Materi Pelatihan

Java Fundamental

Endy Muhardin
endy@artivisi.com
http://endy.artivisi.com

http://www.artivisi.com

Jakarta, 2008

Table of Contents

1.OOP dengan Java

Pada bab sebelumnya, kita telah mengetahui bagaimana cara menulis program dalam bahasa Java. Para pembaca yang sebelumnya bekerja dengan bahasa pemrograman lain telah melihat bagaimana teknik yang ada di VB, PHP, atau Pascal diterapkan dalam Java.

Dalam bab ini, kita akan berpikir dalam level yang lebih tinggi. Yaitu bagaimana cara membuat aplikasi dengan Java.

Pemrograman Berorientasi Objek

Pemrograman berorientasi objek merupakan sebuah cara berpikir. Selain pendekatan Object Oriented (Object Oriented Programming), kita mengenal pendekatan prosedural (Procedural Programming).

Dalam pendekatan struktural, kita berpikir tentang prosedur, fungsi, dan data. Kita mendefinisikan sekumpulan data, kemudian menentukan fungsi dan prosedur yang akan memanipulasi data tersebut.

Dengan menggunakan pendekatan OOP, kita berpikir tentang object. Sebuah object memiliki sekumpulan data (property) dan fungsi (method). Pendekatan OOP memungkinkan kita untuk memodelkan sistem yang akan dibangun dengan menggunakan istilah yang sama seperti sistem aslinya.

Objek dibuat dari class. Dengan kata lain, class merupakan cetakan object. Dari class `Person`, kita dapat membuat object `endy`, `anton`, `maya`, dan seterusnya.

penting

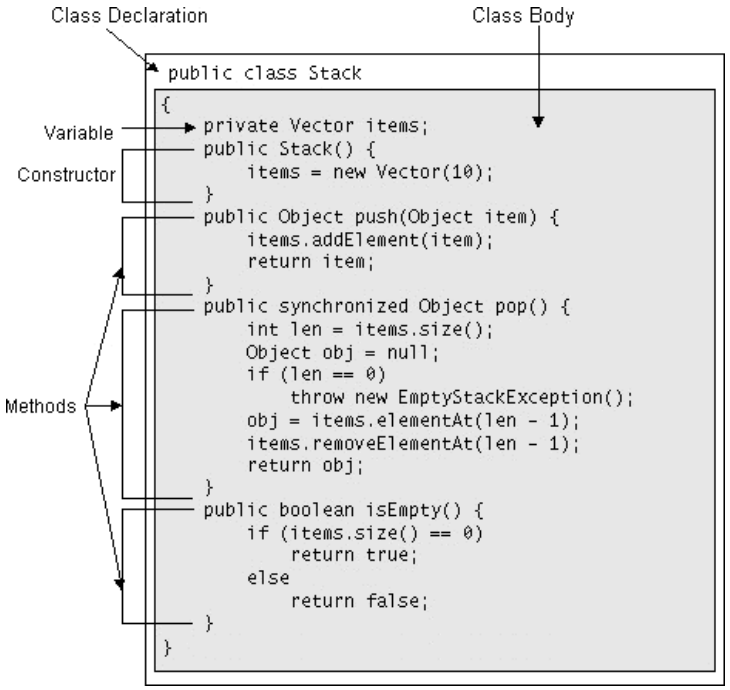
Dalam bahasa prosedural, tipe data yang tersedia terbatas pada apa yang disediakan oleh bahasa tersebut.

Dalam bahasa berorientasi objek, kita dapat mendefinisikan tipe data sendiri.

Tipe data baru ini disebut dengan istilah class.

Anatomi Class

Secara garis besar, anatomi class dapat digambarkan sebagai berikut (gambar diambil dari Java Tutorial)



Seperti kita lihat pada gambar di atas, class memiliki beberapa jenis anggota sebagai berikut:

- Variable atau property
- Methods

Untuk membuat class, kita perlu mendeklarasikan class. Contoh deklarasi class yang lengkap dapat dilihat pada contoh kode berikut.

```
public class Customer extends Person
                        implements Serializable {
}
```

Ada beberapa keyword yang perlu diketahui:

1. public: menyatakan ijin akses class. Public class bisa diakses dari mana saja, sedangkan bila keyword public dihilangkan, class hanya bisa diakses oleh class lain yang berada dalam package yang sama.
2. extends: keyword ini opsional, digunakan untuk menentukan class lain yang merupakan parent-class (superclass) dari class yang ingin kita deklarasikan. Di Java, kita hanya bisa extends dari satu superclass saja.
3. implements: keyword ini juga opsional. Digunakan untuk menentukan interface lain yang merupakan superclass. Di Java, kita bisa implements banyak interface, masing-masingnya dipisahkan dengan koma.
4. abstract: menandai class yang tidak bisa diinstankan (tidak bisa dibuat objectnya).
5. final: menandai class yang tidak bisa di-extend.

Sebagai contoh, dalam membuat aplikasi pergudangan, kita memiliki istilah Gudang atau Warehouse. Sebuah gudang memiliki keterangan sebagai berikut:

- Kode
- Nama
- Kapasitas
- Lokasi

Class tersebut dapat kita representasikan dalam Java sebagai

berikut.

```
public class Warehouse {
  public String code;
  public String name;
  public String location;
  public Double capacity;
}
```

penting

Membuat object dalam bahasa Inggris disebut dengan "instantiate".

Object yang dihasilkan disebut "instance".

Pada buku ini, instantiate akan diterjemahkan menjadi "menginstankan".

Class tersebut dapat digunakan untuk membuat object. Misalnya Kita memiliki gudang g120 yang berlokasi di Jakarta Utara. Maka kita akan membuat objectnya sebagai berikut.

```
Warehouse g120 = new Warehouse();
g120.location = "Jakarta Utara";
```

Pada contoh kode di atas, kita membuat object kosong yangkemudian kita isi property locationnya dengan nilai `Jakarta Utara`.

Constructor

Kita juga bisa langsung menginstankan object dengan sekumpulan nilai, misalnya seperti ini.

```
Warehouse g120 = new Warehouse("g120", "Gudang 120",
"Jakarta Utara", 12000.0);
```

Untuk dapat menginstankan object seperti di atas, kita harus menambahkan constructor di class Warehouse yang menerima masukan (`String, String, String, Double`) sesuai dengan property yang ingin langsung kita tentukan.

Class Warehouse menjadi seperti ini.

```
public class Warehouse {
  public String code;
  public String name;
  public String location;
  public Double capacity;

  public Warehouse(String code, String name,
                   String location, Double capacity) {
    this.code = code;
    this.name = name;
    this.location = location;
    this.capacity = capacity;
  }
}
```

penting

Pada contoh kode di samping, kita melihat ada keyword this.

Keyword this tersebut digunakan untuk mengacu pada variabel yang didefinisikan dalam class, bukan variabel yang ada di argumen constructor.

this.code = code berarti variabel code yang didefinisikan di class diberikan nilai yang sama dengan variabel code dalam argumen method

Perlu diperhatikan bahwa dengan membuat constructor seperti di atas, kita tidak bisa lagi membuat object seperti ini.

```
Warehouse g120 = new Warehouse();
```

Mengapa begitu?

Jawabannya adalah, bila kita tidak mendefinisikan constructor, seperti pada kode Warehouse pertama kali tadi, Java secara default akan membuatkan constructor kosong buat kita. Jadi, bila kita membuat class seperti ini:

```
public class Warehouse {
}
```

Sama dengan kita menulis seperti ini:

```
public class Warehouse {
  public Warehouse() {
  }
}
```

Begitu kita definisikan constructor, Java tidak lagi membuatkan constructor kosong. Sehingga bila kita tetap ingin membuat object kosong, kita harus definisikan sendiri constructor kosong. Class Warehouse menjadi seperti ini.

```
public class Warehouse {
  public Warehouse() {
  }
  public Warehouse(String code, String name,
                   String location, Double capacity) {
    this.code = code;
    this.name = name;
    this.location = location;
    this.capacity = capacity;
  }
}
```

Jadi, saat ini kita sudah memahami bahwa kode ini

```
Warehouse g120 = new Warehouse();
```

artinya adalah, kita membuat object g120 yang bertipe Warehouse dengan menggunakan constructor tanpa argumen.

Sekarang mari kita dalami lebih jauh tentang class dan object.

Class dan Object

Pada bagian ini, kita akan belajar tentang keyword `static`. Keyword ini menyatakan bahwa variabel atau method tertentu adalah milik class. Variabel/method lain yang tidak memiliki keyword ini (non-static) adalah milik object.

Untuk memperjelas konsep di atas, mari kita lihat contoh kode berikut. Masih tentang warehouse.

```
public class Warehouse {
  private String name;
  private static int totalWarehouse = 0;
  public Warehouse(String name) {
    this.name = name;
    totalWarehouse++;
  }
  public void info() {
    System.out.println("Nama gudang: "+name);
    System.out.println("Jumlah gudang: "+totalWarehouse);
  }
}
```

Seperti kita lihat, ada dua variabel di sana. Yang pertama variabel non-static name, sedangkan satu lagi adalah variabel static totalWarehouse.

Sekarang, mari kita buat kode yang menggunakan class Warehouse dan memanggil method info.

```
Warehouse g120 = new Warehouse("Gudang 120");
g120.info();
Warehouse g4 = new Warehouse("Gudang 4");
g4.info();
g120.info();
```

Kita menampilkan informasi tentang `g120` dua kali. Ini untuk menunjukkan kondisi object `g120` setelah `g4` dibuat.

Bila dijalankan, kode di atas akan mengeluarkan hasil seperti ini.

```
Nama gudang: Gudang 120
Jumlah gudang: 1
Nama gudang: Gudang 4
Jumlah gudang: 2
Nama gudang: Gudang 120
Jumlah gudang: 2
```

Kita lihat pada output bahwa isi variabel name pada `g120` tidak berubah setelah `g4` diinstankan. Berbeda dengan variabel `totalWarehouse` yang nilainya bertambah.

penting

Non-static variable disebut juga dengan istilah instance variable.

Jadi jangan bingung bila menemui istilah instance variable pada referensi lain.

Ini menunjukkan bahwa variabel name adalah milik masing-masing object. Nilainya tidak dipengaruhi oleh object lain dari class yang sama.

Variabel `static` adalah milik class. Nilainya sama untuk seluruh object dari class yang sama. Bila salah satu object mengubah nilainya, maka object lain dari class yang sama juga akan mengalami perubahan tersebut.

Property

penting

Property sering juga disebut dengan instance variable atau member variable

Dalam class, kita mendefinisikan property. Sebetulnya property sama dengan instance variable. Jadi seharusnya kita sudah tidak asing dengan konsep property.

Sekarang mari kita lihat teknis penggunaan property. Property dideklarasikan seperti ini:

```
public class Warehouse {
  private String name;
  public transient Connection databaseConnection;
  public static final int ONE_MINUTE = 60;
}
```

Ada beberapa keyword yang digunakan dalam deklarasi property, yaitu:

- Ijin akses. Ada berbagai pilihan ijin akses yang bisa digunakan; public, protected, package, atau private. Topik ini akan dibahas tersendiri pada bagian selanjutnya.
- static. Sudah dibahas di atas, keyword ini digunakan untuk menandai variabel milik class.
- final. Menyatakan bahwa sebuah variabel nilainya tidak boleh berubah. Biasanya keyword ini (bersama dengan static) digunakan untuk membuat konstanta.
- transient. Keyword ini biasanya digunakan dalam Serialization atau Persistence. Artinya, apabila object akan dikirim ke komputer lain atau disimpan ke database, variabel transient ini tidak akan diikut sertakan. Contohnya adalah koneksi ke database atau koneksi socket (TCP/IP) ke komputer lain.
- volatile. Keyword ini digunakan untuk melarang compiler melakukan optimasi pada variabel ini. Keyword ini merupakan fitur tingkat lanjut dan tidak untuk konsumsi pemula.

Method

Pada contoh kode di atas, kita telah melihat contoh method. Untuk menjelaskan tentang deklarasi method, mari kita lihat method yang sering kita lihat tapi belum kita mengerti maksudnya.

penting

Pada bahasa prosedural, method sering disebut function atau procedure.

Di Java, function dan procedure tidak dibedakan, keduanya disebut dengan istilah method.

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println(“Hello World”);
  }
}
```

Ada beberapa keyword yang dapat digunakan pada deklarasi method:

- Ijin akses
- static
- abstract: untuk membuat method yang tidak memiliki implementasi. Keyword ini akan kita bahas lebih lanjut dalam bahasan tentang inheritance.
- final: method final tidak dapat di-override. Konsep override dijelaskan dalam bahasan tentang inheritance.
- native: method native diimplementasikan dalam bahasa lain (misalnya C)
- synchronized: keyword ini digunakan dalam lingkungan multi-threaded, untuk memastikan method diakses bergantian oleh masing-masing thread.

Method harus memiliki return type, yaitu tipe data yang dihasilkan oleh method. Coba lihat contoh kode berikut.

```
public class Calculator {
  public int add(int x, int y) {
    return x + y;
  }
}
```

Method `add` di atas, menerima dua argumen, keduanya integer. Setelah melakukan pertambahan, hasilnya dikeluarkan untuk baris

kode yang memanggilnya. Karena `x` dan `y` bertipe integer, maka hasil penjumlahannya juga bertipe integer. Oleh karena itu, return type dari method `add` adalah integer.

Ada kalanya kita membuat method yang tidak mengembalikan nilai apa-apa. Method seperti ini sering disebut prosedur pada bahasa pemrograman lain. Method seperti ini dideklarasikan dengan return type `void`. Berikut adalah contoh method `void`.

```
public void hello() {
  System.out.println("Hello World");
}
```

Method `hello` di atas hanya menampilkan tulisan ke layar, kemudian selesai. Tidak ada nilai yang dihasilkan dari method tersebut.

Sebuah method dapat menerima masukan. Method `add` di atas menerima dua masukan, masing-masingnya bertipe integer. Di Java, method dapat menerima tipe data apapun sebagai argumen method, tapi tidak bisa menerima function pointer. Dalam bahasa lain seperti JavaScript atau Ruby, kita bisa mengirim function menjadi argumen, tapi tidak di Java. Untuk melakukan hal yang sama, kita dapat menggunakan teknik yang disebut *anonymous inner class*.

penting

Masukan untuk method sering juga disebut dengan istilah argument dan parameter.

Overloading

Di Java, kita bisa mendefinisikan beberapa method dengan nama yang sama, asalkan daftar argumennya berbeda. Sebagai contoh, kode seperti ini diperbolehkan.

```
public void hello(String message) {
  System.out.println(message);
}
public void hello(String message, String name) {
  System.out.println(message + name);
}
```

Tapi seperti ini tidak diperbolehkan. Karena argumen method memiliki tipe data yang sama (String,String).

```
public void hello(String message, String name) {
  System.out.println(message + name);
}
public void hello(String name, String message) {
  System.out.println(message + name);
}
```

Mengelompokkan Class dalam Package

Bila kita membuat aplikasi, jumlah class yang kita miliki pasti akan besar jumlahnya. Untuk aplikasi kecil saja, jumlah class bisa mencapai puluhan atau bahkan ratusan. Tanpa pengaturan yang baik, bisa terjadi banyak kesulitan dalam mengelola class tersebut.

Salah satu permasalahan yang sering muncul adalah penamaan class. Semakin banyak class yang kita definisikan, semakin sulit kita mencari nama untuk class. Pada akhirnya, penggunaan nama yang sama tidak terhindarkan.

Di Java, masalah ini diatasi dengan package. Kita dapat mengelompokkan beberapa class ke dalam package. Dalam package yang sama, tidak boleh ada class yang bernama sama. Jadi bila kita ingin membuat class bernama sama, kita bisa tempatkan class tersebut dalam package yang berbeda.

Nama package harus dideklarasikan di baris paling atas dari source

penting

Dalam bahasa pemrograman PHP, fitur package tidak ada.

Ini sangat menyulitkan programmer, karena nama class tidak boleh sama.

Bila kita menggunakan library orang lain, kita harus melihat nama class yang digunakannya agar kita tidak membuat class yang bernama sama.

Bayangkan kesulitan yang terjadi apabila jumlah library yang kita gunakan cukup banyak.

code. Berikut adalah contohnya.

```
package tutorial.database;
public class Session {
}
```

Class `Session` di atas dapat juga disebut dengan nama lengkapnya, yaitu `tutorial.database.Session`.

Bila kita menggunakan class yang ada di package berbeda, kita harus menyebut nama lengkapnya atau menggunakan `import`. Berikut adalah contoh kode yang menggunakan nama lengkap.

```
package com.artivisi.warehouse;
public class Employee {
  private java.util.Date joinDate;
  public java.util.Date getJoinDate() {
    return this.joinDate;
  }
}
```

Sedangkan bila kita menggunakan `import`, kode di atas dapat ditulis seperti ini.

```
package com.artivisi.warehouse;
import java.util.Date;
public class Employee {
  private Date joinDate;
  public Date getJoinDate() {
    return this.joinDate;
  }
}
```

Kode yang mendeklarasikan package, bila dikompilasi akan membentuk struktur folder sesuai dengan nama packagenya. Sebagai contoh, misalnya kita memiliki beberapa source code

sebagai berikut.

Class Employee.

```
package com.artivisi.warehouse;
public class Employee {}
```

Class EmployeeForm.

```
package com.artivisi.warehouse.ui.desktop;
public class EmployeeForm {}
```

Class EmployeeDao

```
package com.artivisi.warehouse.dao;
public interface EmployeeDao {}
```

Kode tersebut dapat dikompilasi dengan perintah berikut.

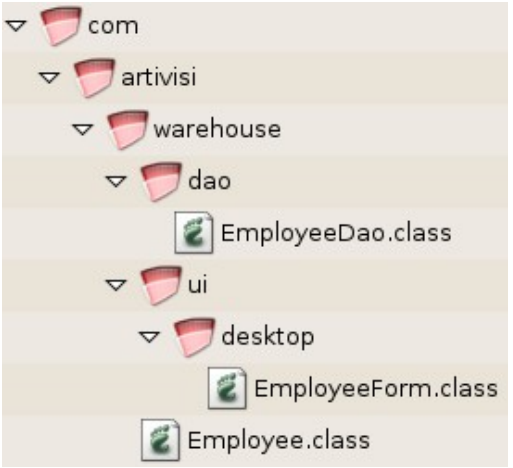
```
javac -d target-folder *.java
```

penting

Sebagai kesepakatan umum, nama package biasanya diambil dari nama domain internet yang kita miliki, dengan urutan dibalik.

Jadi bila perusahaan kita punya domain artivisi.com dan ingin membuat aplikasi akunting, maka kita bisa gunakan nama package: com.artivisi.accounting

Opsi -d menyatakan folder tujuan di mana hasil kompilasi akan diletakkan. Bila kita lihat isi folder tujuan, di dalamnya akan terlihat struktur seperti ini.



Seperti kita lihat, nama package menjadi nama folder. Masing-masing class diletakkan di dalam folder package yang sesuai.

Ijin Akses

Kita sudah melihat di mana ijin akses digunakan, yaitu dalam deklarasi method dan property. Tapi apa sebenarnya ijin akses itu?

Ijin akses digunakan untuk menerapkan encapsulation dalam desain aplikasi kita. Konsep encapsulation akan dibahas pada bagian selanjutnya. Ijin akses mengatur bagian mana dari class kita yang boleh diakses class lain. Mengakses di sini maksudnya adalah melihat nilai variabel, memanggil method, atau menginstankan class.

Berikut adalah berbagai ijin akses yang berlaku bagi class member dan penjelasannya.

- public. Member yang berijin akses public dapat diakses oleh siapa saja.
- private. Member dengan ijin akses private hanya bisa diakses di dalam class tempatnya berada.
- protected. Akses level protected mengijinkan class lain mengakses member, dengan catatan class tersebut adalah subclass atau berada dalam package yang sama dengan class tempat member berada.
- package. Bila kita tidak memberikan ijin akses, secara default Java akan menggunakan ijin package. Ijin ini memperbolehkan class yang berada di package yang sama mengakses class members. Tapi berbeda dengan protected, subclass tidak diijinkan mengakses member bila berada di package yang berbeda.

Untuk memperjelas konsep ijin akses di atas, perhatikan kedua class yang berada dalam package berbeda ini.

```
package com.artivisi.common;
import java.util.Date;
public class Person {
  public String name;
  protected Date birthdate;
  String address;
}
```

Class `Student` merupakan turunan dari `Person`, tapi berada di package yang berbeda. Kita akan mengakses member variable milik class `Person` di dalam constructor class `Student`.

```
package com.artivisi.academic;
public class Student extends Person {
  public Student() {
    System.out.println(name);
    // OK, public member bisa diakses di mana saja

    System.out.println(birthdate);
    // Error compile, protected member hanya bisa diakses
    // dalam package yang sama

    System.out.println(address);
    // Error compile, address hanya bisa diakses dalam
    // package yang sama.
  }
}
```

Ada satu perkecualian dengan ijin akses protected. Di atas disebutkan bahwa protected member bisa diakses oleh subclass. Akan tetapi, untuk subclass yang berada di package berbeda, kasusnya agak unik.

Seperti kita lihat di atas, kita tidak bisa langsung mengakses

variabel `birthdate`, karena class `Student` berada di package berbeda dengan `Person`. Tetapi, kita bisa mengakses variabel tersebut melalui class `Student`, karena class `Student` merupakan turunan `Person`. Lihat contoh kode berikut agar lebih jelas.

```
package com.artivisi.academic;
public class Student extends Person {
  public void coba (Person p, Student s) {
    System.out.println(p.birthdate); // Error
    System.out.println(s.birthdate); // OK
  }
}
```

Encapsulation

Apa itu encapsulation

Apabila kita belajar tentang pemrograman berorientasi objek, kita akan segera menemui istilah Encapsulation. Encapsulate dalam bahasa Indonesia kira-kira artinya membungkus. Apa yang dibungkus? Tentunya bukan kue.

Seni membuat program yang fleksibel terletak di “memisahkan yang berubah dari yang tetap”. Seni ini hanya dapat diperoleh dari jam terbang yang tinggi.

Dengan encapsulation, kita membungkus detail implementasi kode program dari mata pengguna. Pengguna di sini tentunya bukanlah pengguna aplikasi kita saja, tapi juga pengguna kode program kita.

Idealnya, kita hanya menunjukkan kepada pengguna cara menggunakan kode program kita, bukan bagaimana cara kita membuat kode program tersebut. Agar lebih mudah dipahami, mari kita lihat contoh berikut.

Misalnya kita membuat aplikasi buku alamat. Sebuah kontak dalam buku alamat tersebut direpresentasikan dengan class `Contact` berikut.

```
package com.artivisi.addressbook;
public class Contact {
  public String name;
  public String phone;
  public String email;
}
```

Kemudian, melalui sebuah tombol di tampilan aplikasi, kita menyediakan fitur untuk menyimpan data `Contact` tersebut.

Seperti kita ketahui, di masa modern seperti sekarang ini, ada banyak sekali cara menyimpan data. Kita bisa simpan di database, di text file dengan format sederhana, menggunakan XML, atau bahkan terintegrasi dengan buku alamat di handphone.

Untuk itu, mari kita definisikan fitur tersebut. Kita akan gunakan fasilitas interface yang ada di Java. Lebih lanjut tentang interface akan kita bahas pada bagian tersendiri.

penting

Di Java biasanya class yang bertanggung jawab terhadap akses database disebut dengan istilah DAO, yang merupakan singkatan dari Data Access Object

```
package com.artivisi.addressbook.dao;
public interface ContactDao {
  public void save(Contact contact);
}
```

Sampai di sini, kita cuma menyatakan bahwa ada `ContactDao` yang tugasnya menyimpan object `Contact` untuk kemudian bisa diambil lagi kapan-kapan. Bagaimana detail implementasi `ContactDao` tidak dijelaskan di sini.

Nantinya, bila kita membuat *user interface*, kodenya kira-kira seperti ini.

```
package com.artivisi.addressbook.ui;
public class ContactForm {
  private ContactDao contactDao;
  public void onButtonSaveClicked() {
    Contact con = new Contact();
    con.name = txtName.getText();
    contactDao.save(con);
  }
}
```

Dengan memisahkan antara interface (apa yang dilihat user) dengan implementasi, kita bisa lebih bebas dalam membuat tampilan.

Misalnya kita ingin menyimpan data `Contact` dalam database, maka kita buatkan implementasi `ContactDao` yang mengakses database.

```
package com.artivisi.addressbook.dao.mysql;
public class ContactDaoMySQL implements ContactDao {
  private Connection conn;
  public void setConnection(Connection connection) {
    this.connection = conn;
  }
  public void save(Contact contact) {
    String sql = "INSERT INTO contact VALUES (?)";
    PreparedStatements ps = conn.prepareStatement(sql);
    ps.setString(1, contact.name);
    ps.executeUpdate();
  }
}
```

Implementasi prinsip encapsulation di Java tidak berhenti sampai di sana. Programmer Java yang berpengalaman selalu

mendeklarasikan *instance variabel* dengan ijin akses *private*. Class `Contact` kita di atas dibuat seperti ini.

```
package com.artivisi.addressbook;
public class Contact {
  private String name;
  private String phone;
  private String email;
}
```

Lalu bagaimana instance variable tersebut diakses oleh class lain? Oleh `ContactForm` misalnya? Mudah, kita buatkan method untuk mengakses dan mengubah nilainya. Misalnya, untuk variabel `name`, kita buatkan dua method seperti ini.

```
public void setName(String name) {
  this.name = name;
}
public String getName() {
  return name;
}
```

Para programmer dari latar belakang bahasa yang lain tentu akan bertanya-tanya. Apakah ini tidak merepotkan? Bagaimana kalau kita punya 10 variabel? Berarti kita harus menulis 20 method? Lagipula, apa manfaat nyatanya?

Mari kita jawab satu persatu. Kerja keras dalam menulis getter/setter sekarang sudah dipermudah bila kita menggunakan editor yang canggih seperti Eclipse. Kita cukup melakukan klik kanan, dan pilih Generate Getters and Setters. Eclipse akan membuatkan semua method di atas untuk kita.

Banyak manfaat yang kita peroleh dalam menggunakan

penting

Method untuk mengakses instance variable disebut *accessor*, sedangkan method untuk mengubah variabel disebut *mutator*.

Accessor sering disebut juga dengan istilah getter method, sedangkan mutator sering disebut setter method.

Jadi, bila kita menemui istilah getter/setter, maksudnya adalah kedua method accessor/mutator

getter/setter, bahkan untuk variabel sederhana seperti `name` di atas. Ada framework bernama Hibernate yang sering digunakan untuk membantu kita mengakses database. Hibernate ini menggunakan teknik *lazy loading* dalam mengambil nilai dari database. Teknik ini dimungkinkan karena kita menggunakan getter/setter. Jika kita langsung memberikan akses public ke variabel kita tanpa getter/setter, Hibernate akan lebih sulit melakukan *lazy loading*.

Inheritance

Apa itu inheritance

Inheritance dalam bahasa Indonesia dapat diterjemahkan menjadi "warisan". Biasanya kita mengenal warisan ini sebagai komoditas utama yang dijual para produser sinetron di Indonesia. Tapi istilah warisan di dunia OOP berbeda dengan warisan yang diperebutkan orang di sinetron.

Bila kita membicarakan warisan, tentunya ada orang tua yang memberikan warisan, dan ada keturunannya yang menerima warisan. Di Java, orang tua tersebut disebut dengan istilah *superclass*, sedangkan keturunannya disebut dengan istilah *subclass*.

Ada dua jenis inheritance di Java, *type inheritance* dan *implementation inheritance*. Kita akan bahas implementation inheritance yang lebih mudah dipahami.

Implementation Inheritance

Misalnya kita memiliki class yang merepresentasikan orang, kita beri nama class ini `Person`.

```
package com.artivisi.sample;
import java.util.Date;
public class Person {
  private String name;
  private Date birthdate;

  public String getName() {
    return this.name;
  }
  public void setName(String name) {
    this.name = name;
  }
  // getter dan setter birthdate tidak ditampilkan
}
```

Kemudian, kita juga membuat class `Employee` (karyawan) dan `Student` (pelajar). Keduanya jelas-jelas adalah orang, sehingga memiliki variabel nama dan tanggal lahir. Tapi ada beberapa hal yang berbeda antara `Person`, `Employee`, dan `Student`. `Employee` memiliki variabel tambahan, yaitu nomer pegawai. Sedangkan `Student` memiliki variabel nama sekolah tempat dia terdaftar.

Karena ada kesamaan ini, kita dapat membuat class `Employee` dan `Student` sebagai turunan dari class `Person`.

```
package com.artivisi.sample;
public class Employee extends Person {
  private String code;
  public String getCode() {
    return code;
  }
  public void setCode(String code) {
    this.code = code;
  }
}
```

Class `Student` tampak seperti ini.

```
package com.artivisi.sample;
public class Student extends Person {
  private String schoolName;
  public String getSchoolName() {
    return schoolName;
  }
  public void setSchoolName(String schoolName) {
    this.schoolName = schoolName;
  }
}
```

Kedua class tidak perlu lagi membuat variabel `name` dan `birthdate`, karena sudah mendapat warisan dari class `Person`.

Jadi, untuk melakukan inheritance, kita menggunakan keyword `extends`.

Pada contoh kode di atas, class Student dan Employee mendapatkan beberapa warisan sebagai berikut:

- Method getter untuk variabel `name` (`getName`)
- Method getter untuk variabel `birthdate` (`getBirthdate`)

- Method setter untuk variabel `name` (`setName`)
- Method setter untuk variabel `birthdate` (`setBirthdate`)

Keempat warisan di atas disebut juga implementation inheritance. Artinya, subclass mendapatkan warisan berupa implementasi method atau variabel dari superclassnya.

Selain keempat warisan tersebut, ada satu lagi warisan dari class Person, yaitu tipe datanya. Class Student dan Employee secara otomatis memiliki tipe data Person. Hal ini dapat dibuktikan dengan contoh kode berikut.

```
Person endy = new Employee();
endy.setName("Endy Muhardin");
```

Kode di atas tidak error, walaupun kita mendeklarasikan variabel endy dengan tipe data `Person`, tapi menggunakan constructor dari class `Employee` untuk membuat objectnya. Itu artinya class `Employee` juga bertipe data `Person`.

Orang-orang sering mengatakan bahwa Java tidak mendukung *multiple inheritance*. Sebenarnya yang lebih tepat adalah, Java tidak mendukung *multiple implementation inheritance*. Di Java, kita cuma bisa `extends` dari satu superclass saja. Berbeda dengan C++ yang memperbolehkan kita melakukan inheritance dari banyak superclass.

Para pembuat Java memutuskan untuk tidak menyertakan multiple extends untuk menghindari kebingungan.

Tapi sebenarnya, Java mendukung *multiple type inheritance*. Ini dimungkinkan dengan menggunakan interface.

Type Inheritance dengan Interface

Misalnya kita memiliki class `ContactForm` yang dapat menyimpan data `Contact`. Class ini menggunakan `ContactDao` untuk

menyimpan data ke database.

```
package com.artivisi.addressbook.ui;
public class ContactForm {
  private ContactDao contactDao;
  public void setContactDao(ContactDao contactDao) {
    this.contactDao = contactDao;
  }
  public void onButtonSaveClicked() {
    Contact con = new Contact();
    con.name = txtName.getText();
    contactDao.save(con);
  }
}
```

`ContactDao` didefinisikan sebagai berikut.

```
package com.artivisi.addressbook.dao;
public interface ContactDao {
  public void save(Contact con);
}
```

Seperti kita lihat, di dalam interface hanya ada deklarasi method, tanpa implementasi. Implementasinya nanti akan dibuatkan oleh class lain yang meng-*implements* interface ini.

Lalu, kita juga memiliki `GroupForm` untuk mengelola grup. Class ini menggunakan `GroupDao` untuk menyimpan data grup ke database.

```
package com.artivisi.addressbook.ui;
public class GroupForm {
  private GroupDao groupDao;
  public void setGroupDao(GroupDao groupDao) {
    this. groupDao  = groupDao;
  }
  public void onButtonSaveClicked() {
    Group grp = new Group();
    grp.name = txtName.getText();
    groupDao.save(grp);
  }
}
```

`GroupDao` didefinisikan sebagai interface sebagai berikut.

```
package com.artivisi.addressbook.dao;
public interface GroupDao {
  public void save(Group grp);
}
```

Interface `GroupDao` memiliki method yang sama dengan `ContactDao`, yaitu `save`. Bedanya terletak pada tipe data yang menjadi argumen method tersebut.

Kalau kita mendefinisikan masing-masing DAO sebagai interface, kita dapat menggabungkan implementasi `GroupDao` dan `ContactDao` dalam satu class, seperti ini.

```
package com.artivisi.addressbook.dao;
public class AddressBookDatabase
                        implements ContactDao, GroupDao {
  private Connection conn;
  public void setConnection(Connection conn) {
    this.conn = conn;
  }
  public void save(Group grp) {
    // kode untuk menyimpan object Group ke database
  }
  public void save(Group grp) {
    // kode untuk menyimpan object Group ke database
  }
}
```

Nantinya, kita bisa menggunakan class ini di main method kita untuk melayani class `ContactForm` dan `GroupForm`.

```
package com.artivisi.addressbook;
public class AddressBookApplication {
  public static void main(String[] args) {
    AddressBookDatabase adb = new AddressBookDatabase();
    ContactForm cForm = new ContactForm();
    cForm.setContactDao(adb);

    GroupForm gForm = new GroupForm();
    gForm.setGroupDao(adb);
  }
}
```

Polymorphism

Polymorphism adalah kelebihan utama bahasa yang berorientasi objek. Dengan polymorphism, kode program kita bisa lebih fleksibel.

Apa Itu Polymorphism

Polymorphism sering disebut juga *late binding*. Artinya Java menghubungkan (bind) antara pemanggilan method dan pemilihan implementasi yang sesuai pada saat kode program dijalankan (runtime). Ini berbeda dengan bahasa C yang melakukan binding pada saat kompilasi (compile time). Tahap kompilasi tentu lebih awal (early) dibandingkan fase runtime. Sehingga binding pada saat kompilasi disebut *early binding*, dan binding pada saat runtime disebut *late binding*.

Overriding

Dengan adanya polymorphism, kita bisa melakukan *overriding* dalam Java. Override artinya mengganti method yang ada di superclass pada subclassnya. Mari kita lihat contoh berikut. Kita punya satu interface `ContactDao` yang memiliki dua implementasi, yang satu `ContactDaoDatabase` menyimpan data ke database, sedangkan satu lagi `ContactDaoTextFile` menyimpan data ke file teks.

Berikut adalah interface `ContactDao`.

```
public interface ContactDao{
  public void save(Contact con);
}
```

Ini adalah implementasi yang menyimpan data ke database.

```
public class ContactDaoDatabase implements ContactDao {
  public void save(Contact con){
    String sql = "INSERT INTO tbl_contact VALUES (?,?)";
    // eksekusi SQL di atas
  }
}
```

Sedangkan ini adalah implementasi yang menyimpan ke file.

```
public class ContactDaoTextFile implements ContactDao {
  public void save(Contact con){
    File dbFile = new File("contacts.txt");
    // simpan data contact di file
  }
}
```

Di dalam `ContactForm`, kita menggunakan tipe data `ContactDao`.

```
public class ContactForm {
  private ContactDao contactDao;
  public void onButtonSaveClicked() {
    Contact c = new Contact();
    c.setName(txtName.getText());
    contactDao.save(c);
  }
}
```

Apa yang terjadi bila tombol `Save` diklik sangat tergantung pada object yang diberikan pada class `ContactForm`. Bila kita berikan object `ContactDaoDatabase`, maka dia akan menyimpan ke database, sedangkan bila kita berikan `ContactDaoTextFile`, maka data `Contact` akan ditulis ke file teks.

Dengan adanya polymorphism, kita bisa lihat melalui contoh kode di atas bahwa aplikasi yang kita buat menjadi lebih fleksibel. Kita

dapat mengganti implementasi melalui file konfigurasi. Tanpa kompilasi ulang, kode program akan langsung menyesuaikan perilakunya.