

# Daftar ISI

[Daftar ISI](#_heading=h.gjdgxs) 2

1. [Prinsip-Prinsip Dasar OOP](#_heading=h.3rdcrjn) 3
2. [Encapsulation](#_heading=h.lnxbz9) 4
3. [Apa itu encapsulasi ?](#_heading=h.35nkun2) 4
4. [Penerapan enkapsulasi pada kode program ?](#_heading=h.2jxsxqh) 5
5. [Bagaimana Contoh ?](#_heading=h.z337ya) 5
6. [Kode Program Lengkap](#_heading=h.3j2qqm3) 7
7. [Inheritance](#_heading=h.1y810tw) 10
8. [Apa itu inheritance ?](#_heading=h.4i7ojhp) 10
9. [Istilah Inheritance](#_heading=h.2xcytpi) 10
10. [Method Inheritance](#_heading=h.1ci93xb) 10
11. [Contoh Penggunaan Extends](#_heading=h.3whwml4) 11
12. [Contoh penggunaan Method Overloading](#_heading=h.2bn6wsx) 11
13. [Contoh penggunaan Method Overriding](#_heading=h.qsh70q) 12
14. [Keuntungan Ketika Overriding Method](#_heading=h.3as4poj) 15
15. [Aturan-aturan Pada Overriding Method](#_heading=h.1pxezwc) 16
16. [Contoh penggunaan Keyword Super](#_heading=h.49x2ik5) 16
17. [Memanggil Conctructor Dari Superclass](#_heading=h.2p2csry) 17
18. [Memanggil Method Milik Superclass](#_heading=h.147n2zr) 18
19. [Digunakan Untuk Mengacu Secara Langsung Pada Variabel Instance Milik Super Class](#_heading=h.3o7alnk) 22
20. [Polymorphism](#_heading=h.23ckvvd) 26
21. [Apa itu Polymorphism](#_heading=h.ihv636) 26
22. [Jenis Polymorphism](#_heading=h.1hmsyys) 26
23. [Overloading](#_heading=h.41mghml) 26
24. [Abstraction](#_heading=h.2grqrue) 27
25. [Mengapa Harus Pakai Class Abstrak?](#_heading=h.3fwokq0) 27
26. [Gerbace Collector](#_heading=h.1v1yuxt) 30
27. [Garbage Collection pada Java](#_heading=h.4f1mdlm) 30

[Daftar Pustaka](#_heading=h.2u6wntf) 34

# Prinsip-Prinsip Dasar OOP

Dalam membahas apa itu OOP atau Object-Oriented Programming, adalah merupakan hal yang penting untuk juga menjelaskan tentang prinsip-prinsip dasarnya.

Ya, perlu teman-teman ketahui bahwa Object-Oriented Programming (OOP) atau Pemrograman Berorientasi Objek (PBO) ini didasarkan pada prinsip-prinsip berikut.

* Encapsulation
* Abstraction
* Inheritance
* Polymorphism

# *Encapsulation*

## Apa itu encapsulasi ?

*Encapsulation* atau pengkapsulan adalah konsep tentang pengikatan data atau metode berbeda yang disatukan atau “dikapsulkan” menjadi satu unit data.

Encapsulation di Java adalah proses untuk Menyembunyikan

dan melindungi secara menyeluruh (hak akses) dari *fungsionalitas* dan detail implementasinya. Yang dilindungi adalah class, variable, dan method dengan cara pemberian hak akses (modifier) yang terdiri dari empat tipe yaitu private, protected, public, dan kosong/tanpa modifier.



Gambar Modifier

Keterangan :

* Class = Class itu sendiri
* Package = Package Class yang sama
* Sub Class = Class anaknya
* Global = selain di class itu sendiri, package yang sama, dan sub classnya

Berikut penjelasan

* Modifier Private adalah modifier yang hanya bisa di akses oleh class itu sendiri.
* Modifier Protected adalah modifier yang hanya bisa di akses oleh class itu sendiri dan anak2nya (inheritance) serta class yang satu package dengannya.
* Modifier Public adalah modifier yang bisa di akses dari class manapun dan package manapun.
* Tanpa modifier adalah hak akses yang mirip sama protected tapi untuk class anak2nya (inheritance) harus satu package baru dapat akses.

## Penerapan enkapsulasi pada kode program ?

* + setter getter di java
  + model
  + entity

## Bagaimana Contoh ?

Contoh Pada Variabel memberikan akses public

| public String type; public int roda; public Double kecepatan; |
| --- |

Contoh pada Method memberikan akses public

| public void doMelaju(String kecepatan){  System.*out*.println("Melaju dengan kecepatan : "+kecepatan); }  public void doKlakson(){  System.*out*.println("klakson"); }  public void doBelok(String arah){  System.*out*.println("Belok ke arah "+arah); } |
| --- |

Contoh pada class memberikan public

| public Mobil objMobil = new Mobil(); |
| --- |

Contoh Setter dan Getter pada Atribut

| public String getType() {  return type; }  public void setType(String type) {  this.type = type; }  public int getRoda() {  return roda; }  public void setRoda(int roda) {  this.roda = roda; }  public Double getKecepatan() {  return kecepatan; }  public void setKecepatan(Double kecepatan) {  this.kecepatan = kecepatan; } |
| --- |

Jika di run,cara memanggil code seperti kode dibawah ini:

| public static Mobil *objMobil* = new Mobil(); public static void main(String[] args) {  *objMobil*.setRoda(4);  System.*out*.println("isi method roda :" + *objMobil*.getRoda()); } |
| --- |

Dapata dilihat pada kode *objMobil*.setRoda(4), penerapan encapsulasi. Objek tersebut dibungkus dalam method setRoda.

Maka outputnya adalah:



Gambar Output Encapsulasi

## Kode Program Lengkap

Class Mobil

| package com.example.demo.model; //Class public class Mobil {   //Variabel  public String type;  public int roda;  public Double kecepatan;   //Construktor  public Mobil(){};    public Mobil(String type, int roda, Double kecepatan) {  this.type = type;  this.roda = roda;  this.kecepatan = kecepatan;  }   //Method VOID  public void doMelaju(String kecepatan){  System.*out*.println("Melaju dengan kecepatan : "+kecepatan);  }   public void doKlakson(){  System.*out*.println("klakson");  }   public void doBelok(String arah){  System.*out*.println("Belok ke arah "+arah);  }   //Return Method  public double hitungKecepatan(Double kecepatan, int waktuDetik){  return kecepatan/waktuDetik;  }   //Setter and Getter  public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public int getRoda() {  return roda;  }   public void setRoda(int roda) {  this.roda = roda;  }   public Double getKecepatan() {  return kecepatan;  }   public void setKecepatan(Double kecepatan) {  this.kecepatan = kecepatan;  } } |
| --- |

Class MainMobil

| public class MainMobil {  public static Mobil *objMobil* = new Mobil();  public static void main(String[] args){    *objMobil*.setRoda(4);  System.*out*.println("isi method roda :" + *objMobil*.getRoda());  }  } |
| --- |

# Inheritance

## Apa itu inheritance ?

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.

## Istilah Inheritance

* Extends  
  Keyword ini harus kita tambahkan pada definisi class yang menjadi subclass.
* Superclass Superclass digunakan untuk menunjukkan hirarki class yang berarti class dasar dari subclass/class anak.
* Subclass Subclass adalah class anak atau turunan secara hirarki dari superclass.
* Super Keyword ini digunakan untuk memanggil konstruktor dari superclass atau menjadi variabel yang mengacu pada superclass.
* Methode Overriding Pendefinisian ulang method yang sama pada subclass

## Method Inheritance

Terdiri dari :

* Method overriding

Method overriding adalah mendefinisikan kembali method yang sama, baik nama method maupun signature atau parameter yang diperlukan dalam subclass.

Overriden adalah method yang terdapat di super class

Overriding adalah method yang terdapat di sub class

* Method overloading

Method overloading adalah mendefinisikan method yang memiliki nama yang sama, tetapi dengan signature yang berbeda dalam definisi class yang sama.

## Contoh Penggunaan Extends

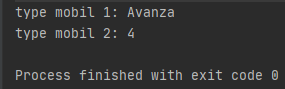
| public class MobilAvanza extends Mobil{  public String deskripsi;  public String type; } |
| --- |

Class *MobilAvanza* adalah class anak , sedangkan class Mobil class induk. Itu ditandai dengan adanya keyword extends di class *MobilAvanza.*

## Contoh penggunaan Method Overloading

| public static String typeMobil(String typeMobil){  return typeMobil; }  public static int typeMobil(int jumlahRoda){  return jumlahRoda; } public static void main(String[] args){  System.*out*.println("type mobil 1: "+*typeMobil*("Avanza"));  System.*out*.println("type mobil 2: "+*typeMobil*(4)); } |
| --- |

Terlihat pada kode program diatas, typeMobil memiliki nama yang sama, namun prameter type datanya berbeda. Sehingga method ini dikatakan sebagai method overloading. Jika di run, outputnya seperti dibawah ini :



Gambar Output Method Overloading

## Contoh penggunaan Method Overriding

Tujuan overriding yaitu sub class ingin mengimplementasikan atau menerapkannya sendiri.

Class Mobil

| public class Mobil {  ...  public void doMelaju(Double kecepatan){  System.*out*.println("Kecepatan Induk : "+kecepatan); }  ...  } |
| --- |

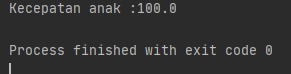
Class MobilAvanza

| public class MobilAvanza extends Mobil {  ...  @Override public void doMelaju(Double kecepatan) {  System.*out*.println("Kecepatan anak :"+kecepatan); }  ...  } |
| --- |

Class MainMobil: cetak ke console

| public class MainMobil {  public static MobilAvanza *objMobil* = new MobilAvanza();   public static void main(String[] args) {  *objMobil*.doMelaju(100.0);  }  } |
| --- |

Jika di run, output nya adalah:



Gambar Output Method Overriding

Seperti yang teman-teman lihat pada kode program diatas, kedua class(Mobil dan MobilAvanza) tersebut memiliki method void doMelaju(…). Class MobilAvanza memberikan implementasinya sendiri pada method doMelaju (…) tersebut.

Dengan kata lain meng-override method doMelaju (…) dari super class.

Sampai di sini tujuan overriding ini menjadi jelas, yaitu sub class ingin mengimplementasikan atau menerapkannya sendiri.

Kode Lengkap

Class Mobil

| package com.example.demo.model.inheritance; //Class public class Mobil {   //Variabel  public String type;  public int roda;  public Double kecepatan;   //Construktor  public Mobil(){};    public Mobil(String type, int roda, Double kecepatan) {  this.type = type;  this.roda = roda;  this.kecepatan = kecepatan;  }   //Method VOID  public void doMelaju(Double kecepatan){  System.*out*.println("Kecepatan Induk : "+kecepatan);  }   public void doKlakson(){  System.*out*.println("klakson");  }   public void doBelok(String arah){  System.*out*.println("Belok ke arah "+arah);  }   //Return Method  public double hitungKecepatan(Double kecepatan, int waktuDetik){  return kecepatan/waktuDetik;  }   //Setter and Getter  public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public int getRoda() {  return roda;  }   public void setRoda(int roda) {  this.roda = roda;  }   public Double getKecepatan() {  return kecepatan;  }   public void setKecepatan(Double kecepatan) {  this.kecepatan = kecepatan;  } } |
| --- |

Class Mobil Avanza

| package com.example.demo.model.inheritance;  public class MobilAvanza extends Mobil {  public String deskripsi;  public String type;   public String getDeskripsi() {  return deskripsi;  }   public void setDeskripsi(String deskripsi) {  this.deskripsi = deskripsi;  }   public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public static String typeMobil(String typeMobil) {  return typeMobil;  }   public static int typeMobil(int jumlahRoda) {  return jumlahRoda;  }   @Override  public void doMelaju(Double kecepatan) {  System.*out*.println("Kecepatan anak :"+kecepatan);  }    // public static void main(String[] args){ // System.out.println("type mobil 1: "+typeMobil("Avanza")); // System.out.println("type mobil 2: "+typeMobil(4)); // }  } |
| --- |

Class MainMobil

| package com.example.demo.model.inheritance;  public class MainMobil {  public static MobilAvanza *objMobil* = new MobilAvanza();   public static void main(String[] args) {  *objMobil*.doMelaju(100.0);  }  } |
| --- |

## Keuntungan Ketika Overriding Method

Terdapat keuntungan ketika meng-override method yaitu super class selaku pemilik overriden method, kodenya tidak perlu mengalami perubahan sama sekali, sementara itu di sisi lainnya sub class dapat mengimplementasikan kode tersebut sesuai dengan kebutuhan.

Bisa dibayangkan jika suatu superclass memiliki banyak subclass. Ketika sub class – sub class tersebut perlu untuk menggunakan method dari super class, mereka dapat menggunakannya karena sub class dapat menggunakan method super class pada konsep inheritance.

Dan jika mereka perlu mengimplementasikannya dengan berbeda mereka tinggal meng-override method dari super class tanpa sama sekali menyentuh kode method dari super class.

## Aturan-aturan Pada Overriding Method

Berikut ini adalah aturan-aturan pada overriding method yang perlu untuk anda ketahui.

* List dari argumen pada overriden dan overriding method harus benar-benar sama, termasuk tipe data dan urutan pada argumen tersebut.
* Acces modifier pada overriding method tidak bisa lebih ketat daripada access modifier di overriden method.
* Jika suatu class inherit dari class abstract atau mengimplementasikan interface, maka class tersebut harus meng-override semua method abstract kecuali class itu sendiri adalah class abstract.
* Method static, private dan final tidak bisa di-override. Namun method static bisa dideklarasikan ulang pada subclass.
* Overriding method harus memiliki tipe return (sub type) yang sama
* Jika method tidak bisa diwariskan, maka tidak bisa di-override
* Sub class yang berada pada package yang berbeda hanya dapat meng-override method non-final yang dideklarasikan sebagai public atau protected
* Suatu constructor tidak bisa di-override
* Overriding method tidak boleh melontarkan checked exception yang baru atau yang lebih luas.
* Untuk memanggil overriden method melalui sub class dapat menggunakan keyword super.

## Contoh penggunaan Keyword Super

Keyword super akan mengacu pada superclass.

Nah, fungsi dari keyword super itu sendiri adalah:

* Memanggil constructor dari superclass
* Memanggil method dari superclass
* Mengakses variabel instance super class

## Memanggil Conctructor Dari Superclass

Constructor digunakan untuk membentuk instance dari suatu class. Namun, tidak seperti properti dan method, constructor ini tidak diwariskan pada subclass.

Sehingga constructor dari subclass ini hanya dapat dipanggil melalui constructor dari sub class dengan menggunakan keyword super.

Syntax nya adalah:

* super()  
  Akan memanggil constructor tanpa argumen dari superclass
* super(parameter)  
  Memanggil constructor dari superclass yang memiliki argumen-argumen yang sesuai.

Pernyataan super() ataupun super(parameter) harus ditempatkan pertama kali di dalam constructor dari sub class.

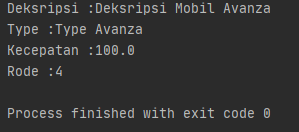
Dan ini adalah satu-satunya cara untuk memanggil constructor super class melalui subclass.

Contoh:

| public class MobilAvanza extends Mobil {  public String deskripsi;  public String type;   public MobilAvanza(String deskripsi, String type) {  super("type dari induk", 4, 100.0);  this.deskripsi = deskripsi;  this.type = type;  }  ...   public static void main(String[] args){  MobilAvanza obj = new MobilAvanza("Deksripsi Mobil Avanza", "Type Avanza");  System.*out*.println("Deksripsi :"+obj.getDeskripsi());  System.*out*.println("Type :"+obj.getType());  System.*out*.println("Kecepatan :"+obj.getKecepatan());  System.*out*.println("Rode :"+obj.getRoda());  }  } |
| --- |

Terlihat dari kode program diatas, penggunaan kelas super pada construktor kelas induk. super("type dari induk", 4, 100.0); .

Jika di run, outputnya adalah :



Gambar output Keyword Super Construktor

## Memanggil Method Milik Superclass

Syntax

| *super.method(parameters);* |
| --- |

Contoh

Class MobilAvanza

| ...  @Override public void doMelaju(Double kecepatan) {  super.doMelaju(kecepatan);  System.*out*.println("Kecepatan anak :"+kecepatan); }  ... |
| --- |

Class MainMobil

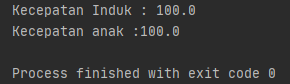
| public class MainMobil {  public static MobilAvanza *objMobil* = new MobilAvanza();   public static void main(String[] args) {  *objMobil*.doMelaju(100.0);  }  } |
| --- |

Class Mobil

| ...  public void doMelaju(Double kecepatan){  System.*out*.println("Kecepatan Induk : "+kecepatan); }  ... |
| --- |

Perhatikan pada class MobilAvanza, ada penambahan keyword super.doMelaju(kecepatan);. Artinya itu adalah memanggil method induknya.

Maka Output



Gambar Output Super Method

Kode Lengkap

Mobil

| package com.example.demo.model.inheritance; //Class public class Mobil {   //Variabel  public String type;  public int roda;  public Double kecepatan;   //Construktor  public Mobil(){};    public Mobil(String type, int roda, Double kecepatan) {  this.type = type;  this.roda = roda;  this.kecepatan = kecepatan;  }   //Method VOID  public void doMelaju(Double kecepatan){  System.*out*.println("Kecepatan Induk : "+kecepatan);  }   public void doKlakson(){  System.*out*.println("klakson");  }   public void doBelok(String arah){  System.*out*.println("Belok ke arah "+arah);  }   //Return Method  public double hitungKecepatan(Double kecepatan, int waktuDetik){  return kecepatan/waktuDetik;  }   //Setter and Getter  public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public int getRoda() {  return roda;  }   public void setRoda(int roda) {  this.roda = roda;  }   public Double getKecepatan() {  return kecepatan;  }   public void setKecepatan(Double kecepatan) {  this.kecepatan = kecepatan;  } } |
| --- |

MobilAvanza

| package com.example.demo.model.inheritance;  public class MobilAvanza extends Mobil {  public String deskripsi;  public String type;   public MobilAvanza(){};   public MobilAvanza(String deskripsi, String type) {  super("type dari induk", 4, 100.0);  this.deskripsi = deskripsi;  this.type = type;  }   public String getDeskripsi() {  return deskripsi;  }   public void setDeskripsi(String deskripsi) {  this.deskripsi = deskripsi;  }   public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public static String typeMobil(String typeMobil) {  return typeMobil;  }   public static int typeMobil(int jumlahRoda) {  return jumlahRoda;  }   @Override  public void doMelaju(Double kecepatan) {  super.doMelaju(kecepatan);  System.*out*.println("Kecepatan anak :"+kecepatan);  }    public static void main(String[] args){  MobilAvanza obj = new MobilAvanza("Deksripsi Mobil Avanza", "Type Avanza");  System.*out*.println("Deksripsi :"+obj.getDeskripsi());  System.*out*.println("Type :"+obj.getType());  System.*out*.println("Kecepatan :"+obj.getKecepatan());  System.*out*.println("Rode :"+obj.getRoda());  }  } |
| --- |

MainMobil

| package com.example.demo.model.inheritance;  public class MainMobil {  public static MobilAvanza *objMobil* = new MobilAvanza();   public static void main(String[] args) {  *objMobil*.doMelaju(100.0);  }  } |
| --- |

## Digunakan Untuk Mengacu Secara Langsung Pada Variabel Instance Milik Super Class

Anda dapat menggunakan keyword super untuk langsung mengakses data member atau field dari super class. Dan ini digunakan jika super class dan sub class memiliki field-field yang sama.

Contoh

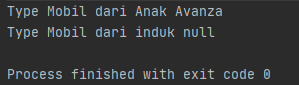
Class MobilAvanza

| ...  public void superVariabel(String type) {  System.*out*.println("Type Mobil dari Anak " + type);  System.*out*.println("Type Mobil dari induk " + super.type); }  ... |
| --- |

Class MainMobil

| public class MainMobil {  public static MobilAvanza *objMobil* = new MobilAvanza();   public static void main(String[] args) { // objMobil.doMelaju(100.0);  *objMobil*.superVariabel("Avanza");  }  } |
| --- |

Output:



Gambar Output Super Variabel Instance

Dapat dilihat ouput dari indunya adalah null, karena dengan menggunakan keyword super, akan memanggil variabel dari class induknya.

Kode Program lengkap

Class Mobil

| package com.example.demo.model.inheritance; //Class public class Mobil {   //Variabel  public String type;  public int roda;  public Double kecepatan;   //Construktor  public Mobil(){};    public Mobil(String type, int roda, Double kecepatan) {  this.type = type;  this.roda = roda;  this.kecepatan = kecepatan;  }   //Method VOID  public void doMelaju(Double kecepatan){  System.*out*.println("Kecepatan Induk : "+kecepatan);  }   public void doKlakson(){  System.*out*.println("klakson");  }   public void doBelok(String arah){  System.*out*.println("Belok ke arah "+arah);  }   //Return Method  public double hitungKecepatan(Double kecepatan, int waktuDetik){  return kecepatan/waktuDetik;  }   //Setter and Getter  public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public int getRoda() {  return roda;  }   public void setRoda(int roda) {  this.roda = roda;  }   public Double getKecepatan() {  return kecepatan;  }   public void setKecepatan(Double kecepatan) {  this.kecepatan = kecepatan;  } } |
| --- |

Class MobilAvanza

| package com.example.demo.model.inheritance;  public class MobilAvanza extends Mobil {  public String deskripsi;  public String type;   public MobilAvanza(){};   public MobilAvanza(String deskripsi, String type) {  super("type dari induk", 4, 100.0);  this.deskripsi = deskripsi;  this.type = type;  }   public String getDeskripsi() {  return deskripsi;  }   public void setDeskripsi(String deskripsi) {  this.deskripsi = deskripsi;  }   public String getType() {  return type;  }   public void setType(String type) {  this.type = type;  }   public static String typeMobil(String typeMobil) {  return typeMobil;  }   public static int typeMobil(int jumlahRoda) {  return jumlahRoda;  }   @Override  public void doMelaju(Double kecepatan) {  super.doMelaju(kecepatan);  System.*out*.println("Kecepatan anak :"+kecepatan);  }   public void superVariabel(String type) {  System.*out*.println("Type Mobil dari Anak " + type);  System.*out*.println("Type Mobil dari induk " + super.type);  }     public static void main(String[] args){  MobilAvanza obj = new MobilAvanza("Deksripsi Mobil Avanza", "Type Avanza");  System.*out*.println("Deksripsi :"+obj.getDeskripsi());  System.*out*.println("Type :"+obj.getType());  System.*out*.println("Kecepatan :"+obj.getKecepatan());  System.*out*.println("Rode :"+obj.getRoda());    }  } |
| --- |

Class MainMobil

| package com.example.demo.model.inheritance;  public class MainMobil {  public static MobilAvanza *objMobil* = new MobilAvanza();   public static void main(String[] args) { // objMobil.doMelaju(100.0);  *objMobil*.superVariabel("Avanza");  }  } |
| --- |

# Polymorphism

## Apa itu Polymorphism

Poly artinya banyak, morfisme artinya bentuk.

Polimorfisme (bahasa inggris polymorphism) adalah sebuah prinsip dalam biologi di mana oraganisme atau spesias dapat memiliki banyak bentuk atau tahapan (stages).

Polimorfisme dalam OOP adalah sebuah prinsip di mana class dapat memiliki banyak “bentuk” method yang berbeda-beda meskipun namanya sama.

“Bentuk” di sini dapat kita artikan: isinya berbeda, parameternya berbeda, dan tipe datanya berbeda.

## Jenis Polymorphism

Polimorfisme pada Java ada dua macam:

* Static Polymorphism (Polimorfisme statis);
* Dynamic Polymorphism (Polimorfisme dinamis).

Polimorfisme statis menggunakan method overloading

Sedangkan polimorfisme dinamis menggunakan method overriding. Method everriding telah dibahas di inheritance, silahkan refer kesana ya.

## Overloading

Overloading merupakan fungsionalitas yang memungkinkan kita untuk mendefinisikan beberapa method yang mempunyai nama sama tetapi memiliki body method yang berbeda

Pada method overloading perbedaan parameter mencakup :

* Jumlah parameter
* Tipe data dari parameter
* Urutan dari tipe data parameter

Contoh

| public class ClassOVerloading {   String laptop(){  return "1";  }  String laptop(String a){  return a;  }   int laptop(int a){  return 1;  }  int laptop(int a,String b){  return 1;  } }  Class ClassOVerloading memiliki dua method yang namanya sama, yakni laptop().  Tapi parameter dan tipe datanya berbeda..  ..dan juga isi atau rumus di dalamnya berbeda.  Inilah yang disebut polimorfisme statis. |
| --- |

Plimorfisme dinamis biasanya terjadi saat kita menggunakan [pewarisan *(inheritance)*](https://www.petanikode.com/java-oop-inheritance/) dan implementasi *interface*.

# Abstraction

Class abstrak adalah class yang masih dalam bentuk abstrak. Karena bentuknya masih abstrak, dia tidak bisa dibuat langsung menjadi objek.

Sebuah class agar dapat disebut class abstrak setidaknya memiliki satu atau lebih method abstrak.

Method abstrak itu adalah method yang tidak memiliki isi. Cuma ada nama saja

Contoh

| // ini abstrak method  void sayHello();  // ini bukan abstrak method karena  // punya implementasi di body method  void greeting(){  System.out.println("Hello Java");  } |
| --- |

## Mengapa Harus Pakai Class Abstrak?

Class abstrak memang belum bisa digunakan secara langsung.

Karena itu, agar class abstrak dapat digunakan, maka ia harus dibuat bentuk konkritnya.

Cara membuat class abstrak menjadi konkrit adalah dengan membuat implementasi dari method-method yang masih abstrak.

Ini bisa kita lakukan dengan [pewarisan (inheritance)](https://www.petanikode.com/java-oop-inheritance/).

Class abstrak biasanya digunakan sebagai class induk dari class-class yang lain. Class anak akan membuat versi konkrit dari class abstrak.

Contoh

| abstract class abstraction {  public abstract void mData(String name, String nim);   public void mBiodata(){  System.*out*.println("biodata saya");  } } |
| --- |

Berikut ini adalah contoh class yang menggunakan Abstraction . terdapat 2 buat method, yaitu mData dan mBiodata. Pada mData terdapat kata abstract itu menunjukan bahwa method itu merupakan method abstract dan harus di override pada class turunanya.

| public class Implementasi extends abstraction {  @Override  public void mData(String name, String nim) {  System.*out*.println("nama="+name);  System.*out*.println("nim="+nim);  } } |
| --- |

ini dia kelas turunan dari class abstrac diatas. Dikarenakan class ini merupakan extends dari class abstrac Biodata makan method abstract pun di implemetasikan di class ini.

| public class ClassMain {  public static void main(String[] args) {  Implementasi imp = new Implementasi();  String nama ="nama saya";  String nim ="1234";   imp.mBiodata();  imp.mData(nama,nim);  } }  output |
| --- |

Pada  imp.mData(nama,nim), berikut adalah untuk memanggil method mData pada class imp atau Implemetasi beserta parameter sesuia kebutuhannya.

Secara singkatnya, ketika kita menggunakan abstraction pada sebuah class dan method nya, dan ketika kita turunkan . Maka class turunanya wajib menyertakan method abstrac melalui override.

# Gerbace Collector

Garbage Collection adalah mekanisme Java untuk menghapus suatu objek dari memori tanpa perlu dideklarasikan secara eksplisit dalam program. Berfungsi untuk meningkatkan Management Memory yaitu kita dapat menghemat penggunaan memori, java akan menghapus sebuah objek yang tidak diperlukan atau tidak di referensikan lagi, ruang memori pada objek tersebut akan dimanfaatkan untuk keperluan lain sehingga tidak ada pemborosan memori. Garbage Collection merupakan salah satu mekanisme dari fitur JVM (Java Virtual Machine). Ini adalah salah satu kelebihan dari bahasa pemrograman Java.

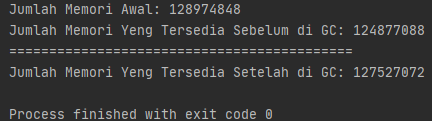
Pada materi berikut ini kita akan belajar mengenai fungsi Garbage Collection pada Java, dengan contoh program sederhana yang menggambarkan bagaimana Garbage Collection tersebut berfungsi.

## Garbage Collection pada Java

Sebagai seorang programmer yang handal, aplikasi atau program yang kita buat harus memiliki penggunaan memori yang kecil atau seminimal mungkin, karena dengan begitu, program yang kita buatpun menjadi lebih handal juga.  
  
Salah satu kelebihan dari pemrograman Java adalah kita dapat menigkatkan efisiensi pada memori, dengan membuang objek yang tidak terpakai atau tidak direferensikan, ini akan secara otomatis dikontrol oleh JVM (Java Virtual Machine).  
  
Untuk melihat bagaimana cara kerja Garbage Collection pada java, kita dapat menambahkan codingan, seperti method finalize() dan juga System.gc(). Contoh Sederhananya seperti berikut ini:

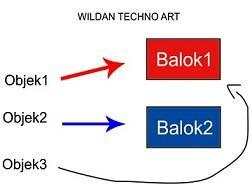
| public class GarbaceCollectorClass {  //Method ini akan ditampilkan jika Garbage Collection bekerja  public void finalize(){  System.*out*.println("Objek Yang Tidak Terpakai Sudah Dibersihkan:");  }   public static void main(String[] args){  Runtime RT = Runtime.*getRuntime*();  System.*out*.println("Jumlah Memori Awal: "+RT.totalMemory());   GarbaceCollectorClass objek1 = new GarbaceCollectorClass(); //Block Satu  GarbaceCollectorClass objek2 = new GarbaceCollectorClass(); //Block Dua  GarbaceCollectorClass objek3 = objek1; //Objek3 Mengembalikan Nilai Objek1  objek1 = null;  objek2 = null;  System.*out*.println("Jumlah Memori Yeng Tersedia Sebelum di GC: "+RT.freeMemory());  System.*gc*();  System.*out*.println("===========================================");  System.*out*.println("Jumlah Memori Yeng Tersedia Setelah di GC: "+RT.freeMemory());  } } |
| --- |

Output:

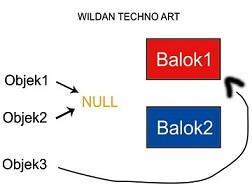


Gambar Output Gerbage Collector

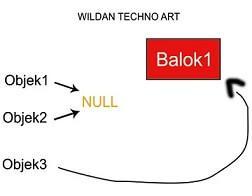
Penjelasan:  
Pada program tersebut, disana kita membuat 3 buah objek atau variable dari Class GarbaceCollectorClass, objek1 dan objek2 diinisialisasikan kedalam objek1 dan objek2, dan objek3 mempunyai nilai kembali dari objek GarbaceCollectorClass1.



Pada baris selanjutnya, objek1 dan objek2 kita ubah nilainnya menjadi null yang berarti kosong, sedangkan objek3 masih mengembalikan nilainya objek1.



Karena pada GarbaceCollectorClass2 tidak ada yang direferensikan nilainya, maka secara otomatis, Balok2 tersebut akan di bawa oleh System Garbage Collection pada JVM.



System.gc(), berfungsi untuk menjalankan Garbage Collection pada java. lalu Method finalize(), digunakan untuk memastikan bahwa objek telah bersih dan Garbage Collection telah bekerja, Di dalam metode finalize(), kita dapat menentukan tindakan yang harus dilakukan sebelum suatu objek hancur.  
  
Dan terakhir, kita menambahkan sebuah variable Runtime, yang digunakan untuk mengecek Total Memori serta Jumlah Memori yang tersedia sebelum dan sesudah di Garbage Collection.

# Daftar Pustaka

https://www.anbidev.com/python-oop-intro/

<https://idcloudhost.com/panduan/mengenal-pengertian-dan-konsep-oop/>

<https://www.anbidev.com/python-oop-intro/>

<https://www.jagoanhosting.com/blog/mengenal-oop-encapsulation-inheritance-polyorphism-dan-abstrak-class/>

<http://imamfarisi.com/encapsulation-di-java/>

<https://andriansyahdoni.wordpress.com/2016/12/23/encapsulation-in-java-programming-using-netbeans/>

<https://sis.binus.ac.id/2017/09/07/inheritance/>

<http://frieyadie.web.id/constructor-pada-java/>

<https://lecturer.ppns.ac.id/khoirul/method-class-object/>

<https://bahasajava.com/method-overriding-java/>

<https://bahasajava.com/keyword-super-program-java/>

https://www.wildantechnoart.net/2017/11/fungsi-garbage-collection-pada-java.html