Kerangka Java

Fungsi input

```
import java.util.Scanner;  // Import the Scanner class

class Main {

  public static void main(String[] args) {

    Scanner myObj = new Scanner(System.in);  // Create a Scanner object

    System.out.println("Enter username");

    String userName = myObj.nextLine();  // Read user input

    System.out.println("Username is: " + userName);  // Output user input

  }

}
```

Tipe Data

byte, short, int, long, float, double, boolean, char

tipe data dengan tambahan postfix:

float num = 5.75f;

double num = 5.75d;

long num = 5L;

For

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

  System.out.println(i);

}
```

Tenary

```
String result = (time < 18) ? "Good day." : "Good evening.";
```

Array multidimensi

```
    int[][] arr = { { 1, 2 }, { 3, 4 } };
```

atau

```
    int[][] array = new int[rows][columns];
```

Setter dan Getter

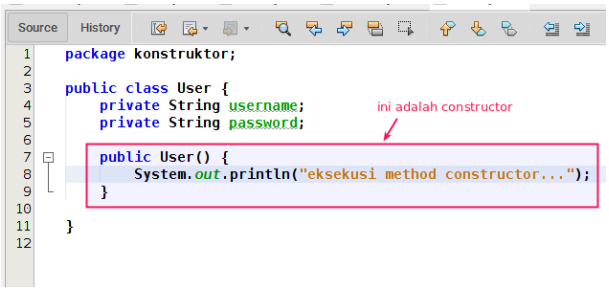
```
  private int value;
  public int getValue() {
    return value;
  }
  public void setValue(int value) {
    this.value = value;
  }
```

KONSEP PBO

CONSTTRUCTOR

Constructor adalah method khusus yang **akan dieksekusi pada saat pembuatan objek** (instance). Biasanya method ini **digunakan untuk inisialisasi** atau mempersiapkan data untuk objek.

Contoh Constructor dan Cara Membuatnya




```
package konstruktor;

public class User {
    private String username;
    private String password;

    public User() {
        System.out.println("eksekusi method constructor...");
    }

}
```

Cara membuat constructor adalah dengan menuliskan nama method constructor sama seperti nama class

Pada contoh di slide sebelumnya constructor ditulis seperti ini:

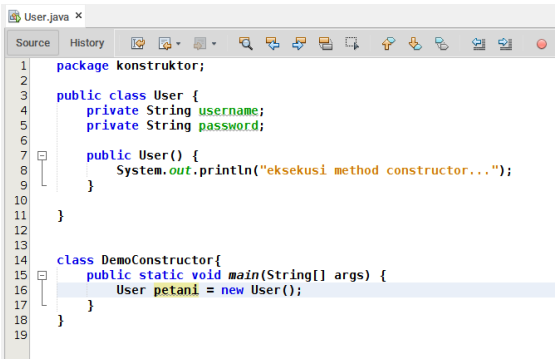
```
public User() {
System.out.println("eksekusi method constructor...");
}
```

Pastikan memberikan modifier public kepada Constructor , karena ia akan dieksekusi saat pembuatan objek (instance).

Mari kita coba membuat objek baru dari class User:

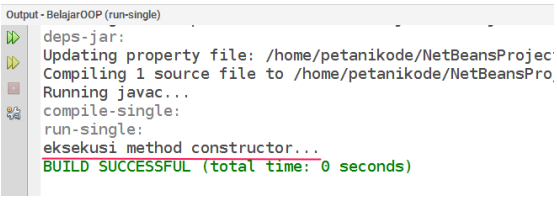
User petani = **new** user();

Sehingga sekarang kita punya kode lengkap seperti ini




```
package konstruktor;

public class User {
    private String username;
    private String password;

    public User() {
        System.out.println("eksekusi method constructor...");
    }

}


class DemoConstructor{
    public static void main(String[] args) {
        User petani = new User();
    }
}
```

Hasilnya saat dieksekusi:



Constructor dengan Parameter

Constructor biasanya digunakan untuk initialize (menyiapkan) data untuk class. Untuk melakukan ini, kita harus membuat parameter sebagai inputan untuk constructor.

```
public class User {
    public String username;
    public String password;
    public User(String username, String password){
      this.username = username;
      this.password = password;
}
```

Pada kode class User di atas , kita menambahkan parameter username dan password ke dalam constructor.

Berarti nanti saat kita membuat objek, kita harus menambahkan nilai parameter seperti ini:

User petani = new User(“petanikode”, “kopi”);

```
Output - BelajarOOP (run-single)
Updating property file: /home/petanikode/NetB
Compiling 1 source file to /home/petanikode/N
Running javac...
compile-single:
run-single:
Username: petanikode
Password: kopi
BUILD SUCCESSFUL (total time: 0 seconds)
```

Destructor dalam Java

Destructor adalah method khusus yang akan dieksekusi saat objek dihapus dari memori. Java sendiri **tidak memiliki method destructo**r , karena **Java menggunakan gerbage collector** untuk manajemen memjorinya.

Jadi Si gerbage collector akan otomatis menghapus objek yang tidak terpakai.

Sementara untuk bahasa pemrograman lain, seperti C++ kita bisa membuat destructor sendiri seperti ini:

```
class User {
   public:
    User(String*username);// <--ini
constructor
    ~User();// <--ini destructor.
   private:
    String username;
    String password;
};
```

Enkapsulasi & Hidding Information

Enkapsulasi merupakan suatu cara pembungkusan data dan method yang menyusun suatu kelas sehingga kelas dapat dipandang sebagai suatu modul dan cara bagaimana menyembunyikan informasi detail dari suatu class information hiding).

Dalam OOP, enkapsulasi sangat penting untuk keamanan serta menghindari kesalahan permrograman , enkapsulasi dimaksudkan untuk menjaga suatu proses program agar tidak dapat diakses secara sembarangan atau di intervensi oleh program lain.

Dua hal yang mendasar dalam enkapsulasi

1) Information hiding: Sebelumnya untuk pengaksesan atribut atau method menggunakan objek secara langsung. Hal ini karena akses kontrol yang diberikan pada atribut dan method di dalam kelas tersebut adalah public. Untuk menyembunyikan informasi dari suatu kelas sehingga anggota kelas tersebut tidak dapat diakses kelas lain yaitu dengan memberi hak akses private pada atributnya .
2) Interface to access data: Interface to access data ini merupakan cara melakukan perubahan terhadap atribut yang disembunyikan , caranya adalah dengan membuat suatu interface berupa method untuk menginisialisasi atau merubah nilai dari suatu atribut tersebut .

Manfaat Enkapsulasi

Manfaat utama teknik encapsulation adalah kita mampu memodifikasi kode tanpa merusak kode yang telah digunakan pada class lain.

Modularitas: Source code dari sebuah class dapat dikelola secara independen dari source code class yang lain. Perubahan internal pada sebuah class tidak akan berpengaruh bagi class yang menggunakannya.

Information Hiding: Penyembunyian informasi yang tidak perlu diketahui objek lain.

Keuntungan menerapkan enkapsulasi

Bersifat independen: Suatu modul yang terencapsulasi dengan baik akan bersifat independen, sehingga tidak akan terikat pada bagian tertentu dari program.

Bersifat transparan: Bila melakukan modifikasi pada suatu modul, maka perubahan tersebut akan dirasakan juga oleh bagian program yang menggunakan modul tersebut.

Menghindari efek diluar perencanaan: Modul yang terencapsulasi dengan baik hanya akan berinteraksi dengan bagian program lainnya melalui variable variabel input output yang telah didefinisikan sebelunnya .

Melindungi listing program: Saat program didistribusikan pada khalayak, untuk melindungi listing program Anda dapat menerapkan prinsip enkapsulasi. Di sini pengguna hanya dapat menggunakan program melalui variable input atau output yang didefinisikan tanpa disertai bagimana proses yang terjadi di dalam modul tersebut.

Contoh:

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

        public void Info() {
                System.out.println("Saya siswa PENS");
        }
}
```

```
1  public class IsiData {
2          public static void main(String args[]) {
3                  Siswa IT2=new Siswa();
4                  IT2.nrp=5;
5                  IT2.nama="Andi";
6                  IT2.Info();
7          }
8  }
```

Hasil Runing ?

```
Hallo.java:4: nrp has private access in Siswa
                IT2.nrp=5;
```

Contoh Enkapsulasi

- Terdapat Class Circle.

Untuk memberikan nilai X dengan menggunakan cara aCircle.setX(10); begitu juga dengan nilai Y dan Z. Untuk mendapatkan jari-jari menggunakan aCircle.getR()

```
class MyMain
{
    public static void main(String args[])
    {
        Circle aCircle;  // creating reference
        aCircle = new Circle(); // creating object
        aCircle.setX(10);
        aCircle.setY(20);
        aCircle.setR(5);
        double area = aCircle.area(); // invoking method
        double circumf = aCircle.circumference();
        System.out.println("Radius="+aCircle.getR()+" Area="+area);
        System.out.println("Radius="+aCircle.getR()+" Circumference ="+circumf);
    }
}
```

Inheritance

Inheritance atau Pewarisan/Penurunan adalah konsep pemrograman dimana sebuah class dapat 'menurunkan' property dan method yang dimilikinya kepada class lain. Konsep inheritance digunakan untuk memanfaatkan fitur 'code reuse' untuk menghindari duplikasi kode program. Sumber Konsep inheritance ini mengadopsi dunia Konsep inheritance ini mengadopsi dunia riil dimana suatu entitas/obyek dapat mempunyai entitas/obyek turunan. Dengan konsep inheritance , sebuah class dapat mempunyai class turunan.

Pengertian dasar

Suatu class yang mempunyai class Suatu class yang mempunyai class turunan dinamakan parent class atau base class. Sedangkan class turunan itu sendiri seringkali disebut subclass atau child class. Suatu subclass dapat mewarisi apa apa yang dipunyai oleh parent class. Karena suatu subclass dapat mewarisi apa Karena suatu subclass dapat mewarisi apa apa yang dipunyai oleh parent class nya , maka member dari suatu subclass adalah terdiri dari apa apa yang ia punyai dan juga apa apa yang ia warisi dari class parent nya . Kesimpulannya , boleh dikatakan bahwa suatu subclass adalah tidak lain hanya memperluas (extend) parent class (extend) parent class nya

Deklarasi inheritance

Dengan menambahkan kata kunci Dengan menambahkan kata kunci extends setelah deklarasi nama class kemudian diikuti dengan nama parent kemudian diikuti dengan nama parent class nya. Kata kunci extends tersebut memberitahu Kata kunci extends tersebut memberitahu kompiler Java bahwa kita ingin melakukan perluasan class perluasan class

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

Semua class di dalam Java adalah merupakan sub c l ass dari class super induk yang bernama **Object**.

Misalnya saja terdapat sebuah class sederhana :

```
Public class A {
…
}
```

Pada saat dikompilasi Kompiler Java Pada saat dikompilasi, Kompiler Java akan membacanya sebagai subclass dari class Object class Object

```
public class A extends Object
   …
}
```

Pada saat dikompilasi Kompiler Java Pada saat dikompilasi, Kompiler Java akan membacanya sebagai subclass dari class Object class Object

```
public class A extends Object
  …
}
```

Kita baru perlu menerapkan inheritance Kita baru perlu menerapkan inheritance pada saat kita jumpai ada suatu class yang dapat diperluas dari class lain yang dapat diperluas dari class lain.

Contoh terdapat class Pegawai

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

Contoh terdapat class Manager

```
public class Manajer {
  public String nama;
  public double gaji;
  public String departemen;
}
```

Dari 2 buah class diatas , kita lihat class Manajer mempunyai data member yang identik sama dengan class Pegawai, hanya saja ada tambahan data member departemen. Sebenarnya yang terjadi disana adalah class Manajer merupakan perluasan dari class Pegawai dengan tambahan data member departemen tambahan data member departemen. Disini perlu memakai konsep inheritance , sehingga class Manajer dapat kita tuliskan seperti berikut

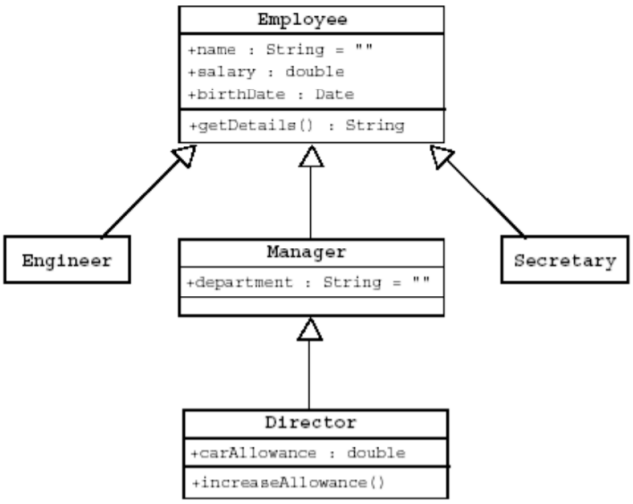
```
public class Manajer extends Pegawai {
  public String departemen;
}
```

Single Inheritance

Konsep inheritance yang ada di Java adalah Java hanya memperkenankan adanya **single inheritance** Konsep single inheritance hanya memperbolehkan suatu sublass **mempunyai satu parent class** .

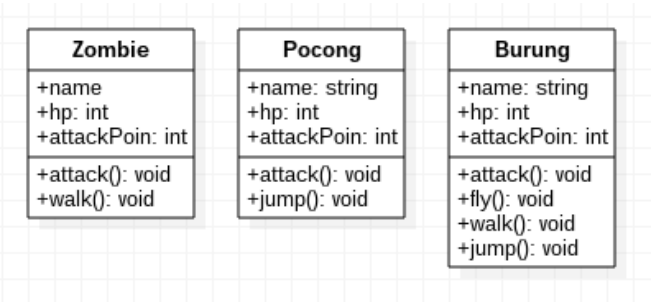
Multilevel Inheritance

Konsep inheritance yang ada di Java Konsep inheritance yang ada di Java memperkenankan adanya multilevel inheritance. Konsep multilevel inheritance memperbolehkan suatu subclass mempunyai subclass lagi.



Contoh tanpa Inheritance

Misalkan dalam Game , kita akan membuat class class musuh dengan perilaku yang berbeda.



Lalu kita membuat kode untuk masing masing kelas seperti ini:

Zombie.java

```
class Zombie {
    String name;
    int hp;
    int attackPoin;

    void attack(){
        // ...
    }

    void walk(){
        //...
    }
}
```

Pocong.java

```
class Pocong {
    String name;
    int hp;
    int attackPoin;

    void attack(){
        // ...
    }

    void jump(){
        //...
    }
}
```

Burung.java

```
class Burung {
    String name;
    int hp;
    int attackPoin;

    void attack(){
        // ...
    }

    void walk(){
        //...
    }

    void jump(){
        //...
    }

    void fly(){
        //...
    }
}
```

Apakah boleh seperti ini?

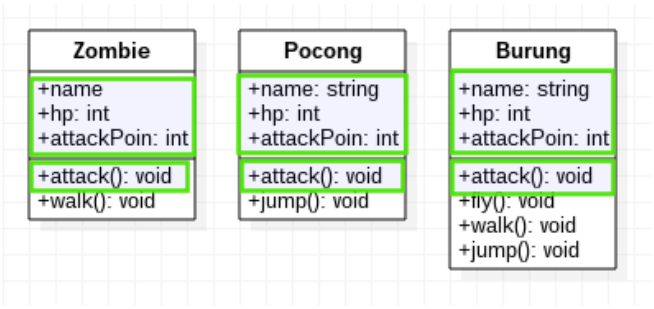
Ya, boleh boleh saja. Akan Tapi ti

```
class Burung {
    String name;
    int hp;
    int attackPoin;

    void attack(){
        // ...
    }

    void walk(){
        //...
    }

    void jump(){
        //...
    }

    void fly(){
        //...
    }
}
```

dak efektif, karena kita menulis berulang ulang properti dan method yang sama.

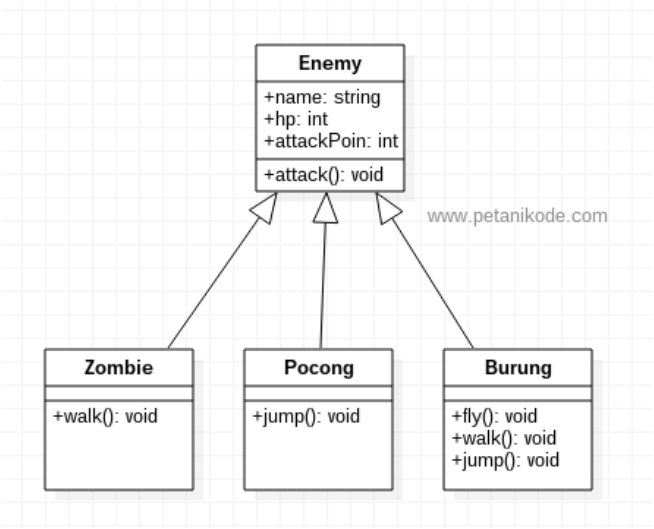
Contoh dengan Inheritance

Ada hal yang sama pada class dibawah



Menggunakan inheritance

Inheritance di StarUML digambarkan dengan garis hubung Generalization. Class Enemy adalah class induk yang memiliki anak Zombie, Pocong , dan Burung . Apapun properti yang ada di class induk , akan dimiliki juga oleh class anak.



Enemy.java

```
class Enemy {
    String name;
    int hp;
    int attackPoin;

    void attack(){
        System.out.println("Serang!");
    }
}
```

Zombie.java

```
class Zombie extends Enemy {
    void walk(){
        System.out.println("Zombie jalan-jalan");
    }
}
```

Pada class anak, kita menggunakan kata kunci extends untuk

menyatakan kalau dia adalah class turunan dari Enemy

Pocong.java

```
class Pocong extends Enemy {
    void jump(){
        System.out.println("loncat-loncat!");
    }
}
```

Burung.java

```
class Burung extends Enemy {
    void walk(){
        System.out.println("Burung berjalan");
    }
    void jump(){
        System.out.println("Burung loncat-loncat");
    }
    void fly(){
        System.out.println("Burung Terbang...");
    }
}
```

Pada class anak, kita menggunakan kata kunci extends untuk menyatakan kalau dia adalah class turunan dari Enemy.

Lalu, bila kita ingin membuat objek dari class class tersebut, Kita bisa membuatnya seperti ini:

```
Enemy monster = new Enemy();
Zombie zumbi = new Zombie();
Pocong hantuPocong = new Pocong();
Burung garuda = new Burung();
```

OVERLOADING

Overloading Constructor di Java memungkinkan Anda untuk menulis lebih dari satu Constructor dalam satu kelas. Setiap metode dalam Java, termasuk **Constructor**, metode yang **memodelkan perilaku obje**k, dan metode yang **melakukan perhitungan**, dapat di-**overload**. Ini berarti Anda dapat memiliki lebih dari satu metode dengan nama yang sama tetapi dengan parameter yang berbeda dalam satu kelas. Setiap Constructor ataupun method yang di-overload memiliki nama yang sama tetapi berbeda dalam **jumlah, tipe**, atau **urutan** parameter. Dengan demikian, kelas dapat memiliki jumlah method atau Constructor yang tidak terbatas.

Contoh overload pada Method

Jumlah parameter

```
public class Calculator {

   public double sum(double num1){
      return num1;
   }//end method sum

   public double sum(double num1, double num2){
      return num1 + num2;
   }//end method sum

   public double sum(double num1, double num2, double num3){
      return num1 + num2 + num3;
   }//end method sum

}//end class Calculator
```

Tipe Parameter

```
public class Calculator {

   public double sum(double num1, double num2){
      return num1 + num2;
   }//end method sum

   public double sum(int num1, int num2){
      return num1 + num2;
   }//end method sum

}//end class Calculator
```

Urutan parameter

```
public class Calculator {

   public double sum(int num1, double num2){
      return num1 + num2;
   }//end method sum

   public double sum(double num1, int num2){
      return num1 + num2;
   }//end method sum

}//end class Calculator
```

CONSTRUCTORS

```
public class UIWheelLight {
     …
   //Constructors
   public UIWheelLight(UIWheel w, double r){
       wheel = w;
       rotation = r;
       isLit = false;
   }//end Constructor

   public UIWheelLight(UIWheel w, double r, boolean l){
      wheel = w;
      rotation = r;
      isLit = l;
   } //end Constructor
} //end class UIWheelLight
```

2 parameters

3 parameters

Pemanggilan Constructor yang di-overload dalam Java memungkinkan suatu objek diinisialisasi dengan memanggil Constructor kelas mana pun yang sesuai. Anda menyediakan argumen, dan Java akan menemukan Constructor yang paling sesuai.

UIWheelLight light1 = new UIwheelLight(blueWheel, 45);

UIWheelLight light1 = new UIWheelLight(blueWheel, 45, false);

Redundansi dalam Constructor

Adanya kode yang sangat serupa yang diulang-ulang dalam Constructor tersebut. Dalam Java, kita bisa meminimalkan redundansi ini untuk memperbaiki efisiensi dan keterbacaan kode.

```
public class UIWheelLight {
   //Constructors
   public UIWheelLight(UIWheel w, double r){
      wheel = w;
      rotation = r;
      isLit = false;
   }//end constructor

   public UIWheelLight(UIWheel w, double r, boolean l){
      wheel = w;
      rotation = r;
      isLit = l;
   }//end constructor
}//end class UIWheelLight
```

First occurrence

Repeated

Dengan menggunakan kata kunci "this" dalam Java, satu Constructor dapat memanggil Constructor lain dalam kelas yang sama. Ini membantu dalam mengurangi redundansi kode.

```
public class UIWheelLight {
   //Constructors
   public UIWheelLight(UIWheel w, double r){
      this(w, r, false);
   }//end constructor

   public UIWheelLight(UIWheel w, double r, boolean l){
      wheel = w;
      rotation = r;
      isLit = l;
   }//end constructor
}//end class UIWheelLight
```

METHODS

Memanggil Overloaded Methods

Memanggil metode yang di-overload di Java melibatkan menyediakan argumen, dan Java akan menemukan metode yang paling sesuai berdasarkan argumen tersebut.

```
public class CalculatorTest{

   public static void main(String[] args){
      Calculator calc = new Calculator();

      calc.sum(1, 2);
      calc.sum(1, 2, 3);
      calc.sum(1.5, 4.5);
   }//end method main

}//end class CalculatorTest
```

Redudansi dalam Methods

```
public class Calculator{
   …
   public double calcY(double m, double x){
      double y = 0;
      y = mx;
      return y; ;
   }//end method calcY
   public double calcY(double m, double x, double b){
      double y = 0;
      y = mx + b;
      return y;
   }//end method calcY
}//end class Calculator
```

First occurrence

Repeated

```
public class Calculator{
   …
   public double calcY(double m, double x){
      return calcY(m,x,0);
   }//end method calcY

   public double calcY(double m, double x, double b){
      double y = 0;
      y = mx + b;
      return y;
   }//end method calcY
}//end class Calculator
```

Method Signature

Metode Signature adalah kombinasi unik dari nama metode, jumlah, tipe, dan urutan parameter. Tidak termasuk nama parameter atau tipe return. Perubahan pada ini tidak cukup untuk overload metode.

This is the method signature

```
public void setPosition(double x, double y){
   //Do math
}//end method setPosition
```

These aren't part of the method signature

```
public void setPosition(double x, double y){
   //Do math
}//end method setPosition
```

Kesalahan Overloading

Tidak sama nama parameter

```
public class Calculator {

   public double sum(double num1, double num2){
      return num1 + num2;
   }//end method sum
   public double sum(double x, double y){
      return x + y;
   }//end method sum
}//end class Calculator
```

Tidak sama return

```
public class Calculator {

   public double sum(double num1, double num2){
      return num1 + num2;
   }//end method sum
   public double sum(double x, double y){
      return x + y;
   }//end method sum
}//end class Calculator
```

Akan menyebabkan Error: Is already defined

OVERRIDDING

```
class Animal {
  public void sound() {
    System.out.println("Hewan bersuara");
  }
}

class Cat extends Animal {
  @Override
  public void sound() {
    System.out.println("Meong");
  }
}

public class Main {
  public static void main(String[] args) {
    Animal animal = new Animal();
    animal.sound(); // Output: Hewan bersuara

    Cat cat = new Cat();
    cat.sound(); // Output: Meong
  }
}
```