MODUL

PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK

(Institusi)

Disusun oleh :

PULUT SURYATI

SUMIYATUN

SEKOLAH TINGGI MANAJEMEN INFORMATIKA DAN KOMPUTER

AKAKOM

YOGYAKARTA

2019

MODUL 8
PEWARISAN

CAPAIAN PEMBELAJARAN

1. Dapat menggunakan overiding metode,menggunakan klausa super
2. Dapat menggunakan klausa final pada metode dan kelas

KEBUTUHAN ALAT/BAHAN/SOFTWARE

Alat : Komputer, Viewer
Software : Java, TextPad/Netbean

DASAR TEORI

Dalam Java, juga memungkinkan untuk mendeklarasikan class-class yang tidak dapat menjadi subclass. Class ini dinamakan **class final**. Untuk mendeklarasikan class untuk menjadi final kita hanya menambahkan kata kunci **final** dalam deklarasi class. Sebagai contoh, jika kita ingin class Person untuk dideklarasikan final, kita tulis,

```
public final class Person
{
//area kode
}
```

Beberapa class dalam Java API dideklarasikan secara final untuk memastikan sifatnya tidak dapat di-*override*. Contoh-contoh dari class ini adalah Integer, Double, dan String.

Ini memungkinkan dalam Java membuat method yang tidak dapat di-*override.* Method ini dapat kita panggil **method final**. Untuk mendeklarasikan method untuk menjadi final, kita tambahkan kata kunci final ke dalam deklarasi method. Contohnya, jika kita ingin method getName dalam class Person untuk dideklarasikan final, kita tulis,

```
public final String getName(){
```

```
    return name;
    }
```

Method static juga secara otomatis final. Ini artinya Anda tidak dapat membuatnya override.

Keyword super

Pada saat penciptaan objek dari *subclass* (kelas anak), konstruktor dari *superclass* (kelas induk) akan dipanggil.

Perhatikan contoh berikut.

```
public class Titik {
  // koordinat Titik
  private int x;
  private int y;

  // Konstruktor tanpa argumen
  public Titik() {
    System.out.println("konstruktor Titik tanpa argumen");
    x = 0;
    y = 0;
  }

  // Konstruktor dengan argumen x dan y
  public Titik(int x, int y) {
    System.out.println("konstruktor Titik dengan argumen");
    this.x = x;
    this.y = y;
  }
  . . .

}

public class Lingkaran extends Titik {
  protected double radius;

  // konstruktor tanpa argumen
  public Lingkaran(){
    radius = 0.0;
  }

  // konstruktor argumen radius
  public Lingkaran(double rad){
    setRadius(rad);
  }
  . . .

}
```

Ketika diciptakan instan dari kelas lingkaran dengan kode:

```
    Lingkaran l = new Lingkaran(12);
```

Maka kode ini akan memberikan tampilan:

```
konstruktor Titik tanpa argumen
```

Hal ini berarti pada kelas Titik konstruktor yang dipanggil adalah konstruktor pertama (yang tanpa argumen).

Bagaimana jika ingin yang dipanggil adalah konstruktor kedua (dengan argumen)? Hal ini dapat dikerjakan dengan menggunakan keyword ***super***. Perhatikan contoh berikut, pada konstruktor Lingkaran.

```
public class Lingkaran extends Titik {
  protected double radius;
  . . .
  // konstruktor tanpa argumen
  public Lingkaran(){
    super(0, 0);   // tambahkan kode ini
    radius = 0.0;
  }
  . . .

}
```

Ketika diciptakan instan dari kelas lingkaran dengan kode:

```
    Lingkaran l = new Lingkaran();
```

Maka kode ini akan memberikan tampilan:

```
konstruktor Titik dengan argumen
```

Perlu diperhatikan bahwa pemanggilan konstruktor superclass harus pada baris pertama dari konstruktor subclass.

Sintaks untuk pemanggilan konstruktor superclass adalah

```
   super();
   --atau--
   super(parameter list);
```

Overriding Method

Method instan dalam subclass dengan nama dan paramater yang sama, serta return type sama dengan yang ada dalam superclass dikatakan meng-*override* (menimpa) method pada superclass.

Kemampuan subclass meng-*override* method mengijinkan kelas mewarisi dari superclass yang mempunyai perilaku "cukup dekat" dan kemudian memodifikasi perilaku jika dibutuhkan.

Pada saat overriding (penimpaan) method, anda bisa menggunakan annotasi **@Override** yang menyuruh compiler bahwa anda bermamksud meng-*override* method dalam *superclass*. Jika, compiler mendeteksi bahwa method itu tidak ada dalam pada *superclass*, ia akan membangkitkan error.

PRAKTIK

Praktik 1. Membuat Program Mobil

```
public class Mobil {
    protected int kecmaks;
    protected String namaken;

    public Mobil() {
        System.out.println("konstruktor mobil");
    }

    public Mobil(int kecmaks) {
        this.kecmaks = kecmaks;
        System.out.println("Kecepatan Maksimal Mobil = " +kecmaks);
    }

    Mobil(int kecmaks, String namaken){
        this.kecmaks=kecmaks;
        this.namaken = namaken;
    }
    public void PrintInfoMobil(){
        System.out.println("kecepatan maksimal:"+kecmaks());
        System.out.println("Nama Kendaraan:"+mamaken);
    }
}
```

Praktik 2. Membuat Kelas Truck

```
public class Truck extends Mobil {
    protected String speksifikasi;

    public Truck(String speksifikasi, int kec) {
        super(kec);
        this. speksifikasi = speksifikasi;
        System.out.println(speksifikasi);
    }

    public Truck(int kecmaks, String namaken, String speksifikasi){
        super(kecmaks, namaken);
        this.speksifikasi = speksifikasi;
    }

    @Override
```

```
    public void PrintInfoMobil() {
        super.PrintInfoMobil();
        System.out.println("Gandengan:"+ speksifikasi);
    }

    public void PrintInfoTruck(){
        System.out.println("kecepatan maksimal:"+kecmaks);
        System.out.println("Nama Kendaraan:"+namaken);
        System.out.println("Spesifikasi:"+ Spesifikasi);
    }
    }
```

Praktik 3. Membuat Kelas TestMobil

```
   public class TestMobil{
       public static void main(String args[]){
           System.out.println("Merupakan pemanggilan object Mobil");
           Mobil avanza = new Mobil(10, "Avanza");
           avanza.PrintInfoMobil();
           System.out.println("");
           System.out.println("Merupakan pemanggilan object Mobil");
           Truck truk = new Truck(7,"Tronton","kapasitas 10 ton");
           truk.PrintInfoMobil();
           truk.PrintInfoTruck();
       }
   }
```

Praktik 4. Jelaskan tentang override dan adakah contoh implementasi dari praktik diatas

Praktik 5. Jelaskan tentang penggunaan klausa super dan adakah contoh implementasi dari praktik diatas

Praktik 6. Pengguna klausa final pada kelas, Modifikasi kelas Mobil dengan menambahkan klausa final pada deklarasi kelas sebagai berikut :

```
public final class Mobil {
    protected int kecmaks;
    protected String namaken;

    public Mobil() {
        System.out.println("konstruktor mobil");
    }

    public Mobil(int kecmaks) {
        this.kecmaks = kecmaks;
        System.out.println("Kecepatan Maksimal Mobil = " +kecmaks);
    }
```

...

Lakukan pengamatan pada kelas truk, berikan penjelasan kenapa terjadi kesalahan.

Praktik 6. Pengguna klausa final pada metode

Modifikasi kelas Mobil dengan menambahkan klausa final pada deklarasi metode printInfoMobil sebagai berikut :

```
public class Mobil {
    protected int kecmaks;
    protected String namaken;

    public Mobil() {
        System.out.println("konstruktor mobil");
    }

    public Mobil(int kecmaks) {
        this.kecmaks = kecmaks;
        System.out.println("Kecepatan Maksimal Mobil = " +kecmaks);
    }

    Mobil(int kecmaks, String namaken){
        this.kecmaks=kecmaks;
        this.namaken = namaken;
    }
    public final void PrintInfoMobil(){
        System.out.println("kecepatan maksimal:"+kecmaks());
        System.out.println("Nama Kendaraan:"+mamaken);
    }
}
```

Lakukan pengamatan pada kelas truk, berikan penjelasan kenapa terjadi kesalahan.

LATIHAN

1. Buatlah kelas baru yang merupakan turunan dari kelas Mobil praktik 1, kelas turunan mengoveride metode yang ada
2. Buatlah contoh pembuatan objek dan penggunakaan membernya

TUGAS

1. Diberikan oleh dosen pengampu

REFERENSI

MODUL 9
KELAS ABSTRAK

CAPAIAN PEMBELAJARAN

Dapat membuat dan menggunakan kelas asbtrak

KEBUTUHAN ALAT/BAHAN/SOFTWARE

Alat : Komputer, Viewer
Software : Java, TextPad/Netbean

DASAR TEORI

Kelas Abstrak adalah kelas yang tidak dapat dibuat obyeknya (diinstantiasi). Hal ini dikarenakan kelas tersebut belum lengkap karena memiliki minimal satu buah method yang tidak mempunyai definisi, hanya berupa deklarasi saja. Agar dapat diinstantiasi maka kelas turunannyalah yang harus membuat implementasinya (membuat definisi dari method yang belum lengkap tadi). Sebelum semua method dibuat definisinya, maka kelas yang mewarisi kelas abstrak juga tidak dapat diinstantiasi, artinya menjadi kelas abstrak juga.

Biasanya suatu kelas dijadikan abstrak karena kelas tersebut terlalu umum. Sebagai contoh kelas makhluk hidup berisi method-method seperti : bernafas, makan, dan masih banyak lagi. Method-method tersebut terlalu abstrak untuk dibuat definisinya sehingga perlu dibuat kelas khusus yang merupakan turunan dari kelas makhluk hidup, yang mengimplementasikan method-method diatas.

Deklarasi kelas abstrak :

```
[modifier] abstract class NamaKelasAbstract {
            daftar field
            deklarasi method-method;
      }

Implementasi kemudian dilakukan pada class turunannya:
      class Turunan extends NamaKelasAbstract {
```

```
        definisi method-method;
        …
}
```

Beberapa hal yang perlu diperhatikan adalah sebagai berikut:

1. Class abstrak tidak dapat dibuatkan instan atau objeknya menggunakan keyword new.
2. Sebuah class dapat dideklarasikan sebagai class abstrak walaupun tidak memiliki method abstrak.
3. Variabel dengan tipe class abstrak tetap bisa diciptakan, tetapi harus mereferensi ke subclass dari class abtrak tersebut yang tentunya tidak abstrak.

PRAKTIK

Praktik 1 : Penggunaan kelas abstrak berikut:

```
abstract class Bentuk {
        protected int panjang;
        protected int lebar;
        public String getBentuk () {
                return "bentuk dasar";
        }
  public abstract int HitungLuas ( );
}

class BujurSangkar extends Bentuk {
        public BujurSangkar(int panjang, int lebar){
                this.panjang = panjang;
                this.lebar = lebar;
        }
        public String getBentuk () {
                return "bentuk bujur sangkar";
  }
  public int HitungLuas ( ){
                return panjang * lebar;
        }
}

class Segitiga extends Bentuk {
        public Segitiga(int panjang, int lebar){
                this.panjang = panjang;
                this.lebar = lebar;
        }
```

```
        public String getBentuk () {
                return "bentuk segitiga";
        }

    public int HitungLuas (){
                return (panjang * lebar)/2;
        }
}

class TestAbstrak2 {
        public static void cetakLuasBentuk(Bentuk btk){
                System.out.println(btk.getBentuk() + " dengan luas :" + btk.HitungLuas());
        }

    public static void main (String [ ] args) {
                BujurSangkar bs = new BujurSangkar(10,20);
                Segitiga st = new Segitiga(5,10);
                cetakLuasBentuk(bs);
                cetakLuasBentuk(st);

        }
}
```

1. Buat objek dari kelas Bentuk, berikan penjelasan mengapa terjadi kesalahan
2. Buat kelas Lingkaran yang merupakan turunan dari kelas Bentuk

```
class Lingkaran extends Bentuk {
  public  Lingkatan(){
     System.out.println("Kelas Lingkatan turunan kelas abstrak bentuk");
  }
  public String getBentuk(){
     Return "Bentuk Lingkatan";
  }
  public static void main(String[] args){
     Bentuk  geometri = new BujurSangkar()
  }
  }
```

Berikan penjelasan kenapa tidak terjadi kesalahan pada pembuatan objek geometri

LATIHAN

1. Buatlah sebuah class abstract bernama Mahasiswa memiliki method sbb :
   - isiBiodata(String nama, int nilai) sebagai method abstract
   - registrasi(), menampilkan tulisan ke layar : "nama.....telah melakukan registrasi"
   - testMasuk(), menampilkan tulisan ke layar : "nilai test anda ....."
2. Tambahkan variabel nama dan nilai pada Mahasiswa
3. Buatlah class MahasiswaBaru yang melakukan extending dari class Mahasiswa. Deklarasi ulang method isiBiodata, bodi method akan

menampilkan ke layar biodata anda (“nama:....., nilai : .....,pilihan jurusan:,.....asal sekolah : .....”)

4. Tambahkan main method pada MahasiswaBaru sehingga output sbb :

```
run:
nama ahmad ,nilai : 80
nama ahmad telah melakukan registrasi
nilai test anda 80
BUILD SUCCESSFUL (total time: 1 second)
```

TUGAS

1. Buatlah kelas abstrak Manusia yang berisi deskripsi umum dari manusia. Kemudian buatlah kelas Mahasiswa dan kelas Dosen yang merupakan turunan dari kelas Manusia, dan masing-masing kelas menangani data mahasiswa dan dosen.
2. Buat objek dari kelas yang dibuat

REFERENSI

MODUL 10
INTERFACE

CAPAIAN PEMBELAJARAN

Dapat membuat dan menggunakan interface pada aplikasi

KEBUTUHAN ALAT/BAHAN/SOFTWARE

Alat : Komputer, Viewer
Software : Java, TextPad/Netbean

DASAR TEORI

Interface adalah kelas yang benar-benar abstrak, artinya hanya berisi deklarasi method dan (jika ada) konstanta saja. Method-method tersebut nantinya harus diimplementasikan pada *real class*. **Interface dapat dianalogikan seperti menandatangani kontrak kerja. Misalnya seorang dosen wajib mengajar, membuat soal, dsb, akan tetapi cara mengajar dan membuat soalnya diserahkan ke masing-masing dosen (tidak ditentukan dalam kontrak kerja).**

Interface mendefinisikan suatu protokol perilaku tanpa dipusingkan dengan implementasinya. Suatu kelas yang mengimplementasikan interface maka pada kelas tersebut akan melekat perilaku yang didefinisikan interface.

Sebagai contoh : Dalam kehidupan nyata dapat diketahui ada manusia yang bekerja sebagai tentara, penyanyi, pengacara, dan sebagainya, tentunya manusia-manusia tersebut selain harus memiliki method standard sebagai seorang manusia, juga harus memiliki method yang sesuai dengan pekerjaannya. Dengan demikian untuk membuat objek manusia yang bekerja sebagai penyanyi, harus dibuat kelas yang merupakan turunan kelas manusia yang meng-implementasikan interface penyanyi.
Delarasi interface :

```
[modifier] interface NamaInterface {
        deklarasi konstanta;
        deklarasi method-method;
}
```

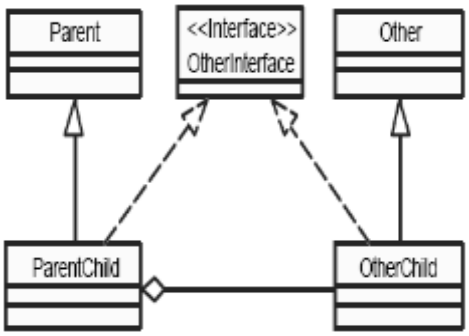
Implementasi kemudian dilakukan pada class lainnya:

```
[modifier] class NamaKelas implements NamaInterface{
        definisi method-method;
        …
}
```

Pewarisan merupakan proses penurunan field dan method dari suatu kelas induk kepada satu/lebih subkelas. Seringkali dibutuhkan suatu kelas yang field dan methodnya berasal dari lebih dari satu kelas induk (pewarisan jamak). Pewarisan jamak memang mempercepat dalam pembuatan kelas. Tetapi mempunyai beberapa kelemahan diantaranya adalah terjadi ambiguitas karena adanya method yang sama yang diturunkan dari beberapa kelas induk.

Java tidak mengijinkan pewarisan jamak. Sebagai gantinya java menggunakan interface. Terdapat 2 jenis penyelesaian masalah pewarisan jamak di java, yaitu :

1. Kombinasi antara turunan satu kelas dan interface




```
[modifier] class NamaKelas extends NamaKelasInduk implements NamaInterface
{
   //isi kelas
}
```

2. Dengan implementasi multiple interface

```
[modifier] class NamaKelas implements NamaInterface1, NamaInterface2, … {
   //isi kelas
}
```

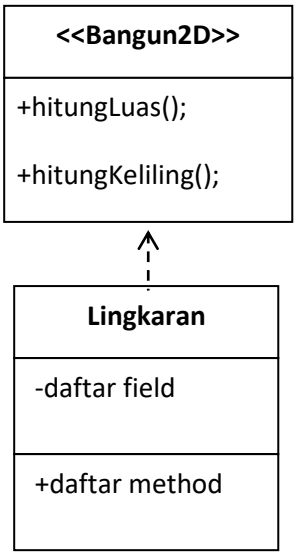
Suatu Interface dapat diperluas menjadi Interface baru yang lebih lengkap.

```
[modifier] interface NamaInterface2 extends  NamaInterface1{
    //daftar konstanta dan method
}
```

PRAKTIK

Praktek 1 : Interface Bangun2D

Interface Bangun2D memiliki 2 method, yaitu : hitung_Luas() dan hitung_Keliling(). Kelas Lingkaran yang mengimplementasikan interface Bangun2D harus membuat definisi untuk kedua method tersebut.




```
interface Bangun2D{
        public double hitungLuas();
        public double hitungKeliling();
}

class Lingkaran implements Bangun2D{
        private double jejari;

        public void setJejari(double jejari){
                this.jejari=jejari;
        }
        public double getJejari(){
                return this.jejari;
        }
        public double hitungLuas(){
                return (3.14 * this.jejari * this.jejari);
        }
        public double hitungKeliling(){
                return (2 * 3.14 * this.jejari);
        }
}
```

```
class TestLingkaran{
	public static void main(String[] arg){
		Lingkaran bunder = new Lingkaran();
		bunder.setJejari(10);
		double luas = bunder.hitungLuas();
		double keliling = bunder.hitungKeliling();
		System.out.println("Luas lingkaran dengan jejari "+bunder.getJejari()+ "
		adalah "+luas);
		System.out.println("Keliling lingkaran dengan jejari "+bunder.getJejari()+"
		adalah "+keliling);
	}
}
```

- Dari contoh penggunaan interface di atas jelaskan maksud dari hasil program tersebut

Praktek 2 : Kombinasi antara turunan satu kelas dan interface

```
interface MProvides{
	void func();
}

interface MRequires{
	int getValue();
}

class Mixin implements MProvides
{
	private final MRequires parent;
	public Mixin(MRequires parent) {
		this.parent = parent;
	}
	public void func() {
		System.out.println("Nilai dari method func: " + parent.getValue());
	}
}

class Parent
{
	private int value;
	public Parent(int value ) {
		this.value = value;
	}
	public int getValue() {
		return this.value;
	}
}

class Child extends Parent implements MRequires, MProvides{
	private final MProvides mixin;
	public Child(int value){
```

```
                super(value);
                this.mixin = new Mixin(this);
        }

        public void func(){
                mixin.func();
        }
}

class TestInherInterface{
        public static void main(String[] arg){
                Child anak = new Child(5);
                anak.func();
                System.out.println("nilai dari method getValue:"+anak.getValue());
        }
}
```

- Dari contoh penggunaan interface di atas jelaskan maksud dari hasil program tersebut

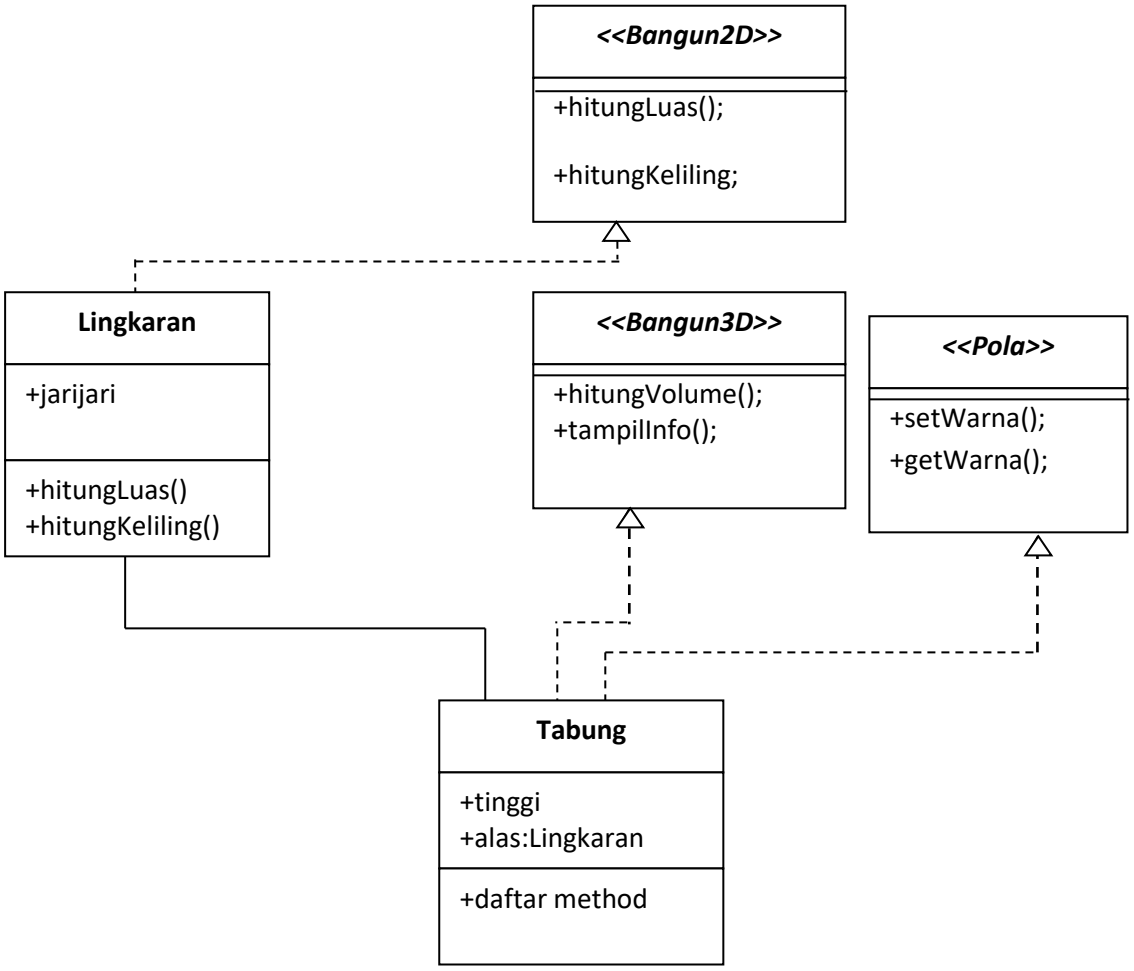
Praktek 3 : Multiple interface

```
interface Interface1 {
  public void tampilInfo();
  public void setInfo(String info);
}

interface Interface2 {
  public void cetakInfo();
}

public class MultiInterfaces implements Interface1, Interface2 {
  private String info;

  public void setInfo(String info) {
            this.info = info;
  }
  public void tampilInfo(){
            System.out.println(this.info+": ini info dari method tampilInfo");
  }

  public void cetakInfo(){
            System.out.println(this.info+": ini info dari method cetakInfo");
  }

  public static void main(String[] a) {
          MultiInterfaces t = new MultiInterfaces();
           t.setInfo("Hai");
           t.tampilInfo();
  }
}
```

- Dari contoh penggunaan interface di atas jelaskan maksud dari hasil program tersebut

LATIHAN

1. Gambarkan class diagram untuk program pada praktek 1!
2. Buatlah program Java yang mengimplementasikan multiple interface seperti diagram kelas berikut,



TUGAS

1. Jelaskan perbedaan antara kelas abstrak dan interface!

REFERENSI

MODUL 11
POLIMORFISME

CAPAIAN PEMBELAJARAN

Dapat membuat aplikasi yang mengimplementasi compile-time (early binding) polimorphisme, dapat membuat aplikasi yang mengimplementasi runtime (late binding) polimorphisme

KEBUTUHAN ALAT/BAHAN/SOFTWARE

Alat : Komputer, Viewer
Software : Java, TextPad/Netbean

DASAR TEORI

Polimorfisme adalah kemampuan sebuah variabel reference untuk merubah behavior sesuai dengan apa yang dipunyai object.

Polimorfisme membuat objek-objek yang berasal dari subclass yang berbeda, diperlakukan sebagai objek-objek dari satu superclass. Hal ini terjadi ketika memilih method yang sesuai untuk diimplementasikan ke objek tertentu berdasarkan pada subclass yang memiliki method bersangkutan.

Polimorfisme dapat diterjemahkan pula sebagai “sebuah method yang sama namanya (homonim) tetapi mempunyai tingkah laku yang berbeda”. Komputer membedakan method berdasarkan signature method (saat compile) atau berdasarkan reference object (saat runtime).

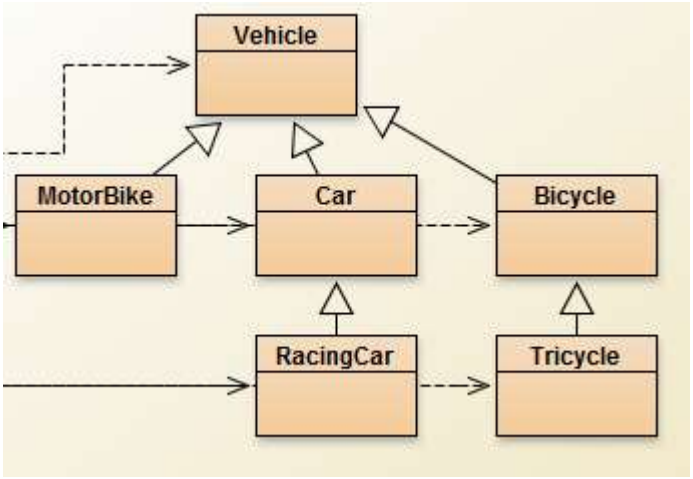
Ada dua bentuk polimorfisme

1. Overloading (compile-time (early binding) polimorphisme)
   Overloading adalah salah satu cara penerapan dalam konsep polimorfisme. Overload merupakan pendefinisian ulang suatu metode dalam class yang sama. Syarat overload yaitu metode dan tipe parameter harus berbeda dalam kelass yang sama.

2. Overriding (runtime (late binding) polimorphisme)
   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.

   Dikatakan **late-binding (run time) polymorphism** karena saat compile komputer tidak tahu method mana yang harus dipanggil, dan baru akan diketahui saat run-time. Run-time polymorphism dapat diterapkan melalui **overridden method.** Run time polymorphism juga disebut dengan istilah **dynamic binding.**

   Anggaplah subclass meng-override method tertentu pada superclass. Andai kita cipta objek dari subclass dan memasukkannya dalam superclass. Walau objek subclass telah dimasukkan pada superclass, sewaktu objek tersebut memanggil method yang dioverride, ia tetap memanggil method yang versi subclassnya, bukan versi superclass.
   Contoh



- Class Vehicle memiliki method move().
- Class MotorBike meng-override method move().
  Java akan menunggu saat runtime untuk menentukan method move() yang mana akan dipanggil.

PRAKTIK

Praktik 1. compile-time (early binding) polimorphisme dengan overloading method

```
class Jumlah{

   public int tambah(int x, int y){
```

```
   return x + y;
   }

   public int tambah(int x, int y, int z){
   return x + y + z;
   }

   public int tambah(double pi, int x){
   return (int)pi + x;
   }
}

public class Penjumlahan{

public static void main(String [] args){
Jumlah obj = new Jumlah();
System.out.println(obj.tambah(2,5)); // int, int
System.out.println(obj.tambah(2, 5, 9)); // int, int, int
System.out.println(obj.tambah(3.14159, 10)); // double,
int
}
}
```

Pada praktik di atas polymorphism ditunjukkan dengan menggunakan berbagai method ***tambah***. Komputer membedakan mana method yang dipanggil berdasarkan signature dari method yaitu parameter inputnya (jumlah parameter input, type data parametr input, dan urutan type data).

Bentuk polymorphism ini dinamakan **early-binding (**atau **compile-time) polymorphism** karena komputer mengetahui mana method tambah yang dipanggil setelah mengcompile byte code . Jadi setelah proses compile ketika code sudah menjadi byte code, komputer akan "tahu" method mana yang akan dieksekusi Bentuk ini yang kita kenal dengan **overloaded method.**

Praktik 2. compile-time (early binding) polimorphisme dengan overloading Konstruktor

```
class Penggajian{
 double gapok;
 double masa_kerja;
 Penggajian(double g, double mk){
       gapok = g;
       masa_kerja=mk;
 }
 Penggajian() {
 gapok =0;
 masa_kerja=0;
 }
```

```
 Penggajian(double lembur)   {
       gapok = masa_kerja = lembur;
 }
 double hitung_gaji()  {
       return gapok*masa_kerja;
 }
}

class OverloadingKonstruktor {
 public static void main(String args[]) {
       Penggajian Karyawan1 = new Penggajian(10,15);
       Penggajian Karyawan2 = new Penggajian();
       Penggajian Karyawan3 = new Penggajian(5);
       double gaji;
       gaji = Karyawan1.hitung_gaji();
       System.out.println("Gaji Karyawan 1= " +gaji);
       gaji = Karyawan2.hitung_gaji();
       System.out.println("Gaji Karyawan 2= " +gaji);
       gaji = Karyawan3.hitung_gaji();
       System.out.println("Gaji Karyawan 3= " +gaji);
       }
 }
```

Praktik 3. Runtime polimorfisme dengan overriding method

```
class Kendaraan {
public void info() {
    System.out.println("Info pada kendaraan : ");
    }
}

class Roda2 extends Kendaraan {
public void info() {
    System.out.println ("Info pada Roda2");

    }
}

class Motor extends Roda2{
public void info() {
    System.out.println("Info pada Motor");
    }
}



public class Test {
public static void main(String[] args){
    Roda2 roda2ku;
```

```
    Motor motorku;
    Kendaraan k = new Kendaraan();
    roda2ku=new Roda2();
    motorku=new Motor();
    k.info();
    k=roda2ku;
    k.info();
    k=motorku;
    k.info();
}
}
```

Praktik 4. Runtime polimorfisme dengan data member

```
class Induk {
int x = 5;
public void Info() {
System.out.println("Ini class Induk");
}
}

class Anak extends Induk {
int x = 10;
public void Info() {
System.out.println("Ini class Anak");
}
}

public class Test2 {
public static void main(String args[])
{
Induk tes=new Anak();
System.out.println("Nilai x = " + tes.x);
tes.Info();
}
}
```

Ketika program di atas dijalankan akan terlihat bahwa ketika diakses atribut x dari objek tes, maka yang muncul adalah nilai x dari super kelas, bukan atribut x dari subkelas. Akan tetapi ketika diakses method Info() dari objek tes, yang muncul adalah method Info() dari sub kelas, bukan method Info() dai superkelas.

Praktik 5. Runtime polimorfisme dengan passing parameter

```
class AlatGerak{
      void bergerak(){
            System.out.println(" Saya mampu bergerak");
      }
}

class Sayap extends AlatGerak{
      void bergerak(){
            System.out.println(" Saya bisa terbang");
      }
}

class Kaki extends AlatGerak{
      void bergerak(){
      System.out.println(" Saya bisa jalan-jalan");
      }
}

class Burung {
      private AlatGerak alatGerak=new AlatGerak();
      Burung(){
            System.out.println("Hai saya Burung");
      }

public void bergerak() {
      alatGerak.bergerak();
}

public void setAlatGerak(AlatGerak alatGerak) {
      this.alatGerak = alatGerak;
      System.out.println("Sekarang saya pakai " +
alatGerak);
      }
}

class BurungTest{
public static void main(String[] args){
Burung merpati=new Burung();
merpati.bergerak();
Sayap sayap=new Sayap();
Kaki kaki=new Kaki();
merpati.setAlatGerak(sayap);
merpati.bergerak();
```

```
merpati.setAlatGerak(kaki);
merpati.bergerak();
}
}
```

Method setAlatGerak dapat menerima berbagai type yaitu Sayap dan Kaki. Hal ini legal karena pada kenyataannya Sayap dan Kaki adalah AlatGerak. Saat method bergerak() milik object trutle dipanggil, method yang dipanggil adalah sesuai dengan method yang dimiliki oleh type yang diberikan.

LATIHAN

1. Diberikan oleh dosen pengampu

TUGAS

1. *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
2. *Dikerjakan di rumah dan dilampirkan pada laporan.*

REFERENSI

MODUL 12
EXCEPTION HANDLING

CAPAIAN PEMBELAJARAN

Dapat menggunakan perintah yang menangani Exception pada program,menggunakan throws untuk melempar eksepsi dan membuat Execption sendiri

KEBUTUHAN ALAT/BAHAN/SOFTWARE

Alat : Komputer, Viewer
Software : Java, TextPad/Netbean

DASAR TEORI

Dalam pembuatan program seringkali dijumpai error atau kesalahan. Oleh karena itu, diperlukan suatu mekanisme yang membantu menangani error atau kesalahan yang terjadi, baik saat pembuatan maupun implementasi program. Java menyediakan mekanisme dalam pemrograman untuk menangani hal-hal tersebut yang disebut dengan ***exception***.

Exception adalah event yang terjadi ketika program menemui kesalahan pada saat instruksi program dijalankan. Banyak hal yang dapat menimbulkan event ini, misalnya crash, harddisk rusak dengan tiba-tiba, sehingga program-program tidak bisa mengakses file-file tertentu. Programmer pun dapat menimbulkan event ini, misalnya dengan melakukan pembagian dengan bilangan nol, atau pengisian elemen array melebihi jumlah elemen array yang dialokasikan dan sebagainya.

Exception terdiri dari dua macam kelompok, yaitu :

- Exception yang merupakan subclass RunTimeException
- Exception yang bukan subclass RunTimeException

RunTimeException biasanya disebabkan oleh kesalahan program atau pada desain program. Misalnya NullPointerException yang disebabkan oleh proses inisialisasi program yang tidak sempurna dan ArrayIndexOutOfBoundsException yang disebabkan akses array yang melebihi kapasitas array yang ada.

Dalam bahasa Java, ketika terjadi kesalahan, otomatis akan dilemparkan sebuah objek yang disebut ***exception***, yang kemudian dapat diproses lebih lanjut oleh fungsi-fungsi yang siap menangani kesalahan tersebut. Proses pelemparan exception tersebut sering dikenal dengan istilah ***throwing exception***, sedangkan proses penerimaan exception yang bersangkutan dikenal dengan istilah ***catch exception***

Ada lima kata kunci yang digunakan oleh Java untuk menanngani exception ini, yaitu, **try**, **catch**, **finally**, **throw**, dan **throws**.

Situasi yang menyebabkan exception dibagi menjadi 3 kategori yaitu,

- **Kesalahan kode atau data**. Contohnya jika kode berusaha mengakses suatu indeks dari array yang di luar batas array.
- **Metode standar exception**. Contohnya, metode substring() dalam kelas String, dapat memunculkan pesan dalam bentuk StringIndexOutofBoundsException.
- **Kesalahan Java**. Hal ini merupakan kesalahan dalam mengeksekusi Java Virtual Machine yang dijalankan pada saat kompilasi.

Istilah exception digunakan untuk menunjukkan kesalahan pada hardware, software, serta algoritma. Suatu exception adalah obyek dari kelas standar Throwable, yang mempunyai turunan terdiri dari :

- **Error**. Exception yang didefinisikan dalam keas ini mempunyai karakteristik bahwa kondisi yang terjadi tidak dapat diubah. Terdapat tiga macam subclass yang berada dibawahnya yaitu, ThreadDeath, Linkage Error, VirtualMachineError.
- **Exception**. Untuk kelas yang diturunkan dari exception kompiler memeriksa bahwa kita bisa menangani exception dengan metode yang ada.

Kelas exception mempunyai banyak kelas turunan. Dua kelas yang penting yaitu,

- **IOException**
- **RuntimeException**

Tabel Subkelas dari RunTime Exception

Kelas	Keterangan
ArithmeticException	Kesalahan pada operasi aritmatika
IndexOutofBoundsException	Beberapa jenis indeks di luar batas
NegativeArraySizeException	Array diciptakan dengan ukuran negatif
NullPointerExcepotion	Penggunaan acuan null yang tidak valid
ArrayStroreException	Penyimpanan array dengan tipe data yang tidak sesuai.
ClassCastException	Cast yang tidak valid
IllegalArrayArgumentException	Argumen yang tidak benar
SecurityException	Aturan sekuriti yang dilanggar
IllegalMonitorStateException	Operasi monitor ilegal
IllegalStateException	Lingkungan yang tidak benar
UnsupportedOperationException	Operasi yang tidak didukung

Kata kunci throw digunakan di program untuk melempar (throw) exception secara eksplisit. Bentuk umum kalimat adalah,

```
Throw ThrowableInstance;
```

Instan Throwable harus merupakan obyek dengan tipe Throwable atau subkelas dari Throwable. Terdapat dua cara untuk memperoleh obyek Throwable, yaitu,

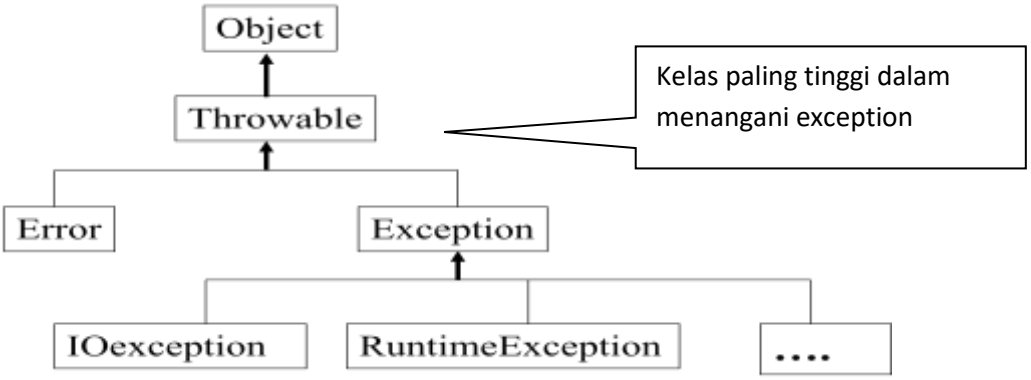
1. Menggunakan parameter di klausa catch
2. Menciptakab salah satu dengan menggunakan operator new()

Eksekusi program akan dihentikan segera setelah kalimat throw, kalimat-kalimat setelah kalimat throw tidak dieksekusi. Java akan melakukan inspeksi blok try terdekat untuk menemukan catch yang cocok dengan dengan tipe exception yang dilempar. Jika Java menemukannya, maka kendali program ditransfer ke kalimat itu. Jika tidak ditemukan, maka Java akan melakukan penelusuran ke blok berikutnya dan bila tetap tidak ditemukan, maka penanganan exception secara default akan menghentikan programnya.

Penggunaan kata kunci throws berhubungan erat dengan penggunaan exception yang dicek oleh Java. Klausa throws mendaftarkan tipe-tipe exception yang dapat dilempar method. Hal ini diperlukan agar diketahui semua exception

yang mungkin dilempar method atau subkelasnya, kecuali tipe Error atau RunTimeException yang dilakukan sistem Java secara otomatis bila menemui pelangggaran aturan-aturan Java. Semua exception yang hendak dilempar method harus dideklarasikan di klausa throws. Jika method melemparkan exception yang tidak dideklarasikan di deklarasi method, maka kompilator akan memberi pesan kesalahan

Hirarki kelas yang menangani exception:



Struktur penulisan exception adalah :

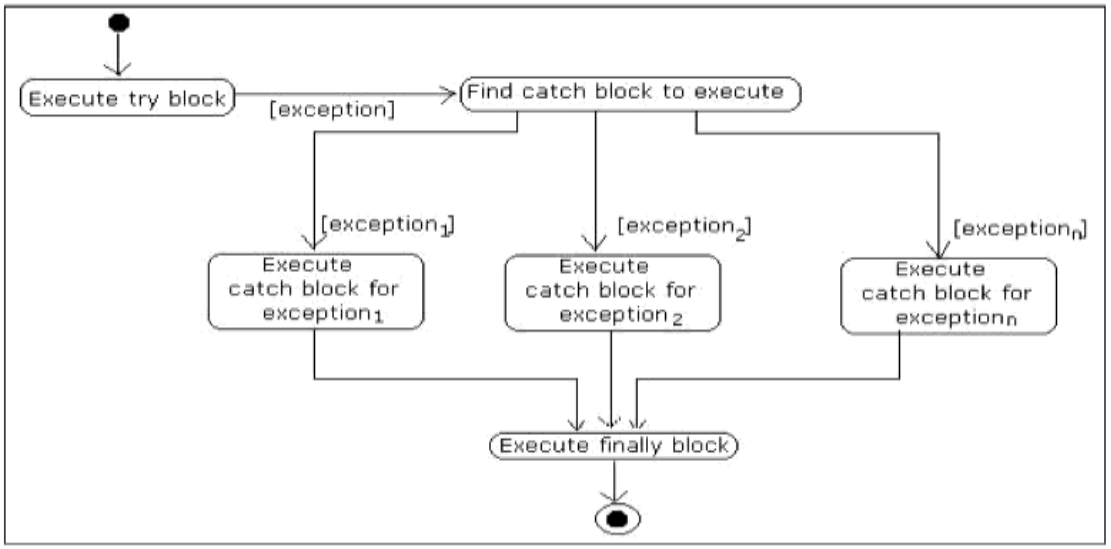
```
try {
   // blok program tempat menuliskan statement yang akan di uji
apakan ada kesalahan atau tidak
}
catch ( <tipe exception1> <nama variabel>) {
   // blok program tempat menuliskan aksi program bila sebuah
exception terjadi
}
. . .
catch ( <tipe exception2> <nama variabel>) {
   // blok program tempat menuliskan aksi program bila sebuah
exception terjadi
}
finally {
   // blok program lanjutan

}
```

Ketentuan dalam membuat exception adalah:

a. Wajib membuat notasi blok
b. Setiap blok try boleh memiliki lebih dari satu blok catch dan hanya boleh memiliki satu blok finally
c. Blok catch dan blok finally harus muncul bersama blok try
d. Blok try harus diikuti minimal satu blok catch, atau satu blok finally, atau kedua blok catch dan finally
e. Setiap blok catch mendefinisikan penanganan exception. Di dalam header blok catch terdapat satu argumen yang akan ditangani oleh blok

exception. Exception harus berasal dari class Throwable atau dari class turunannya.

Alur kerja dari exception :



PRAKTIK

Praktik 1 . Membuat kelas tanpa exception

```
1. public class BagiNol {
2. public static void main(String[] args) {
3.      System.out.println("Sebelum pembagian");
4.      System.out.println(5/0);
5.      System.out.println("Sesudah pembagian");
6.   }
7. }
```

Baris ke-5 tidak akan dieksekusi karena ada kesalahan pembagian dengan bilangan nol pada baris ke-4

Praktik 2 . Membuat kelas menggunakan exception

```
1.  public class BagiNol2 {
2.  public static void main(String[] args) {
3.   System.out.println("Sebelum pembagian");
4.   try { System.out.println(5/0); }
5.      catch (Exception t) {
```

```
6.       System.out.print("Pesan kesalahan: ");
7.       System.out.println(t.getMessage());
8.     }
9.     System.out.println("Sesudah pembagian");
10.  }
11.  }
```

Program tidak berhenti di baris ke -4 , dibuktikan dengan munculnya hasil dari baris ke-9

Praktik 3 . Menggunakan lebih dari satu statement catch

```
public class MultiCatchException{
public static void main(String args[]){
   try{
    System.out.println(5/0);
   }
   catch(ArithmeticException e){
    System.out.println(0);
   }
   catch(ArrayIndexOutOfBoundsException e){
    System.out.println(1);
   }
   catch(Exception e){
     System.out.println(2);
   }
}
}
```

Output adalah angka nol yang artinya pada blok try terdapat kesalahan *arithmetic* maka kesalahan tersebut ditangkap oleh blok catch dari kelas ArithmeticException.

Praktik 4 . Menggunakan statement finally

```
public class MultiCatchException{
public static void main(String args[]){
   try{
    System.out.println(5/0);
   }
   catch(ArithmeticException e){
    System.out.println(0);
   }
   catch(ArrayIndexOutOfBoundsException e){
    System.out.println(1);
   }
   catch(Exception e){
     System.out.println(2);
   }
   finally{
     System.out.println(3);
   }
```

```
    }
    }
```

Bagian blok *finally* adalah bagian blok yang selalu dikerjakan. Output program adalah angka nol dan tiga, artinya setelah salah satu blok *catch* dieksekusi maka alur program selanjutnya adalah mengeksekusi bagian blok *finally.*

Praktik 5. Menggunakan kata kunci throw untuk **eksepsi**

```
public class DemoEksepsi {
public static void methodLain() {
  try     {
      throw new ArrayIndexOutOfBoundsException(1);
  }
  catch (ArrayIndexOutOfBoundsException e){
      System.out.println("Penanganan eksepsi dalam Method MethodLain()");
      throw e;
}
}
public static void main(String[]args) {
      try    {
        methodLain();
       }
     catch (ArrayIndexOutOfBoundsException e){
        System.out.println("Penanganan eksepsi dalam   Method main()");
     }
 }
 }
```

LATIHAN

1. Diberikan oleh dosen pengampu

TUGAS

1. Tugas diberikan oleh dosen pengampu pada akhir praktikum.
2. Dikerjakan di rumah dan dilampirkan pada laporan.

REFERENSI

MODUL 13
INNER CLASS

CAPAIANPEMBELAJARAN

Dapat mendefinsikan inner class, nested class, non-static inner class, local class dan Anonymous class, serta dapat mengimplementasi konsep inner class pada aplikasi

KEBUTUHAN ALAT/BAHAN/SOFTWARE

Alat : Komputer, Viewer
Software : Java, TextPad/Netbean

DASAR TEORI

Java mengijinkan kita mendefiniskan suatu kelas di dalam kelas lain, hal ini disebut sebagai *nested class*. Disebut nested karena class bersifat tersarang terhadap kelas – kelas utamanya, seperti halnya blok penyeleksian (if, for) yang tersarang pada blok penyeleksian lainnya atau method yang tersarang pada method lainnya. *nested class* dibagi menjadi dua kategori, yaitu *static* dan *non static nested class*. Istilah *inner class* biasa digunakan untuk *non static nested class*.

```
Class OuterClass{
        ...
        static class StaticNestedClass{
                ...
        }
        Class InnerClass{
                ...
        }
}
```

Nested class merupakan member dari kelas Outer(kelas terluar). Non static nested class (inner class) dapat mengakses semua member dari kelas Outer

bahkan yang mempunyai hak akses private. Static nested class tidak mempunyai akses kepada member dari kelas Outer. Sebagai member dari Outer class, sebuah nested class dapat dideklarasikan dengan hak akses private, public, proteced, sedangkan Outer class hanya bisa dideklarasikan dengan hak akses public.
Anonymous inner class adalah sebuah inner class yang dideklarasikan tanpa memberikan nama kelas. Anonymous class akan membuat kode program menjadi lebih ringkas karena dengan Anonymous class kita dapat mendeklarasikan dan meng-instant suatu kelas sekaligus dalam satu langkah.

PRAKTIK

Membuat inner class

1. Tulislah program file **Luar.java** berikut:

```
public class Luar{
 private int angkaLuar;
public Luar(int angkaLuar){
this.angkaLuar=angkaLuar;
}

int getAngkaLuar(){
 return angkaLuar;
}
          //inner class
           class Dalam{
           private int angkaDalam;

           public Dalam(){
                    angkaDalam=9;
           }

           public void cetakDalam(){
           System.out.println("Ini angka luar:"+angkaLuar);
           System.out.println("Ini angka dalam:"+angkaDalam);
           }
           }//batas inner class
public void cetakLuar(){
   Dalam dl=new Dalam();
   dl.cetakDalam();
  }
  public static void main(String args[]){
   Luar lu=new Luar(5);
   lu.cetakLuar();
  }
}
```

2. Tulislah program file **RoundShape.java** berikut:

```
public abstract class RoundShape
{ // coordinates of center represented by an inner class
  protected class Center
  {  int x,y; }
protected Center C = new Center();
protected float radiusOfCircle;

// constructor
public RoundShape(int xCenter, int yCenter, float radius)
{
C.x=xCenter;
C.y=yCenter;
radiusOfCircle = radius;
}
// abstract method
abstract public float area();
}
```

3. Tulislah program file **Circle.java** berikut:

```
public class Circle extends RoundShape {
 // constructor
  public Circle(int xCenter, int yCenter, float radius){
       super(xCenter, yCenter, radius);
}

public float area(){
  float areaOfCircle = (float)(Math.PI*Math.pow(radiusOfCircle,2.0));
  return areaOfCircle;
}
}
```

4. Tuliskan program file **Sphere.java**

```
public class Sphere extends RoundShape{
public Sphere(int xCenter, int yCenter, float radius){
      super(xCenter, yCenter, radius);
}

public float area() {
      float surfaceArea =   (float)(4.0*Math.PI*Math.pow(radiusOfCircle,2.0));
      return surfaceArea;
}
}
```

5. Tuliskan **TestRoundShape.java**

```
import java.text.DecimalFormat;
public class TestRoundShape{
public static void main(String args[]){
DecimalFormat digit = new DecimalFormat ("0.##");
Circle lingk1=new Circle(20,5,10.0f);
System.out.println(digit.format(lingk1.area()));
}
}
```

Praktik 6. membuat Anonymous Class

Tuliskan **TestAnonymous.java**

```
class Awal{
int x=8;
 void methodAwal(){
 System.out.println("Nilai x= "+x);
 }
}

public class TestAnonymous{
 public static void main(String args[]){
       Awal noName=new Awal(){ //instant anonymous class
        void methodAwal(){
        x+=3;
        System.out.println("x= "+x);
        }
      };
      noName.methodAwal();
 }

}
```

Baris program di bawah tulisan ”public static void main” adalah proses instant dari sebuah anonymous class. Obyek noName bukan obyek dari kelas Awal namun obyek dari anonymous class hal ini dapat dilihat dari nilai x=11, bukan x=8 ketika program dijalankan

LATIHAN

1. *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
2. *Dikerjakan di laboratorium pada jam praktikum.*

TUGAS

1. *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
2. *Dikerjakan di rumah dan dilampirkan pada laporan.*

REFERENSI

MODUL 14
KONKURENSI

CAPAIAN PEMBELAJARAN

Dapat menggunakan kelas Threat untuk menangani konkurensi

KEBUTUHAN ALAT/BAHAN/SOFTWARE

Alat : Komputer, Viewer
Software : Java, TextPad/Netbean

DASAR TEORI

Sebuah thread, digunakan untuk meningkatkan fungsionalitas dan performansi dengan cara melakukan beberapa tugas pada saat yang sama, yaitu bersamaan. Ada dua metode untuk menerapkan thread di dalam Java :

1. penerapan sebuah antarmuka
2. perpanjangan sebuah class

Thread merupakan pemrograman Java tingkat menengah, untuk itu mahasiswa diharapkan sudah mengenal dengan konsep-konsep dasar Object Oriented Paradigm dan mengerti dengan istilah seperti 'extending', 'interface' dan 'class'. Alasan mengapa ada dua cara membuat thread. Hal Ini dikarenakan jika sebuah class sudah menjadi sebuah class turunan dari beberapa class selain 'Thread', maka ini tidak dapat memperpanjang 'Thread' karena beberapa turunan tidak diperbolehkan di dalam pemrograman bahasa Java. Jadi, dalam kasus seperti itu kita gunakan antarmuka 'Runnable' sebagai gantinya.

Cara membuat thread.

Metode pertama adalah memperpanjang atau menurunkan class 'Thread'. Class 'Thread' didefinisikan di dalam paket java.lang, dimana perlu diimpor. Perhatikan kode dibawah untuk mendapat ide yang lebih baik :

```
import java.lang.*;
public class MyExample extends Thread   {
 public void run()   {

  ....

 }
 }
```

Cara yang lain untuk melakukan hal ini dengan cara menerapkan 'Runnable', seperti terlihat dibawah,

```
import java.lang.*;
 public class MyExample implements Runnable {
          Thread T;
  public void run()   {
  ....
  }
  }
```

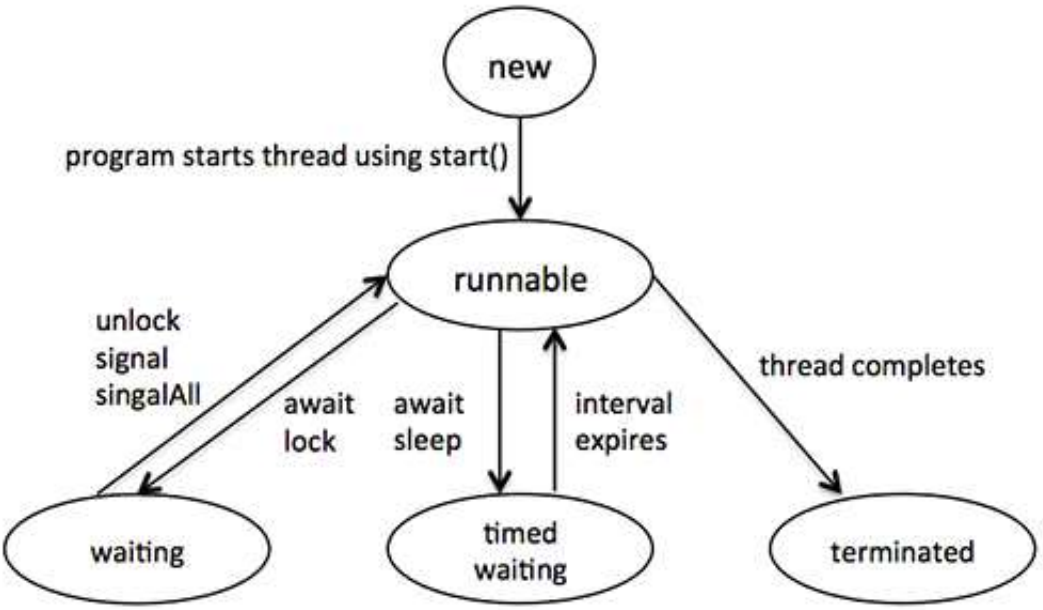
Perhatikan pada kedua metode menggunaan fungsi 'run()', dimana bertanggung jawab pada kerja Thread. Antarmuka 'Runnable' sebenarnya bukan apa-apa tetapi sebuah class yang berisi hanya satu metode abstrak, 'public abstract void run();'. Yang perlu diingat adalah bahwa sebuah antarmuka hanya menyediakan sebuah rancangan framework atas class yang dapat diterapkan. Ini juga menarik untuk dicatat bahwa sebenarnya class 'Thread' juga menerapkan antarmuka 'Runnable'.

Menciptakan Thread :

```
Thread thread = new Thread();
thread.start();
```

Pada kode Thread diatas tidak melakukan sesuatu apapun yg berguna, agar Thread tsb bisa berguna maka dibutuhkanmethod run(),ada 3 buah cara tuk menciptakan Thread dgn method run() -nya.

Sebuah thread runnable melalui berbagai tahap dalam siklus hidupnya. Sebagai contoh, thread new, start, runnable, dan kemudian terminated. Setelah diagram menunjukkan siklus hidup lengkap thread.



New: Sebuah thread baru start siklus hidupnya di negara baru. Masih dalam keadaan ini sampai program dijalankan thread. Hal ini juga disebut sebagai thread new.

Runnable: Setelah thread baru new start, thread menjadi runnable. Sebuah thread di negara bagian ini dianggap melaksanakan tugasnya.

Waiting: Kadang-kadang, thread transisi ke negara menunggu sementara thread menunggu thread lain untuk melakukan task.A thread transisi kembali ke keadaan runnable hanya ketika thread lain sinyal thread menunggu untuk melanjutkan mengeksekusi.

Timed waiting: Sebuah thread runnable dapat memasuki state menunggu waktunya untuk interval waktu tertentu. Sebuah thread di state bagian ini transisi kembali ke keadaan runnable ketika interval waktu berakhir atau ketika acara itu sedang menunggu terjadi.

Terminated: Sebuah thread runnable memasuki keadaan dihentikan ketika selesai tugasnya atau berakhir.

PRAKTIK

Praktik 1: membuat Thread

```
class RunnableDemo implements Runnable {
   private Thread t;
   private String threadName;

   RunnableDemo( String name){
       threadName = name;
       System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", "+i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
     } catch (InterruptedException e) {
     System.out.println("Thread " + threadName+ " interrupted.");
     }
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

Praktik 2. Menggunakan kelas RunnableDemo

```
public class TestThread {
   public static void main(String args[]) {
      RunnableDemo R1 = new RunnableDemo( "Thread-1");
      R1.start();
      RunnableDemo R2 = new RunnableDemo( "Thread-2");
      R2.start();
   }
}
```

Praktik 3. Membuat Thread menggunakan extend Thread

```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;

   ThreadDemo( String name){
       threadName = name;
       System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: "+threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
     } catch (InterruptedException e) {
         System.out.println("Thread "+threadName+" interrupted.");
     }
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }
}
```

Praktik 4. Menggunakan Thread

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

      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();

      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```

PRAKTIK 5 Dengan menggunakan Anonymous Inner Classes

Kadang kita tidak ingin menciptakan class untuk Thread dan kondisinya tidak cocok untuk pake interface Runnable. Jika hal tersebut terjadi, maka dapat membuat anonymous inner class

```
public class MyClass{
public MyClass(){
new Thread(){
public void run(){
    System.out.println("Threat dijalanankan ");
   }
}.start();
}

public static void main(String[] args){
new MyClass ();
}
}
```

LATIHAN

- *Latihan diberikan oleh dosen pengampu pada saat praktikum.*
- *Dikerjakan di laboratorium pada jam praktikum.*

TUGAS

- *Tugas diberikan oleh dosen pengampu pada akhir praktikum.*
- *Dikerjakan di rumah dan dilampirkan pada laporan.*

REFERENSI