagaimana cara kelas mahasiswa mengakses dan menganti data untuk variabel nama dan alamat jika variabel dan alamat pada kelas Person berakses modifier private
3. Buatlah kelas Pegawai yang merupakan turunan dari kelas Person.
   Pegawai memiliki atribute NIP, nama, alamat, departemen, gaji.
   Metode : menampilkan informasi pegawai

MODUL V
POLYMORPHISME

1. TUJUAN

- Mahasiswa memahami konsep polymorphisme
- membuat program dengan konsep polymorphisme

2. TEORI DASAR

Kata ***polimorfisme*** yang berarti satu objek dengan banyak bentuk yang berbeda, adalah konsep sederhana dalam bahasa pemrograman berorientasi objek yang berarti kemampuan dari suatu variabel referensi objek untuk memiliki aksi berbeda bila method yang sama dipanggil, dimana aksi method tergantung dari tipe objeknya. Kondisi yang harus dipenuhi supaya ***polimorfisme*** dapat diimplementasikan adalah :

- Method yang dipanggil harus melalui variabel dari basis class atau superclass.
- Method yang dipanggil harus juga menjadi method dari basis class.
- Signature method harus sama baik pada superclass maupun subclass.
- Method access attribute pada subclass tidak boleh lebih terbatas dari basis class.

Ada dua bentuk polymorphism

1. Overriding

   Override merupakan pendefinisian ulang suatu metode oleh subclass. Syarat Override yaitu metode, return type, dan parameter harus sama. Jika tidak sama maka bukan dianggap sebagai override tetapi metode yang baru pada subclass.

2. Overloading

   Overload merupakan pendefinisian ulang suatu metode dalam class yang sama. Syarat overload yaitu metode dan tipe parameter harus berbeda dalam kelass yang sama.

Contoh Kode **Polimorphisme bentuk override**

```
class Binatang {
public void info() {
   System.out.println("  Info tentang Hewan : ");
   }
}

class Herbivora extends Binatang {
public void info() {
```

```
    System.out.println ("Info pada herbivora: Memakan makanan berupa tumbuh
– tumbuhan");
    }
}
class Kelinci extends Herbivora
{
public void info() {
    System.out.println("Info pada Kelinci: Memakan makanan berupa wortel");
    }
}

public class Polimorphisme {
public static void main(String[] args){
    Herbivora herbivora;
    Kelinci kelinciku;
    Binatang hewan = new Binatang();
    herbivora=new Herbivora();
    kelinciku=new Kelinci();
    hewan.info();
    hewan=herbivora;
    hewan.info();
    hewan=kelinciku;
    hewan.info();
}
}
```

3. LATIHAN

1. Buatlah program Polimorphisme dengan konsep Overloading
2. Jelaskan konsep polimorphisme dari program yang anda buat

MODUL VI
EXCEPTION HANDLING

1. TUJUAN

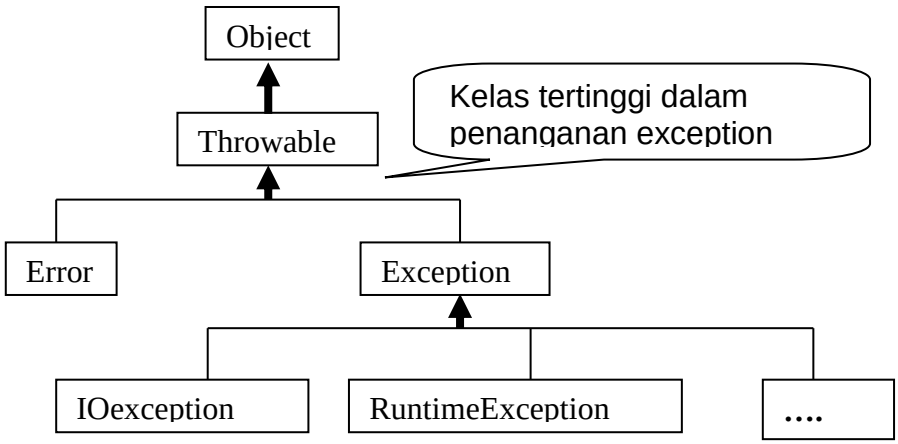
- Mahasiswa mengenal kelas – kelas yang menangani exception.
- Mahasiswa mampu menangani exception dengan menggunakan try, catch
- Mahasiswa mampu menangani exception dengan menggunakan try,catch dan finally

2. TEORI DASAR

Kesalahan merupakan bagian normal dari pemrograman. Beberapa kesalahan merupakan kelemahan perancangan atau implementasi, jenis kesalahan ini disebut bug. Kesalahan kedua bukan merupakan bug tapi karena hasil suatu situasi/kondisi/lingkungan seperti munculnya memori habis atau pengaksesan array diluar jangkauan array tersebut. Kondisi abnormal ketika program sudah dijalankan disebut exception.

Seluruh exceptions adalah subclasses, baik secara langsung maupun tidak langsung,dari sebuah root class *Throwable*. Kemudian, dalam class ini terdapat dua kategori umum : *Error class* dan *Exception class*

Berikut ini disajikan struktur hirarki dari exception dalam java



Untuk menangani exception di dalam java digunakan pernyataan *try, catch,* ataupun bisa ditambahkan pernyataan *finally* seperti dibawah ini:

```
try { }
catch (<exception Type> <varName>){
 }
...
```

catch (<exception Type> <varName>){

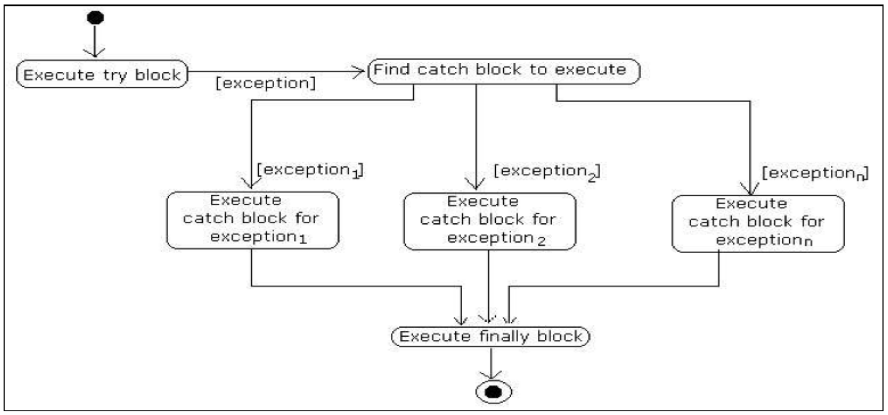
}

finally{

}

Beberapa hal yang perlu diperhatikan ketika melakukan pemrograman dengn menggunakan exception handling adalah sebagai berikut::

- Wajib membuat notasi blok
- Setiap blok try boleh memiliki lebih dari satu blok catch dan hanya boleh memiliki satu blok finally
- Blok catch dan blok finally harus muncul bersama blok try
- Blok try harus diikuti minimal satu blok catch, atau satu blok finally, atau kedua blok catch dan finally



Gambar alur kerja try,catch,finally

Contoh Program 1

```
public class LatihanException1 {
  public static void main(String[ ] args) {
        int x = 2;
       int y = x - 2;
      int z = x / y;     //penyebab error
     System.out.println("Nilai z = " + z);
   }
  }
```

Pada program diatas ketika dijalankan akan muncul error *Exception in thread "main" java.lang.Arithı* karena nilai X dibagi dengan 0

Program 2: menggunakan exception

```
public class LatihanException2 {
    public static void main(String[] args) {
     int x = 2;
     int y = x - 2;
```

```
    try{
        int z = x / y;
        System.out.println("Nilai z = " + z);
    }
    catch (ArithmeticException e) {
        System.out.print("Terdapat Kesalahan: ");
        System.out.println(e.getMessage());
     }
   }
 }
```

Pada program no.2 diatas, tidak terjadi error disebabkan kesalahan yang disebabkan oleh pembagian suatu bilangan dengan 0, sudah ditanggani oleh class *ArithmeticException*.

Program 3 : menggunakan finally

```
public class ExceptionFinally {
public static void main( String[ ] args ){
  int ages[]=new int[5];
        try{  System.out.println( ages[7] ); }
        catch( ArrayIndexOutOfBoundsException exp ){
        System.out.println("Array Tidak bisa diakses");
        }

        finally{
         System.out.println("Sudah dilakukan pengecekan");
  }
```

Pada program diatas menggunakan try, catch serta pernyataan finally. Setelah dilakukan pengujian pada blok try , bila ada kesalahan maka akan ditangkap pada bagian blok catch dan kemudian blok finally selanjutnya diekseskusi. Sedangkan bila tidak ada kesalahan pada blok try, selanjutnya langsung mengekseskusi blok finally. Jadi blok finally akan selalu dieksekusi dibagian akhir

3. LATIHAN

- Buatlah program yang hanya bisa menerima input berupa angka saja. Bila diinputkan selain angka (numeric) maka akan ditangkap oleh blok catch
- Berdasarkan program diatas tambahkan blok finally untuk menyatakan bahwa telah dilakukan pengecekan terhadap data yang diinputkan

MODUL VII
KELAS ABSTRAK & INTERFACE

1. Tujuan

- Mahasiswa mampu mengenal konsep kelas abstrak dan interface
- Mahasiswa mampu membuat kelas abstrak & mampu membuat kelas interface

2. TEORI DASAR

Abstrak secara umum dapat kita artikan sebagai gambaran tentang sesuatu hal namun belum begitu jelas gambarannya. Sehingga masih butuh suatu proses lagi supaya gambaran tentang sesuatu tersebut menjadi lebih jelas(konkrit). kelas Abstrak tidak berbeda dengan kelas-kelas lainnya yaitu memiliki class members (method dan variabel). Sebuah class adalah abstrak jika salah satu *method*nya dideklarasikan abstract. Method abstrak adalah method yang ditentukan dari dalam kelas tapi tidak disertai definisinya . Beberapa hal yang perlu diperhatikan adalah sebagai berikut:

- Class abstrak tidak dapat dibuatkan instant atau objeknya menggunakan *keyword new.*
- Sebuah class dapat dideklarasikan sebagai class abstrak walaupun tidak memiliki method abstrak.

Definisi kelas diletakkan pada masing – masing kelas turunan (*subclass*). Dalam hal ini setiap subclass dari kelas abstrak harus mendefinisikan method-method yang tergolong sebagai method abstrak. Untuk membuat kelas abstrak digunakan key word abstract dengan cara penulisan seperti berikut:

```
abstract public <nama method>{
  //variabel instant
 //method non abstrak
 //method abstrak
}
```

Contoh dari kelas abstrak adalah obyek bangun 2 dimensi (2D). Semua obyek bangun 2D dapat dihitung luas dan kelilingnya, tapi karena rumus untuk mendapatkan luas dan kelilingnya berbeda-beda untuk tiap – tiap bangun 2D maka method untuk menghitung luas dan keliling dibuat sebagai abstrak. Pada subclass – subclassnya baru akan diimplementasikan secara detail disertai penulisan rumus untuk masing – masing bangun 2D.

Selain kelas abstrak terdapat pula kelas *interface*. Kelas ini sering disebut *pure abstract*. Sebuah interface dapat dibayangkan sebagai kelas abstrak yang seluruh methodnya adalah abstrak. Interface digunakan pada kelas-kelas lain dengan cara diimplementasikan dengan menggunakan *key word implements*. Sedangkan untuk membentuknya menggunakan *key word interface* dengan aturan seperti berikut

```
<modifier> interface <nama interface> {
        //definisi konstanta
        // method abstrak
}
```

Interface dapat dibedakan dengan kelas abstrak dalam hal seperti berikut :

- kelas abstrak dapat memiliki method abstrak maupun non abstrak tapi interface harus berisi method abstrak semua
- kelas abstrak dapat mendeklarasaikan variabel instant, sedangkan interface hanya dapat mendefinisikan konstanta
- interface dapat diimplementasikan lebih dari dari satu interface
- pada interface modifier yang digunakan hanya public atau tidak sama sekali

Berikut ini adalah contoh program untuk membuat kelas abstrak dengan kasus bangun 2D seperti contoh pada teori diatas.

- Simpan file dengan nama BangunDatar.java

```
public abstract class BangunDatar {
    public abstract double hitungLuas();
    public abstract double hitungKeliling();
}
```

- Simpan file dengan nama BujurSangkar.java

```
public class BujurSangkar extends BangunDatar {
    private double sisi;
    public BujurSangkar(double sisi) {
        this.sisi = sisi;
    }
    public double hitungLuas() {
        return sisi * sisi;
    }
    public double hitungKeliling() {
        return 4 * sisi;
    }
}
```

- Simpan file dengan nama AbstractSample.java

```
public class AbstractSample {
    public static void main(String[] args) {
        BangunDatar obyek1 = new BujurSangkar(10);
```

```
        System.out.println("Luas bujur sangkar dengan sisi 10 = " +
        Obyek1.hitungLuas());
        System.out.println("Kelilingnya = " +
        Obyek1.hitungKeliling());
    }
}
```

Interface

Untuk mempercepat proses gunakan file dari praktek abstrak dan modifikasi.

- Simpan file dengan nama InterfaceBangunDatar.java

```
public interface InterfaceBangunDatar {
    static final String JENIS="Bangun Datar 2D";
    public abstract double hitungLuas();
    public abstract double hitungKeliling();
}
```

- Simpan file dengan nama InterfaceBujurSangkar.java

```
public class InterfaceBujurSangkar implements InterfaceBangunDatar {
    private double sisi;
    public String getJenis(){
       return JENIS;
    }

    public InterfaceBujurSangkar(double sisi) {
        this.sisi = sisi;
    }
    public double hitungLuas() {
        return sisi * sisi;
    }

    public double hitungKeliling() {
        return 4 * sisi;
    }
}
```

- Simpan file dengan nama InterfaceSample.java

```
public class InterfaceSample{
    public static void main(String[] args) {
       InterfaceBujurSangkar obyek1 = new InterfaceBujurSangkar(10);
       System.out.println("Jenisnya adalah: "+
       obyek1.getJenis());
       System.out.println("Luas bujur sangkar dengan sisi 10 = " +
       obyek1.hitungLuas());
       System.out.println("Kelilingnya = " +
       obyek1.hitungKeliling());
    }
}
```

3. LATIHAN

- Modifikasi program berdasarkan praktek abstrak. Tambahkan sebuah kelas yang berisi definisi untuk menghitung luas dan keliling lingkaran,simpan dengan nama file “lingkaran.java“. Kemudian buat obyek lingkaran pada method main() dan tampilkan hasilnya.
- Modifikasi program berdasarkan praktek interface. Tambahkan sebuah kelas yang berisi definisi untuk menghitung luas dan keliling segitiga,simpan dengan nama file “InterfaceSegitiga.java“. Kemudian pada method main() buat obyek segitiga dan tampilkan hasilnya.

MODUL VII
THREAD

1. TUJUAN

- Mahasiswa mampu mendefinisikan thread dari class thread
- Mahasiswa mampu mendefinisikan thread dari interface runnable
- Mahasiswa mampu melakukan sinkronisai beberapa thread

2. TEORI DASAR

Thread adalah unit terkecil dari eksekusi suatu program. *Thread* mengeksekusi rangkaian instruksi satu demi satu. Instruksi-instruksi dalam program akan dieksekusi oleh *thread* ini secara berantai, satu demi satu dari awal hingga akhir.

Dalam sistem komputer modern, beberapa *thread* bisa tercipta dalam satu waktu. Pada satu saat tertentu, hanya ada satu *thread* yang bisa dijalankan, karena CPU hanya bisa melakukan satu hal dalam satu waktu. Pada komputer dengan multiprosesor, multicore, dan hyper-*thread*ing, masing-masing prosesor atau core melakukan *thread* yang berbeda-beda. Akan tetapi sebenarnya komputer membagi waktu menjadi bagian-bagian kecil sehingga seolah-olah seluruh *thread* dijalankan secara bersama-sama. Pembagian waktu berarti CPU mengeksekusi suatu *thread* dalam kurun waktu tertentu, setelah itu beralih mengeksekusi *thread* yang lain, kemudian *thread* lain, dan seterusnya dan kemudian kembali ke *thread* pertama. Urutan dalam menjalankan *Thread* ini bersifat *non-deterministik(*tidak bisa ditentukan). *Thread* pada java ditangani melalui dua mekanisme:

1. Membuat kelas turunan dari *Thread* dan meng*override* method *run()*.
2. Mengimplementasikan interface *Runnable* dan mengimplementasikan methoda *run( )*.

Beberapa method yang disediakan oleh kelas *Thread*

Thread Methods
public static Thread currentThread()
Mengembalikan sebuah reference kepada thread yang sedang berjalan.
public final String getName()
Mengembalikan nama dari thread.
public final void setName(String name)
Mengulang pemberian nama thread sesuai dengan argument *name*. Hal ini dapat menyebabkan *SecurityException*.
public final int getPriority()

Mengembalikan nilai prioritas yang telah diberikan kepada thread tersebut.
public final boolean isAlive()
Menunjukkan bahwa thread tersebut sedang berjalan atau tidak.
public static void sleep(long millis)
Menunda thread dalam jangka waktu milis. Hal ini dapat menyebabkan *InterruptedException*.
public void run()
Eksekusi thread dimulai dari method ini.
public void start()
Menyebabkan eksekusi dari thread berlangsung dengan cara memanggil method run.

Prioritas Thread:

Thread juga dapat ditentukan prioritasnya. Penjadwal *thread* akan cenderung menjalankan *thread* dengan prioritas tertinggi terlebih dahulu. Berikut daftar konstanta untuk menentukan prioritas:

Thread Constants
public final static int MAX_PRIORITY
Nilai prioritas maksimum, 10
public final static int MIN_PRIORITY
Nilai prioritas minimum, 1.
public final static int NORM_PRIORITY
Nilai default prioritas, 5.

Sinkronisasi Thread:

Didalam situasi-situasi tertentu, sebuah *thread* yang berjalan bersama-sama kadang-kadang membutuhkan resource atau method dari luar. Oleh karena itu,mereka butuh untuk berkomunikasi satu dengan yang lain sehingga dapat mengetahuistatus dan aktifitas mereka. Sinkronisasi antar *Thread* bisa dilakukan dengna menggunakan *key word synchronized*

Contoh Kode Menggunakan class Thread

```
class PrintNameThread extends Thread {
   PrintNameThread(String name) {
```

```
    super(name);
    start();  //menjalankan thread
   }

    public void run() {
      String name = getName();
       for (int i = 0; i < 10; i++) {
      System.out.print(name);
      }
   }
}

public class TestThread1{
 public static void main(String args[ ]) {
   PrintNameThread pnt1 = new PrintNameThread("A,");
   PrintNameThread pnt2 = new PrintNameThread("B,");
  PrintNameThread pnt3 = new PrintNameThread("C,");
  PrintNameThread pnt4 = new PrintNameThread("D,");
}
}
```

Contoh Kode Mengimplementasikan Interface Runnable

```
class PrintNameThread implements Runnable {
  Thread thread;
  PrintNameThread(String name) {
  thread = new Thread(this, name);
  thread.start();
  }

public void run() {
    String name = thread.getName();
    for (int i = 0; i < 10; i++) {
    System.out.print(name);
  }
 }
}

public class TestThread2 {
    public static void main(String args[ ]) {
        new PrintNameThread("A");
        new PrintNameThread("B");
         new PrintNameThread("C");
         new PrintNameThread("D");
      }
     }
```

Contoh Kode Sinkronisasi Thread

```
class TwoStrings {
static void print(String str1, String str2) {
System.out.print(str1);
try {
```

```
Thread.sleep(500);
} catch (InterruptedException ie) {
}
System.out.println(str2);
}
}
class PrintStringsThread implements Runnable {
Thread thread;
String str1, str2;
PrintStringsThread(String str1, String str2) {
this.str1 = str1;
this.str2 = str2;
thread = new Thread(this);
thread.start();
}
public void run() {
TwoStrings.print(str1, str2);
}
}
public class TestThread3 {
public static void main(String args[ ]) {
new PrintStringsThread("Hello ", "there.");
new PrintStringsThread("How are ", "you?");
new PrintStringsThread("Thank you ", "very much!");
}
}
```

Program diatas, antar thread belum dilakukan sinkronisasi maka outputnya akan menjadi kalimat yang tidak beraturan. Untuk melakukan sinkronisasi antar thread, pada baris kedua, diawal scriptnya tambahkan *key word synchronized*. Sekarang amati perbedaan hasilnya dan bahas dalam laporan.

3. Latihan

ubahlah program sinkronisasi thread dengan memanfaatkan super class dari class thread dan buat sebuah thread untuk skenario pengambilan uang di mesin ATM.