PROGRAM STUDI INFORMATIKA
FAKULTAS KOMUNIKASI
DAN INFORMATIKA

PEMROGRAMAN BERORIENTASI OBJEK

MODUL PRAKTIKUM

Dimas Aryo Anggoro, S.Kom.,M.Sc.
Dyah Priyawati, S.T.,M.Eng.
Dedy Gunawan, S.T.,M.Sc.
Gunawan Ariyanto, Ph.D.
Widiyarti Endang Saputri

PEMROGRAMAN BERORIENTASI OBJEK

MODUL PRAKTIKUM

Dimas Aryo Anggoro, S.Kom.,M.Sc
Dyah Priyawati, S.T.,M.Eng
Dedy Gunawan, S.T.,M.Sc.
Gunawan Ariyanto, Ph.D.
Widiyarti Endang Saputri

2019

PEMROGRAMAN BERORIENTASI OBJEK
MODUL PRAKTIKUM

Penulis

Dimas Aryo Anggoro, S.Kom.,M.Sc
Dyah Priyawati, S.T.,M.Eng
Dedy Gunawan, S.T.,M.Sc.
Gunawan Ariyanto, Ph.D.
Widiyarti Endang Saputri

Layout & Cover

Ali Himawan

ISBN: 978-602-361-238-3

Cetakan 1, September 2019

Penerbit **Muhammadiyah University Press**

Universitas Muhammadiyah Surakarta
Gedung i Lantai 1
Jl. A Yani Pabelan Tromol Pos 1 Kartasura Surakarta 57102
Jawa Tengah - Indonesia
Telp: (0271) 717417 Eks. 2172
Email: muppress@ums.ac.id

KATA PENGANTAR

Puji syukur kehadirat Allah SWT atas rahmat dan karunia-Nya sehingga modul praktikum Pemrograman Berorientasi Obyek (PBO) ini dapat terseleasikan dengan baik. Secara umum modul ini berisikan tentang konsep PBO / *Object Oriented Programming* (OOP) beserta implementasinya menggunakan bahasa pemrograman Java. Pada modul ini terdapat materi berupa konsep dasar, latihan soal sebagai pendukung materi, dan pekerjaan rumah – yang diharapkan dapat mengasah kemampuan pemrograman mahasiswa dengan mengimplementasikan konsep PBO.

Secara umum, modul ini membahas beberapa konsep dasar PBO seperti class, object, variable, methods, modifier, constructor, inheritance, encapsulation, polymorphism, abstraction, dan nested class. Konsep PBO di modul ini dikemas dengan menggunakan Java, maka sebelum mempelajari tentang konsep PBO, mahasiswa mempelajari tentang Java yang akan dijelaskan pada bab pertama. Selain itu, salah satu output dari mata kuliah ini adalah mahasiswa diharapkan dapat membuat suatu aplikasi. Oleh karena itu, materi tentang Graphical User Interface (GUI) juga akan dipelajari pada bab terakhir modul ini.

Akhirnya, ucapan terima kasih penulis haturkan pada Program Studi Informatika UMS yang telah memberikan kesempatan pada penulis dalam menyusun modul praktikum ini. Selain itu, penulis juga mengucapkan terima kasih kepada pihak-pihak lain terkait yang telah membantu penulis dalam penyusunan modul ini. Penulis menyadari bahwa dalam penyusunan modul ini terdapat banyak kekeliruan, sehingga saran dan kritik yang membangun sangat penulis harapkan untuk perbaikan modul ini.

Surakarta, Agustus 2019

Penulis

DAFTAR ISI

BAB 1
PENGENALAN JAVA

1.1. PEMROGRAMAN BERORIENTASI OBJEK DALAM JAVA

Pemrograman berorientasi object (PBO) –disebut juga dengan Object Oriented Programming (OOP)– merupakan konsep pemrograman yang menggambarkan suatu proses yang terlibat di dalam program dianalogikan sebagai sebuah object yang saling berinteraksi satu sama lain.

Object adalah kata kunci untuk memahami konsep PBO. Object di dalam PBO sendiri diciptakan berdasarkan deskripsi dan sifat-sifat dari Object nyata yang ada di kehidupan nyata kita. Seperti kita ketahui bersama, di dalam lingkungan sekitar, kita dapat dengan mudah menemukan suatu object semisal: Mobil, Sepeda, Kucing, atau Pohon.

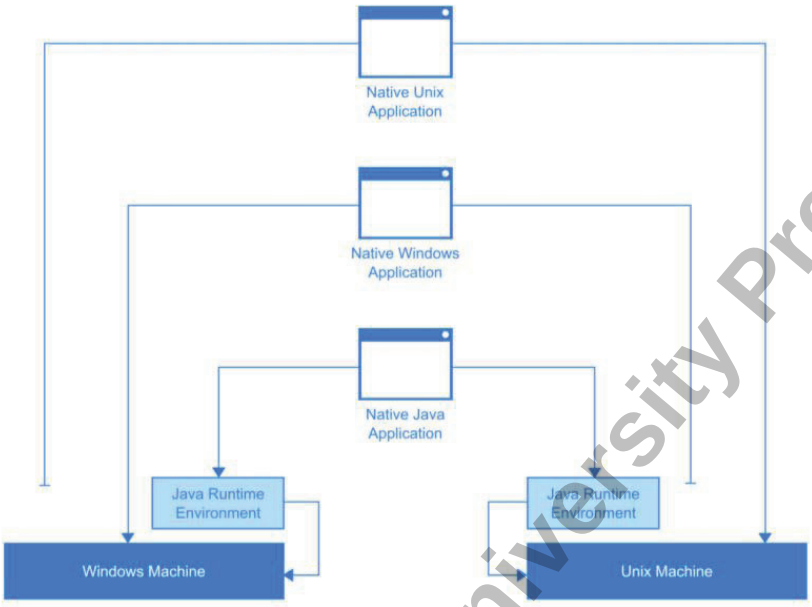
Semua Object riil di atas sama-sama meiliki memiliki dua hal yang melekat padanya, yaitu berupa: kondisi (state) dan perilaku (behavior). Perilaku identik dengan kata kerja dan kondisi identik dengan kata sifat, atau kata benda dari object yang bersangkutan. Mobil memiliki kondisi berupa warna, umur, posisi persneling, kecepatan, dll, dan perilaku berupa injak gas, injak rem, ganti persneling, ganti warna cat, dll. Serupa dengan mobil, Object sepeda memiliki kondisi berupa posisi gigi, kecepatan, dll dan perilaku berupa ganti gigi, naikan kecepatan, pengereman, dll. Sedangkan Object kucing memiliki kondisi berupa warna bulu, jenis, nama, dll dan perilaku berupa meong, mencakar, dll.

Object yang digunakan di dalam pemrograman secara konsep memiliki kesamaan dengan object riil. Object di dalam OOP memiliki variabel dan method. Variabel sebagai analogi dari data atau kondisi sedangkan method sebagai analogi dari perilaku. Salah satu bahasa pemrograman yang mendukung konsep pemrograman berorientasi object dan yang paling banyak digunakan adalah Java. Java sendiri sebenarnya merupakan sebuah platform dimana program-program lain yang akan dijalankan harus menggunakan fitur dari java itu sendiri. Java memiliki arsitektur yang tidak memiliki ketergantungan (dependencies) terhadap sistem operasi, sehingga apapun jenis sistem operasi yang di-gunakan kita masih bisa menjalankan program java tanpa harus mengganti kode program java-nya. Arsitektur dari Java Environment bisa dilihat pada Gambar 1.1.

1.2. JAVA PLATFORM

Java platform merupakan nama platform komputer yang diberikan oleh Oracle yang bisa membantu para pengguna dalam mengembangkan dan menjalankan aplikasi yang berbasis Java. Selain itu, Java platform juga menyediakan berbagai perlengkapan yang memberikan kemudahan bagi para programmer dalam membuat program sehingga pekerjaan merekan menjadi lebih efisien. Komponen Java platform diantaranya:

- Java Runtime Environmental (JRE) merupakan lingkungan komputasi yang diperlukan untuk menjalankan aplikasi Java dan Applets.
- Java Development Kit (JDK) merupakan perlengkapan yang digunakan untuk membuat aplikasi berbasis Java dan Applets. Ketika kita melakukan instalasi JDK ke dalam sistem komputer maka secara otomatis JRE juga akan terinstal.



Gambar 1.1: Arsitektur JAVA

Untuk melakukan pengecekan apakah komputer yang kita gunakan sudah memiliki java platform atau belum, kita bisa mengetikkan kata java pada command prompt dan tekan enter. Jika JDK sudah terinstal maka tampilan command prompt akan seperti berikut :

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All
rights reserved.
C:\Users\Dell>java
Usage: java [-options] class [args...]
(to execute a class)
or java [-options] -jar jarfile [args...]
(to execute a jar file)
where options include:
-d32 use a 32-bit data model if available
-d64 use a 64-bit data model if available
-server to select the "server" VM
The default VM is server.
```

Salah satu komponen dari JRE yang paling penting adalah Java Virtual Machine (JVM). JVM berfungsi sebagai interpreter yang mentransformasikan byte-code menjadi seperti native kode bagi sistem operasi. Byte-code merupakan file hasil kompilasi dari file .java yang berekstensi .class, dimana file .class ini akan bisa terkompilasi disemua sistem operasi sehingga programmer tidak perlu menulis ulang kode program pada sistem operasi yang berbeda. untuk melakukan kompilasi kode program java, kita perlu membuka command prompt dan ketikkan perintah:

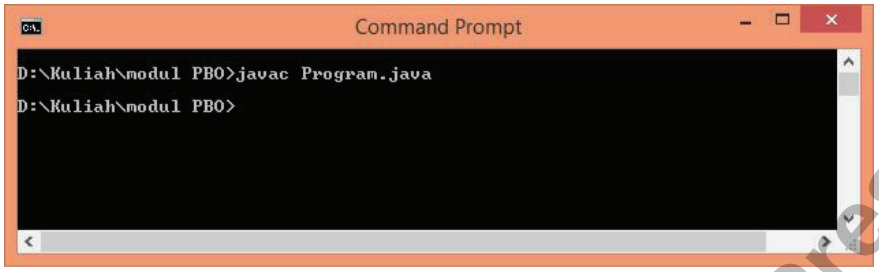
```
javac namaprogram.java
```

Setelah kode java terkompilasi maka akan muncul satu file byte-code yang nama filenya sama dengan nama file kode programmnya yaitu Program.class yang letaknya berada pada folder yang sama dengan program java-nya. Contoh sederhana kode java diperlihatkan pada Program 1 di bawah ini. Tulis kembali dan simpan dengan nama Program.java. Catatan: Nama file java harus sama dengan nama kelas yang ada di kode program.

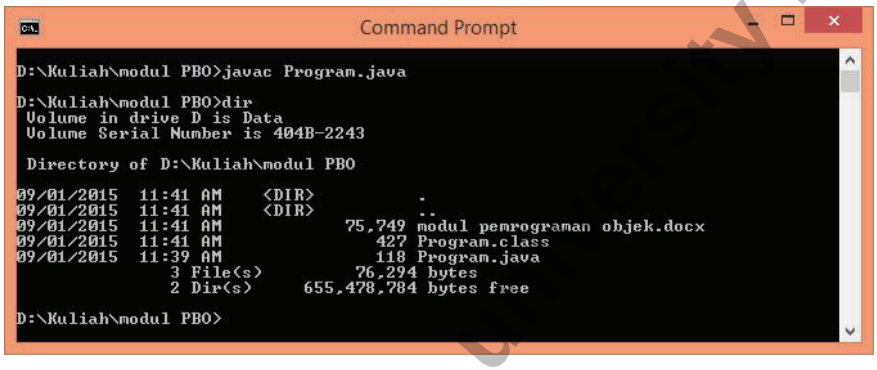
```
1  public class Program{
2    public static void main(String[] args) {
3          System.out.println("Hello World");
4    }
5  }
```

Program 1. Contoh kode Java `Program.Java`

Kompile kode java yang sudah anda tulis dengan cara seperti gambar 1.2. Jika kompilasi berhasil maka tampilan command prompt akan seperti gambar di atas dimana command prompt tidak menampilkan adanya error. Untuk memastikan apakah file byte-code sudah tercipta atau belum kita bisa melakukan pengencekan pada folder dimana file java disimpan seperti ditunjukkan pada Gambar 1.3. Buka command prompt dan ketikkan perintah dir atau buka melalui windows explorer untuk melihat `file.class`. Setelah itu, untuk menjalankan program, pada command prompt tuliskan perintah `java namaFile.`

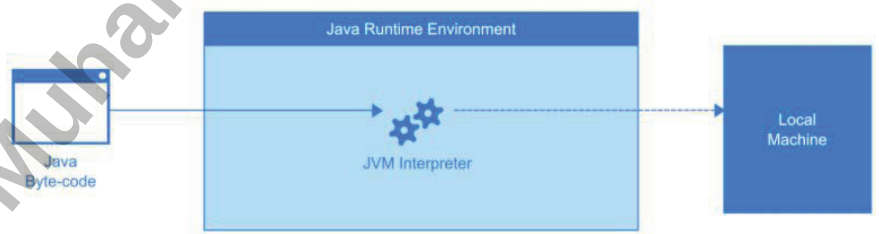


Gambar 1.2: Kompilasi Kode JAVA



Gambar 1.3: File class berhasil dibuat

Pada Gambar 1.3. terlihat bahwa file yang bernama Program.class berada pada lokasi folder yang sama, yang berarti proses kompilasi program berhasil. File Program.class inilah yang nantinya bisa dijalankan diberbagai jenis sistem operasi tanpa perlu mengubah atau menulis ulang kode Program.java. Proses kompilasi dari byte code menjadi program yang bisa dilihat hasilnya melibatkan JVM interpreter, seperti ditunjukkan Gambar 1.4.



Gambar 1.4: Proses pembacaan byte code

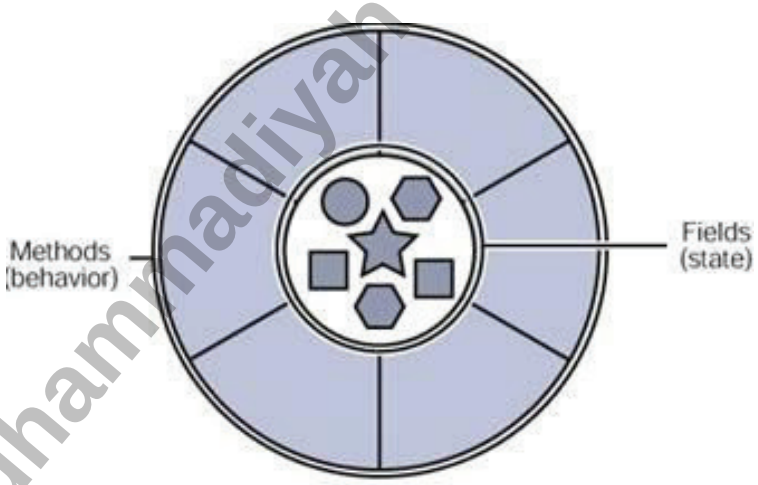
Perangkat lunak pendukung praktikum diantaranya adalah :

- Java SDK 8 (https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)
- Netbeans IDE (https://netbeans.org/downloads/)

BAB 2
CLASS DAN OBJECT

2.1. CLASS

Class adalah cetak biru / *blueprint* dari object. Dengan kata lain, suatu object dapat dibuat/diturunkan dari suatu model rancangan yang disebut dengan class. Berdasarkan definisi ini, maka suatu class cukup dibuat hanya satu buah saja, dari class tersebut dapat dibuat/diciptakan beberapa object (puluhan atau bahkan ratusan) yang memiliki sifat yang sama.



Gambar 2.1. Model Object

Pada Gambar 2.1. terlihat bahwa suatu class yang dianalogikan seperti cetakan roti. Dari cetakan tersebut terdapat bagian yang berupa *field* dan *method*. Field dapat juga disebut dengan variabel

yang merepresentasikan data yang tersimpan, contoh: rasa dan warna. Selain itu, method merepresentasikan kata kerja yang bisa diterapkan pada ataupun dilakukan oleh object yang akan dihasilkan dari suatu class.

2.2. OBJECT

Object merupakan representasi nyata dari sebuah class, yang bisa kita artikan sebagai contoh nyata dari class. Kita tidak bisa membuat object kalau tidak ada class yang bisa dijadikan sebagai panduan, sehingga class harus diciptakan terlebih dahulu sebelum suatu object dibuat.

Untuk membuat "*object*" roti dari class roti di atas, kita harus membuat cetakan roti terlebih dahulu dan kita hanya memerlukan satu cetakan roti saja. Bentuk umum dari sebuah roti biasanya memiliki warna, rasa, harga dan berat, sehingga semua roti yang kita buat pasti akan memiliki karakter yang sama. Di dalam pemrograman berorientasi object, bentuk, ukuran dan berat disebut sebagai properti atau field. Setiap field memiliki tipe yang berbeda satu sama lain, misalnya berat yang nantinya berisi angka, warna yang berisi nama-nama warna dan lain sebagainya.

Untuk lebih jelasnya silahkan tulis dan jalankan contoh kode Program 1. dibawah ini.

```
1  public class Roti {
2  String warna;
3  String rasa;
4  int berat;
5  double harga;
6
7  void beriWarna(String warnaRoti) {
8    warna = warnaRoti;
9  }
10
11 void beriRasa(String rasaRoti) {
12   rasa = rasaRoti;
13 }
```

```
14
15  void timbangBerat(int beratRoti) {
16    berat = beratRoti;
17  }
18
19  void hargaJual(double hargaRoti) {
20    harga = hargaRoti;
21  }
22
23  void infoRoti() {
24     System.out.println(
25          "Warna Roti : " + warna + "\n" +
26          "Rasa Roti : " + rasa + "\n" +
27          "Berat Roti : " + berat + "\n" + "gr"
    +
28          "Harga Roti : Rp. " + harga);
29  }
30  }
```

Program 1. Implementasi class dan objek `Roti.Java`

Kode Program 1 di atas merupakan template atau bentuk abstrak dari roti akan tetapi kode diatas tidak bisa dijalankan untuk menampilkan informasi dari object roti, sehingga ketika kita ingin menampilkan informasi dari object roti menggunakan class di atas kita harus membuat sebuah fungsi `main()` atau fungsi utama. Fungsi `main()` bisa dibuat di dalam class tersebut ataupun dibuat di class lain dengan terlebih dahulu kita membuat object baru. Pembuatan object baru dari suatu class memiliki sintaks sebagai berikut :

```
NamaClass namaObject = new NamaConstructor ();
```

Kata kunci `new` mengindikasikan bahwa object akan diciptakan dari class java dengan nama object yang sudah dituliskan setelah nama class java. Untuk membuat object roti dari class Roti, kita bisa membuat class baru atau bisa juga membuat object roti di dalam class yang sama. Contoh kali ini, kita akan membuat kode program untuk membuat object `roti` seperti terlihat pada Program 2.

```
1   public class RotiDemo{
2     public static void main(String[] args) {
3           Roti roti = new Roti();
4           roti.beriWarna("Hijau");
5           roti.beriRasa("Pandan");
6           roti.timbangBerat(30);
7           roti.hargaJual(6000);
8           roti.infoRoti();
9     }
10  }
```

Program 2. Contoh pembuatan fungsi `main()` dan objek baru

Class Roti tidak memiliki fungsi `main()` dikarenakan class tersebut bukan merupakan program aplikasi melainkan hanyalah sebuah cetak biru untuk menciptakan object roti. Untuk menjalankan suatu program aplikasi, maka harus membuat suatu class yang memiliki sebuah fungsi utama main method. Seperti yang kita lihat bahwa kode diatas merupakan kode program aplikasi untuk membuat satu buah object roti dengan nama `RotiDemo` yang mana kode diatas bisa kita jalankan dan bisa menampilkan informasi tentang roti yang sudah kita buat.

2.3. UML CLASS DIAGRAM

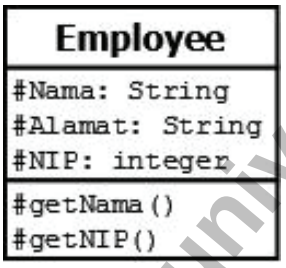
Unified Modeling Language (UML) merupakan diagram yang biasa digunakan untuk mengilustrasikan sebuah class dan sekaligus hubungan antar class. Gambar 2.2. menunjukkan di dalam satu class diagram terdapat tiga row atau baris dimana masing-masing row memiliki kegunaan yang berbeda-beda, diantaranya :

1. Baris Pertama digunakan untuk menunjukkan nama class
2. Baris kedua digunakan untuk menyatakan variabel yang terdapat pada class
3. Baris ketiga digunakan untuk menyatakan method-method yang ada pada suatu class

UML juga bisa digunakan untuk memberikan indikasi tentang *access modifier* dari sebuah variabel atau method. Akses modifier

merupakan ketentuan diperbolehkan atau tidak dalam mengakses variable atau method tertentu yang akan dipelajari pada bab 4 nanti. UML untuk access modifier diantaranya adalah sebagai berikut :

+ : Tanda ini menunjukkan bahwa suatu variabel atau method memiliki akses modifier **public**
- : Tanda ini menunjukkan bahwa suatu variabel atau method memiliki akses modifier **private**
: Tanda ini menunjukkan bahwa suatu variabel atau method memiliki akses modifier **protected**



Gambar 2.2: Class Diagram

2.3.1. Tujuan Penggunaan Class Diagram

Tujuan dari penggunaan UML class diagram sebenarnya untuk menggambarkan model dari aplikasi yang akan kita buat. Class diagram merupakan satu-satunya diagram yang dapat menggambarkan dengan detail bentuk class yang akan dibuat. Berikut adalah tujuan utama penggunaan class diagram:

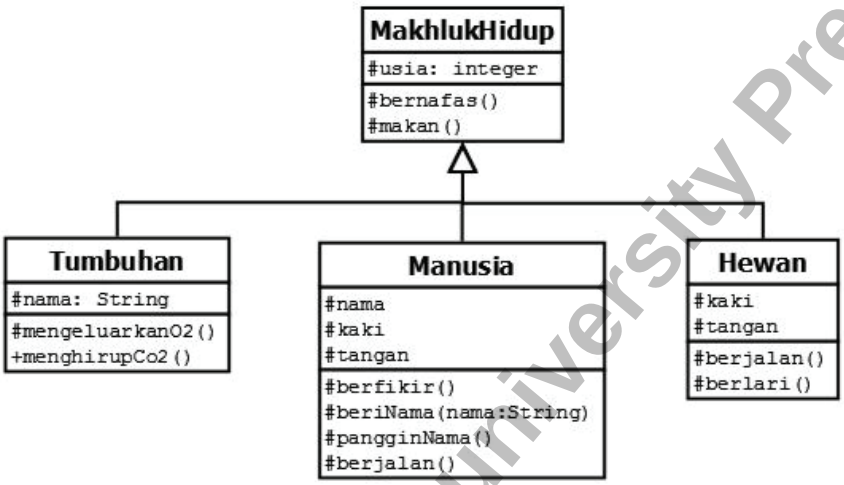
- Untuk membuat analisis dan desain statis dari aplikasi
- Menggambarkan kemampuan program yang akan kita buat
- Sebagai dasar komponen pengembangan system

2.3.2. Aturan Pembuatan Class Diagram

- Nama dari class harus memiliki arti yang bisa mewakili fungsi sesungguhnya dari aplikasi
- Atribut (Variabel dan Method) harus disebutkan dengan jelas

- Gunakan catatan tambahan untuk mendeskripsikan beberapa aspek pada diagram

Gambar 2.3. berikut ini adalah contoh penggunaan class diagram dalam pemrograman berorientasi object.



Gambar 2.3. Penggunaan Class Diagram dalam PBO

2.4. LATIHAN

1. Silakan modifikasi class `RotiDemo` dan buatlah 3 object baru di dalamnya
2. Gambarkan class diagram dari class `RotiDemo`
3. Buatlah satu class baru yang bisa digunakan sebagai template/ *blueprint* dari class `CarDemo` seperti terlihat pada Program 3. Class baru tersebut tidak memiliki fungsi `main()`.

```
1  public class CarDemo{
2  public static void main(String[] args) {
3      Car car1 = new Car();
4  Car car2 = new Car();
5
6  car1.changeCadence(50);
7  car1.speedUp(20);
```

```
8   car1.changeGear(2);
9   car1.printInfo();
10
11  car2.changeCadence(30);
12  car2.speedUp(10);
13  car2.changeGear(1);
14  car2.printInfo();
15  }
16  }
```

Program 3. `CarDemo.java`

4. Buatlah suatu class yang dapat merepresentasikan sifat-sifat dari object `Kucing`. Object ini memiliki field/variable/properties berupa `umur`, `warna bulu` dan method berupa `meong()` dan `umur()`.
5. Salah satu aplikasi PBO yang sangat umum adalah berupa aplikasi keuangan. *Bank Account* (Rekening Bank) adalah salah satu hal yang dapat dijadikan sebagai suatu object di dalam PBO.
   a. Buatlah suatu class yang dapat merepresentasikan Object Rekening tersebut. Variabel dari object ini adalah `saldo`, `no_rekening`, nama dan method berupa `cek_saldo()`, `menabung()`, `menarik()`, dan `transfer()`.
   b. Buatlah suatu class yang memiliki sebuah fungsi `main()` yang digunakan untuk mendemokan pembuatan object tersebut.
6. Perhatikan Class String yang ada di dalam dokumentasi Java. Sebutkan daftar variable dan fungsi/method yang dimiliki oleh Class String tersebut.

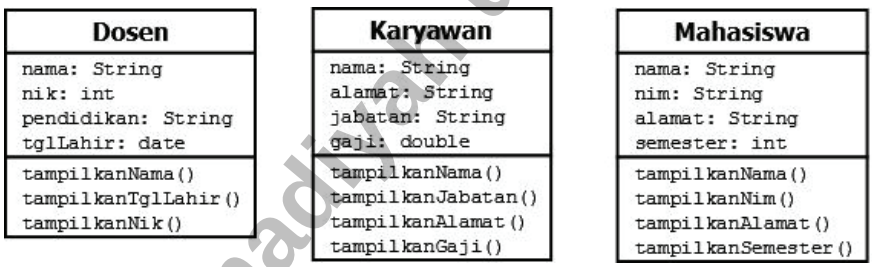
2.5. PEKERJAAN RUMAH

1. Buatlah class `Hewan` dan juga ciptakan object dari class tersebut sehingga kita bisa membuat berbagai macam object hewan dengan karakternya masing-masing! Kode program akan menghasilkan output seperti berikut.

```
Nama Hewan : Harimau
Jumlah Kaki : 4
Makanan : Daging
Type Hewan : Karnivora

Nama Hewan : Kerbau
Jumlah Kaki : 4
Makanan : Rumput
Type Hewan : Karnivora
```

2. Buatlah class berdasarkan class diagram pada Gambar 2.4. berikut ini!



Gambar 2.4. Tugas berdasarkan UML Class Diagram

BAB 3
CLASS MEMBER: VARIABEL DAN METHOD

Dalam pemrograman berorientasi object, setiap object memiliki propertis dan *behaviour* atau perilaku yang menunjukkan karakter dari object tersebut. Selain itu, setiap object juga mampu berkomunikasi satu dengan yang lainnya melalui sebuah method dengan cara mengirimkan pesan tertentu. setiap pesan yang dikirim oleh suatu object akan diterima dan dikenali oleh object lain jika pesan tersebut memiliki tipe data yang dikenali oleh object lain.

3.1. VARIABEL

Variabel atau disebut juga field merupakan member dari class dimana setiap variabel harus memiliki tipe data tertentu. Ketika kita membuat suatu variabel maka akan ada memori komputer yang disediakan untuk menyimpan variabel tersebut. Tipe data di dalam pemrograman java bisa berupa tipe data primitif dan tipe data reference atau tipe data object. Tipe data primitif artinya tipe data ini sudah didefinisikan oleh bahasa pemrograman dengan kata kunci (*keyword*) khusus. Sedangkan tipe data reference merupakan tipe data yang merujuk pada suatu class object. Tipe data primitif diperlihatkan pada Tabel 3.1 Penulisan tipe data pada variabel harus sesuai dengan syntax pemrograman java, seperti contoh berikut ini.

```
tipe data namaVariabel = value;
tipe data namaVariabel;
namaVariabel = value;
```

Tabel 3.1. Tipe data primitive

Data Type	Default Value	Default Size
int	0	4 byte
float	0.0f	5 byte
long	0L	8 byte
double	0.0d	8 byte
boolean	false	1 bit
char	'u0000'	2 byte
byte	0	1 byte
short	0	2 byte

Contoh :

```
int a, b, c, d = 10;
byte e = 22;
double pi = 3.14159;
char f = 'f';
```

Ada beberapa jenis variabel yang biasa digunakan dalam pemrograman berorintasi object yaitu *local variabel*, *instance variabel* dan *class/static variabel*. Ketiga jenis variabel tersebut memiliki karakteristik yang berbeda-beda.

3.1.1. LOCAL VARIABLE

Local variable / Variabel lokal merupakan variabel yang dideklarasikan di dalam sebuah method, konstruktor ataupun dalam suatu blok program. Variabel jenis ini akan diciptakan ketika method dijalankan dan akan dihapus dari memori komputer ketika method tersebut dihentikan. Sehingga lokal variabel hanya bisa digunakan oleh method yang mendeklarasikannya dan blok atau fungsi lain tidak bisa menggunakan variabel lokal dari method lain. Hal yang perlu diperhatikan adalah lokal variabel tidak memiliki nilai default sehingga setiap variabel harus di deklarasikan nilai awalnya.

Program 1. adalah contoh penerapan variabel lokal terdapat pada sebuah program. Amati hasilnya.

```
1   public class LocalVariable{
2     public void hitungUsia() {
3           int usia = 0;     // local variable
4           int tahunSekarang = 2019;
5           int tahunLahir = 1993;
6
7           usia = tahunSekarang - tahunLahir;
8
9           System.out.println("Usia saya : " +
    usia);
10    }
11  }
```

Program 1. Penerapan variabel lokal

LATIHAN 1

1. Buatlah method baru untuk menghitung berat badan dengan nama `void beratBadan()`, di dalam method tersebut buatlah variabel lokal `beratLahir`. Isikan nilai awal pada berat lahir kemudian hitunglah berat badan dengan rumus **beratBadan = beratLahir + (umur//2)**
2. Bisakah nilai dari variabel umur dipanggil dari method `void beratBadan()`? berilah alasannya!

3.1.2. INSTANCE VARIABLE

Instance variabel atau bisa kita sebut dengan variabel global merupakan variabel milik dari suatu class dan berada diluar blok atau method tertentu. Variabel jenis ini bisa diakses dari semua method yang menjadi member dari class, sehingga semua method bisa memanipulasi nilai dari variabel global tersebut. Berbeda dengan variabel lokal, variabel global bisa memiliki akses modifier. Program 2. adalah contoh penerapan variabel global.

```
1   public class InstanceVariable{
2     int nilai; // instance variable
3
4     void firstMethod() {
5           // detail
```

```
6       }
7
8       void secondMethod() {
9             // detail
10      }
11  }
```

Program 2. Penerapan variabel global

LATIHAN 2

1. Modifikasi class `LocalVariable` pada Program 1., dengan menambahkan satu variabel global untuk menampung nilai dari umur.
2. Gunakan nilai umur untuk menyelesaikan permasalahan pada method menghitung berat badan yang ada di Latihan 1.
3. Analisa hasilnya dan bandingkan dengan hasil percobaan sebelumnya!

3.1.3. STATIC VARIABLE

Variabel statis merupakan variabel yang disimpan di dalam memori statis dan hanya memiliki satu nilai di mana nilai tersebut akan dipakai oleh semua object yang dibuat tanpa ada batasan jumlah object. Variabel statis biasanya digunakan untuk membuat nilai konstanta. Variabel statis juga bisa diakses oleh semua method yang ada pada suatu class dengan cara memanggil nama dari class dan variabelnya seperti `ClassName.VariableName`. Tulis dan jalankan kode Program 3 di bawah ini:

```
1   public class StaticVariable{
2   public static char akreditasi;
3   public static final String jurusan =
    "Informatika";
4
5   void firstMethod() {
6     System.out.println(jurusan);
7   }
8
```

```
9   void secondMathod() {
10    System.out.println("Akreditasi : " +
    akreditasi );
11  }
12  }
```

Program 3. Penerapan variabel statis

Variabel statis tidak bisa dijadikan sebagai variabel lokal. Sehingga untuk membuatnya, static modifier harus dituliskan pada variable global. Selain itu, untuk mengakses variabel statis kita tidak perlu membuat object dari class yang memiliki variabel statis. Perhatikan Program 4. di bawah ini.

```
1   public class StaticVariableAccess{
2     public static void main(String[] args) {
3           StaticVariable.akreditasi = 'B';
4           StaticVariable sv = new
    StaticVariable();
5           sv.firstMethod();
6           sv.secondMathod();
7     }
8   }
```

Program 4. Contoh pengaksesan variabel statis

Program 4. pada baris ke 3 adalah cara mengakses variabel akreditasi dari class `StaticVariable` tanpa perlu membuat object dari class tersebut.

3.2. METHOD

Method merupakan blok kode program yang menggambarkan perilaku atau perintah suatu program. Method memiliki tiga kategori, yang pertama adalah method dengan tipe data tertentu yang memiliki nilai `return` dan yang kedua adalah method dengan kata kunci `void`. Ketiga, method juga bisa memiliki *parameter*. Parameter di dalam method sifatnya tidak wajib dan itu sangat tergantung pada kegunaan dari method tersebut. Akan tetapi setiap

parameter di dalam method harus memiliki tipe data tertentu Secara umum bentuk sebuah method adalah seperti blok berikut.

```
TipeMethod NamaMathod (parameter) {
  /* code */
}
```

3.2.1. METHOD NON-VOID

Method jenis ini adalah method yang harus memiliki type data pada body method. Dengan demikian maka Method harus memiliki nilai `return` dimana nilai `return` akan digunakan oleh method lain. Nilai `return` yang dihasilkan oleh method ini memiliki tipe data yang sama dengan tipe data yang dituliskan sebelum nama method. Contoh :

```
public String nama() {
  return nama;
}

public double nilaiMax() {
  return 100;
}
```

Dari contoh di atas terlihat bahwa nilai `return` yang dihasilkan oleh method nama akan bertipe data `String` dan method nilai akan menghasilkan nilai `return` dengan tipe data `double`. Tulis ulang kode Program 5. di bawah ini!

```
1  public class NonVoidMethod{
2    public String getNama() {
3          return nama;
4    }
5
6    public String getNIM() {
7          return nim;
8    }
9  }
```

Program 5. Metode non-void

Cobalah untuk memodifikasi kode Program 5, dengan menambahkan dua variabel `nama` dan `nim`, kemudian buatlah fungsi `main()` yang bisa menampilkan nama dan nim Saudara. Tunjukkan hasilnya kepada Asisten Praktikum / Dosen Pengampu.

3.2.2. PARAMETER METHOD

Parameter pada method merupakan sebuah variabel lokal yang hanya bisa diakses oleh method tersebut. Sebuah method boleh memiliki parameter ataupun tidak bahkan bisa memiliki lebih dari satu parameter, tergantung dari kegunaan method yang kita buat. Karena parameter merupakan variabel lokal maka setiap parameter yang dituliskan dalam method harus memiliki tipe data tertentu. Program 6 merupakan contoh parameter pada method.

```
1  public class MethodParameter {
2  String nama;
3  public String setNama (String nama) {
4    return this.nama = nama;
5  }
6  }
```

Program 6. Parameter pada method

Berdasarkan Program 6 di atas, terlihat bahwa baris ke-2 merupakan sebuah variabel global yang bertipe data `String`, kemudian pada baris ke-3 sampai 4 merupakan sebuah blok method. Di dalam method tersebut terdapat parameter yang bertipe `String`. Ketika kita akan menggunakan atau mengirimkan nilai dari suatu parameter ke parameter lain, maka parameter tersebut harus memiliki tipe data yang sama. Baris ke-4 pada kode di atas akan mengembalikan nilai dari `beriNama` dan kemudian akan disimpan ke dalam variabel `nama`. Tambahkan kode berikut ini kedalam class `MethodParameter` untuk melihat proses kerja dari method `setNama()`.

```
1  public static void main (String[] args) {
2    MethodParameter mp = new MethodParameter();
```

```
3       mp.setNama("Luffy");
4       System.out.println(mp.nama);
5   }
```

Program 7. Method `main()` pada class `MethodParameter`

LATIHAN 3

Lengkapilah kode Program 8 di bawah ini dengan menambahkan method yang memiliki parameter kemudian tampilkan hasilnya! Buatlah minimal 5 Object pegawai dengan nama, nim dan gaji yang berbeda-beda!

```
1   public class Pegawai{
2       String nama;
3       int nip;
4       double gaji;
5   }
```

Program 8. Latihan membuat parameter pada method

3.2.3. METHOD VOID

Method void merupakan method yang tidak menghasilkan nilai kembalian atau return value. Method dengan tipe void juga bisa memiliki parameter ataupun tidak, tergantung dari tujuan penggunaannya. Tuliskan kode di bawah ini dan amati perbedaannya dengan method non-void.

```
1   public class VoidMethod{
2   int hour, minute, second;
3
4   public void duration(int hour,int minute, int
    second) {
5       this.hour = hour;
6       this.minute = minute;
7       this.second = second;
8   }
```

```
9
10  public void info() {
11    System.out.println("Total Waktu \n" +
12  hour + " jam " + minute +
13  " menit " + second + " detik");
14  }
15
16  public static void main (String[] args) {
17    VoidMethod vm = new VoidMethod();
18    vm.duration(1, 30, 15);
19    vm.info();
20  }
21  }
```

Program 9. Method void

3.3. PEKERJAAN RUMAH

```
1   public class Nilai{
2   int nilaiUTS;
3   int nilaiUAS;
4   int nilaiTugas;
5   }
```

Program 10. Tugas implementasi method

1. Lengkapilah kode pada Program 10 di atas dengan menambahkan method void dan method return, yang mengembalikan nilai dari setiap parameter method void.
2. Ubahlah tipe data dari `int` ke `double` dan tambahkan satu variabel `double nilaiTotal`, kemudian hitung `nilaiTotal` dengan rumus berikut :

nilaiTotal = (nilaiUTS + nilaiUAS + nilaiTugas)/3

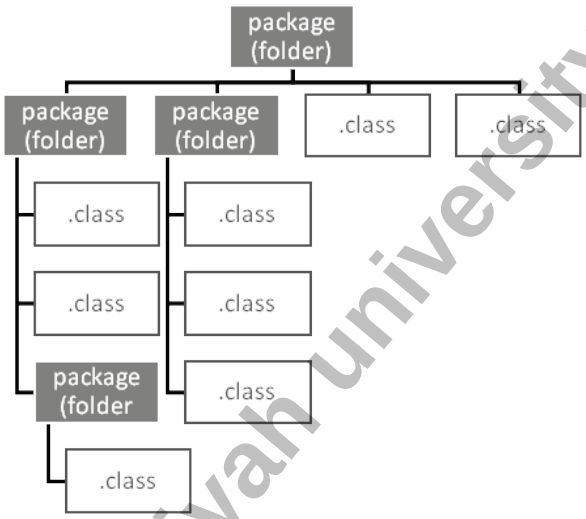
BAB 4
ACCESS MODIFIER

Setiap variabel atau field dalam kode java, sebaiknya memiliki *access modifier* / penentu akses. Maksud dari penentu akses adalah untuk mengijinkan apakah *variable*/field dan method/fungsi boleh diakses oleh class lain atau tidak. Inilah yang disebut dengan *encapsulation* yang akan dijelaskan lebih lengkap di bab 7. Akses terhadap suatu instance variable pada luar class biasanya tidak diperkenankan. Hal ini bermaksud untuk menjaga data dari akses class lain yang bisa mengakibatkan kesalahan. Sebagai penggantinya, disediakan metode yang diperlukan untuk mengakses variabel instan. Berkaitan dengan boleh tidaknya suatu variabel instan diakses dari luar class, Java menyediakan beberapa penentu akses antara lain: `default` (Tanpa penentu akses), `public`, `protected` dan `private`. Untuk bisa memahami perbedaan dari masing-masing tipe akses, kita perlu memahami sedikit tentang *package* atau paket terlebih dahulu.

Package merupakan suatu folder tempat menyimpan class java, sehingga program yang kita buat bisa tersusun dengan rapi. Selain itu package juga berfungsi untuk mengelompokan class java sesuai dengan grupnya masing-masing sehingga bisa meminialisir kesalahan dalam pengaksesan class. Jika tidak mendefinisikan package dalam pembuatan class, maka class tersebut akan dimasukkan ke dalam package default. Package default tidak menyalahi aturan Java, namun untuk pengembangan dan pendistribusian program Java nanti maka package perlu definisikan. Deklarasi package harus dilakukan di dalam class, di baris pertama kode seperti pada Gambar 4.1. Contoh pengaturan class dan package ditunjukkan pada Gambar 4.2.

```
1
2    package praktikum.pbo;
3
4    public class NewClass {
```

Gambar 4.1. Deklarasi package "pbo" di dalam package "praktikum"



Gambar 4.2. Pengaturan class dan package

Apabila menggunakan class yang berada di package yang lain, maka tidak dapat digunakan secara langsung. Berikut ini adalah pilihan yang dapat dilakukan yaitu :

- Menggunakan nama package setiap kali class digunakan. Misalkan, class Mahasiswa berada di dalam package pbo, maka:

```
pbo.Mahasiswa mhs = new pbo.Mahasiswa();
```

- Mengimpor class yang akan digunakan dengan kata kunci import.

```
import pbo.Mahasiswa;
```

- Mengimpor package, yang artnya mengimpor semua class yang masuk dalam package tersebut. Misalkan dalam package

pbo terdapat beberapa class dan akan digunakan semua, maka deklarasinya sebagai berikut :

```
import pbo.*;
```

Selain itu, access modifier dapat diimplementasikan untuk variable atau method. Adapun penjelasan secara umum tentang access modifier `private`,`default` (tanpa penentu akses), `protected` dan `public` adalah sebagai berikut**.**

1. **Private**, variabel atau method hanya dapat diakses dari dalam class itu sendiri.
2. **Default**, atau tanpa penentu akses, yang berarti setiap variabel atau method dapat diakses oleh class yang berada dalam package yang sama.
3. **Protected**, digunakan untuk membuat class tersebut dapat diakses hanya oleh class lain dan turunan dalam satu package.
4. **Public**, variabel atau method dapat diakses pada semua class yang menggunakan class yang berisi variabel tersebut meskipun berada pada package yang berbeda.

4.1. PRIVATE MODIFIER

Jika menerapkan tipe akses `private` pada metode atau atribut class tertentu maka class lain tidak diijinkan untuk mengaksesnya. Meskipun kedua class tersebut berada dalam satu package. Contoh penggunaan tipe akses `private` ditunjukkan pada Program 1.

```
1  public class PrivateModifier {
2      private String nama;
3      private int umur;
4
5      public void printInfo(){
6         System.out.println("private modifier");
7      }
8  }
```

Program 1. Contoh program dengan tipe akses `private`

4.1.1. Latihan

Tuliskan kode di atas dan buatlah class baru untuk mengakses variable dari Program 1. di atas! Apakah variable tersebut dapat diakses dari class lain? Jelaskan!

4.2. DEFAULT MODIFIER

Tipe akses `default` bisa diterapkan baik dalam variabel maupun method. Modifier `default` tidak perlu dituliskan secara eksplisit pada variabel atau method, karena secara otomatis Java akan mengeksekusinya sebagai `default`. Sehingga, yang perlu kita tuliskan adalah tipe data kemudian nama variabel atau nama method saja.

Sifat tipe akses `default`, adalah dapat diakses class lain dalam package yang sama. Perhatikan contoh kode di bawah ini dan kerjakan latihannya!

```
1   class DefaultModifier {
2   int a = 1;
3   int b = 2;
4   int c;
5   void jumlah(){
6   c = a + b;
7   System.out.println(c);
8   }
9   }
```

Program 2. Contoh program dengan tipe akses `default`

4.2.1. Latihan

1. Untuk membuktikan sifat dari tipe akses default, maka buat satu class baru dan coba akses variabel dan method dari Program 2! Amati dan jelaskan hasil yang Saudara dapatkan!
2. Buat package lain dan beri nama yang unik sebagai pembeda! Tambahkan satu class di dalamnya. Kemudian

lakukan pengaksesan variable dan method Program 2. melalui class baru tersebut! Amati dan jelaskan hasil yang Saudara dapatkan!

4.3. PROTECTED MODIFIER

Tipe akses `protected` digunakan untuk membuat class tersebut dapat diakses hanya oleh class lain di dalam suatu package yang sama serta seluruh subclass-nya meskipun berada di package yang berbeda. Perhatikan contoh kode Program 3. di bawah ini.

```
1   public class ProtectedModifier {
2       protected void printInfo(){
3           System.out.println("Protected Modifier");
4       }
5       protected void sendMessage(){
6           System.out.println("this is a message");
7       }
8   }
```

Program 3. Contoh program dengan tipe akses `protected`

4.3.1. Latihan

Buatlah class baru dan object-nya untuk mengakses method dari Program 3., dan bandingkan dengan dua modifier sebelumnya!

4.4. PUBLIC MODIFIER

Tipe akses `public` merupakan penanda bahwa suatu method atau variabel bisa diakses dari berbagai class dan package. `public` biasanya dipakai oleh kode program yang memang mengijinkan semua class bisa mengaksesnya. Perhatikan contoh kode Program 4. berikut ini.

```
1   public class PublicModifier {
2   public int a = 2;
3   public int b = 5;
```

```
4   public int c = 9;
5   public void kali(){
6     int d = a*b*c;
7     System.out.println("Hasil kali = "+d);
8   }
9   }
```

Program 4. Contoh program dengan tipe akses `public`

4.4.1. LATIHAN

1. Lakukan percobaan seperti pada **Latihan 4.2.1.** dan bandingkan hasilnya!
2. Tambahkan method baru diantaranya `tambah()`, `kurang()`, `bagi()`, dan `rata_rata()`!

BAB 5
CONSTRUCTOR

Constructor adalah metode yang dapat digunakan untuk memberikan nilai awal saat object diciptakan. Metode ini akan dipanggil secara otomatis oleh java ketika new dipakai untuk menciptakan object dari suatu kelas.

Constructor memiliki beberapa karakteristik antara lain:

- Nama constructor sama dengan nama kelas
- Tidak memiliki nilai kembali atau return value
- Constructor memiliki beberapa tipe diantaranya :
- Default constructor
- Parameterized constructor

5.1. DEFAULT CONSTRUCTOR

Default constructor merupakan constructor yang secara otomatis dibuatkan oleh java *compiler*, sehingga meskipun kita tidak menuliskan atau membuat constructor secara eksplisit java sudah menyediakannya.

```
1  public class Constructor {
2     public Constructor() {
3           System.out.println("Default
   Constructor");
4     }
5  }
```

Program 1. Default Constructor

Baris kode ke 2-4 pada Program 1 merupakan constructor yang tidak memiliki detail implementasi. Constructor yang seperti ini disebut dengan *default constructor*, kita boleh menuliskannya maupun tidak. Tujuan sebenarnya dari constructor ini adalah untuk memberikan nilai awal pada object seperti 0, `NULL`, dan sebagainya.

Dari contoh Program 1, nilai default dari object yang diciptakan melalui constructor adalah menampilkan output ke layar berisi teks "`Default Constructor`". Program 2 berikut ini dapat digunakan untuk menambah pemahaman tersebut.

```
1  public class ConstructorAccess {
2    public static void main(String[] args) {
3          Constructor cons = new Constructor();
4    }
5  }
```

Program 2. Mengakses default contructor

5.1.1. Latihan 1

Buatlah sebuah class yang bersi variabel `nama`, `nim` dan `alamat` serta sebuah *constructor default* yang dapat menampilkan nama, nim dan alamat Saudara. Selanjutnya buatlah kelas baru dengan fungsi `main()` di dalamnya untuk menampilkan hasilnya.

5.2. PARAMETERIZED CONSTRUCTOR

Parameterized constructor sebenarnya merupakan sebuah method, yang mana method bisa memiliki beberapa parameter atau bahkan tidak memiliki parameter (*default constructor*). Constructor bisa memiliki beberapa parameter sebagai penentu nilai awal dari sebuah object. Seperti layaknya method, parameter di dalam constructor juga harus memiliki tipe data tertentu. Tipe data disini bisa berupa tipe data reference ataupun tipe data primitive. Program 3 berikut ini adalah contoh parameterized constructor.

```
1   public class ParamConstructor{
2   String nama, nim;
3   int semester;
4
5   public ParamConstructor(String nama, int
    semester, String nim) {
6     this.nama = nama;
7     this.semester = semester;
8     this.nim = nim;
9     }
10
11    public void info() {
12    System.out.println("Nama : " + nama + "\nNim :
    "
13  + nim +"\nSemester : " + semester);
14    }
15  }
```

Program 3. Parameterized Constructor

Program 3 terdapat constructor dengan tiga parameter yaitu : `nama`, `semester` dan `nim` dengan tipe data tertentu. Ketika membuat object baru dari kelas tersebut, maka kita harus mengisikan ketiga parameter tersebut pada setiap object yang dibuat. Untuk melihat hasilnya dapat dilihat melalui Program 4.

```
1   public class ConstructorAccess {
2     public static void main(String[] args) {
3     ParamConstructor pc = new
4   ParamConstructor("Luffy", 3, "L2001500xy");
5     pc.info();
6     }
7   }
```

Program 4. Mengakses parameterized constructor

Pada Program 4 baris ke-3, terlihat jelas bahwa ketika kita membuat object baru kita diwajibkan mengisikan nilai dari masing-masing parameter sesuai dengan tipe datanya masing-masing, dimana nilai dari parameter inilah yang kita sebut sebagai *isi dari object*.

5.2.1. Latihan 2

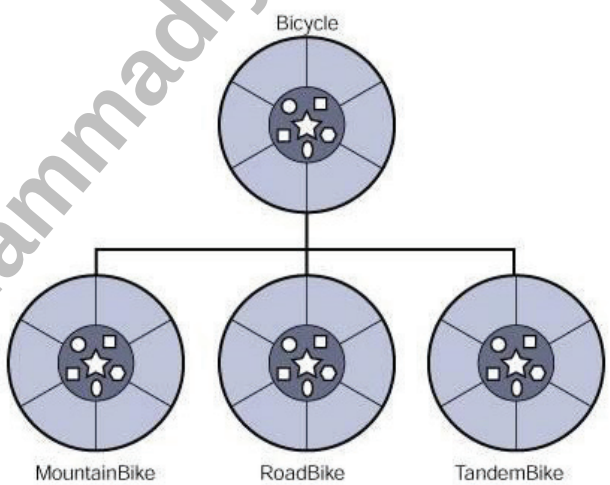
1. Buatlah class `Buku` dengan variabel `String namaPengarang`, `String judulBuku`, `int tahunTerbit`, `int cetakanKe` dan `double hargaJual`. Selanjutnya buatlah minimal tiga constructor dengan parameter yang berbeda, serta sebuah method untuk menampilkan informasi buku tersebut!
2. Buatlah 10 object buku dengan fungsi `main()` yang dapat menampilkan nilai dari semua parameter Buku tersebut.

5.3. TUGAS

1. Buatlah sebuah class yang didalamnya terdapat *default constructor* dan *parameterized constructor*!
2. Tambahkan class baru berisi method `main()` sebagai implementasi dari kelas di atas (No.1)!

BAB 6
INHERITANCE

Inheritance (pewarisan) adalah salah satu ciri utama dari PBO dan merupakan fitur yang sangat memudahkan bagi programmer untuk membuat dan mengatur program. Sepertihalnya dalam kehidupan riil, sifat seoran anak secara umum mirip dengan sifat orang tuanya. Demikian juga pada kasus pemrograman. Suatu class yang diturunkan (*inherited from*) dari induknya (*parent* class) akan mewarisi semua atau sebagian sifat (variabel dan fungsi) dari class induknya. Dengan inheritance kita tidak perlu membuat (mengimplementasikan) ulang suatu class jika kita sudah memiliki class lain yang serupa. Inheritance dapat digambarkan pada contoh Gambar 6.1 di bawah ini.



Gambar 6.1. Inheritance

Gambar 6.1 mengilustrasikan bahwa MountainBike, Roadbike dan TandemBike pas-tilah memiliki banyak kesamaan sifat dengan sepeda pada umumnya, sehingga untuk membuat ketiga jenis sepeda tersebut kita cukup mengambil karakter dari sepeda secara umum semisal semua sepeda pasti memiliki gear, peda, rem, dan lain sebagainya. Oleh karena itu kita hanya perlu menurunkan sifat sepeda (*Bicycle*) kemudian kita terapkan pada MountainBike, RoadBike dan TandemBike.

Untuk menggunakan inheritance kata kunci yang diperlukan adalah extends. Secara umum sintaks dari inheritance adalah seperti berikut :

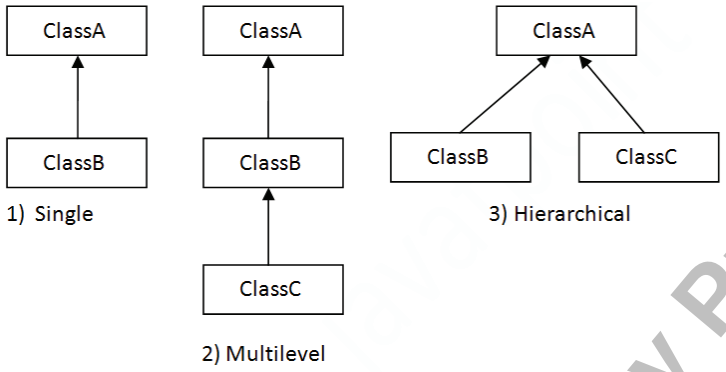
```
class SubClass extends SuperClass{
  /* code */
}
```

Class SubClass merupakan class turunan dari SuperClass. Oleh karena itu properties yang dimiliki oleh SuperClass seperti variabel dan method akan bisa juga dimiliki oleh SubClass.

6.1. TIPE INHERITANCE

Dalam pemrograman berorientasi object inheritance memiliki 3 tipe antara lain

a. **Single inheritance**: Suatu class hanya memiliki satu turunan
b. **Multilevel inheritance**: konsep inheritance yang mengijinkan suatu subclass mempunyai subclass lagi.
c. **Hierarchical inheritance**: Suatu class dapat memiliki lebih dari satu class turunan



Gambar 6.2 Tipe inheritance

6.1.1. Single Inheritance

Single inheritance memperbolehkan subclass hanya mempunyai satu superclass. Program 1 berikut ini adalah contoh single inheritance. Program 1 menunjukkan bahwa class Programmer merupakan class turunan dari class Pegawai, sehingga kita bisa menggunakan variabel dari class Pegawai dan menggunakannya ke dalam class Programmer. Pada baris ke-6 terlihat bahwa class programmer menggunakan variabel gajiPokok dari class pegawai dan menjumlahkannya dengan vari-abel yang ada pada class programmer.

```
1   public class Pegawai {
2     double gajiPokok = 10000000;
3   }
4
```

```
1   public class Programmer extends Pegawai{
2     public static void main(String[] args) {
3     double lemburPerjam = 50000;
4     double gajiPerBulan;
5     Pegawai pegawai = new Pegawai();
6     gajiPerBulan = pegawai.gajiPokok +
    lemburPerjam; System.out.println(gajiPerBulan);
7     }
8   }
```

Program 1. Contoh program single inheritance

6.1.2. Hierarchical Inheritance

Dari Program 1 di atas, kelas `Pegawai` dapat ditambahkan beberapa class turunan yang lain sehingga class `Pegawai` nantinya memiliki beberapa class turunan. Inilah yang disebut dengan *hierarchical inheritance*. Silahkan tulis kode Program 2 berikut ini untuk mempermudah pemahaman.

```
1  public class Teknisi extends Pegawai{
2    public static void main(String[] args) {
3    double lemburPerjam = 40000;
4    double gajiPerBulan;
5    Pegawai pegawai = new Pegawai();
6    gajiPerBulan = pegawai.gajiPokok +
   lemburPerjam; System.out.println(gajiPerBulan);
7    }
8  }
```

Program 2. Contoh program hierarchical inheritance

Program 2 menunjukkan bahwa variabel dari class `Pegawai` bisa dipakai oleh class-class turunannya tanpa perlu kita membuat variabel sendiri di masing-masing class turunannya.

6.1.3. Multilevel Inheritance

Multilevel inheritance merupakan model class turunan dimana suatu sub-class memiliki sub-class lain. Sub-class dari sub-class juga memiliki akses ke class induk yang paling atas. Jika kita ilustrasikan dalam kehidupan nyata seperti seorang cucu yang memiliki orang tua dan kakek, dimana cucu tersebut boleh menggunakan barang milik ayahnya sekaligus barang milik kakeknya. Tulislah Program 3 berikut ini dan jalankan!

```
1  public class A {
2    public void MessageA() {
3          System.out.println("Info dari Class A");
4    }
5  }
```

```
1   public class B extends A {
2     public void MessageB() {
3           System.out.println("Info dari Class B");
4     }
5   }
```

```
1   public class C extends B {
2     public void MessageC() {
3           System.out.println("Info dari Class C");
4     }
5   }
```

Program 3. Contoh program hierarchical inheritance

Berdasarkan contoh Program 3 di atas, untuk melihat apakah class turunan yang paling bawah (`class C`) bisa mengakses properti dari class A dan class B maka Program 4 perlu dijalankan.

```
6   public class InheritanceTest {
7     public static void main(String[] args) {
8     A a = new A();
9     B b = new B();
10    C c = new C();
11
12    a.MessageA();
13    b.MessageB();
14    b.MessageA();
15    c.MessageC();
16    c.MessageB();
17    c.MessageA();
18    }
19  }
```

Program 4. Contoh pengaksesan properties pada multilevel inheritance

Program 4 menunjukkan bahwa object dari `class A` hanya bisa menggunakan method yang ada pada `class A` sendiri. Tetapi object dari `class B` bisa menggunakan method dari `class A`. Begitu juga dengan `class C` dimana object dari `class C` memiliki akses ke method yang ada pada `class A` dan `B`. Dengan demikian

dapat diambil kesimpulan bahwa induk class (Super class) tidak bisa mengakses variabel atau method dari anak class (sub class), tetapi sub-class memiliki akses pada variabel atau method dari super class selama variabel atau class tersebut tidak bersifat `private`.

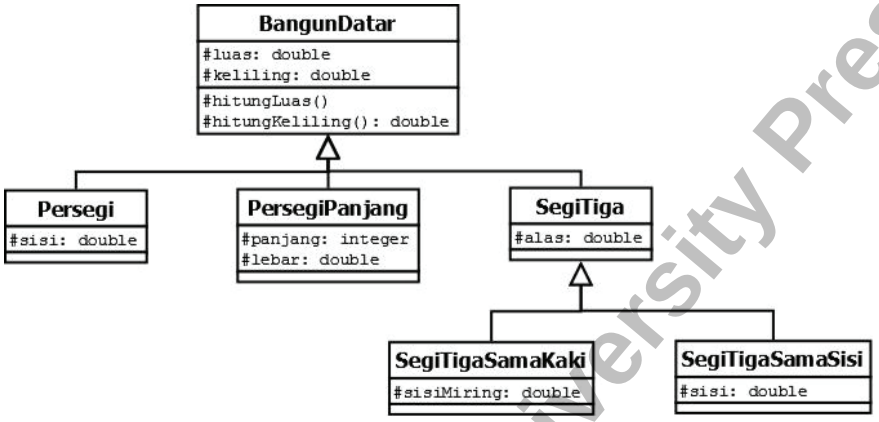
6.2. LATIHAN

1. Buatlah suatu class Kendaraan, yang memiliki minimal 3 instance variables yang memiliki nilai awal. Setelah itu, buatlah class Mobil dan Pesawat, yang masing-masing merupakan subclass dari Kendaraan.
2. Dari tiap subclass, buatlah 1 variable unique yang memiliki nilai awal, yang tidak terdapat pada superclass, dan 1 variable – memiliki nilai awal – memiliki nama yang sama dengan nama variable yang ada pada superclassnya.
3. Buatlah class dengan main method untuk menampilkan hasil dari kode diatas. Main method harus menampilkan nilai yang tersimpan pada seluruh instance variable yang terdapat pada subclass dan superclass.

   NB: Pada kasus soal nomor 3, variable yang ada di superclass hanya boleh diakses melalui subclassnya. Mahasiswa tidak diperkenankan mengakses langsung variable superclass dari object yang dihasilkan dari superclass.

6.3. TUGAS

Buatlah kode program berdasarkan Gambar 6.3 di bawah ini, untuk mengimplementasikan inheritance!



Gambar 6.3: UML Class Diagram

BAB 7
ENCAPSULATION

7.1. GAMBARAN UMUM

Encapsulation merupakan teknik untuk melindungi atau membungkus data/variabel supaya tidak dapat diakses langsung dari class lain. Dengan menggunakan encapsulation maka data bisa disembunyikan dari class lain yang tidak berhak memiliki akses. Untuk menerapkan teknik encapsulation ada beberapa hal yang harus dilakukan antara lain:

- Mendeklarasikan variabel dengan akses modifier **`private`**
- Menyediakan method yang memiliki akses modifier **`public`** sebagai **getter** dan **setter**. Fungsi dari getter adalah untuk menampilkan private variabel, sedangkan setter digunakan untuk memodifikasi nilai private variable.

Program 1 memperlihatkan penggunaan modifier `private` dalam encapsulation.

```
20  public class Karyawan {
21    private String nama;
22    private float gaji;
23    private int usia;
24
25    public String getNama() {
26          return nama; }
27
28    public void setNama(String nama) {
29          this.nama = nama; }
30
```

```
31    public float getGaji() {
32          return gaji; }
33
34    public void setGaji(float gaji) {
35          this.gaji = gaji; }
36
37    public int getUsia() {
38          return usia; }
39
40    public void setUsia(int usia) {
41          this.usia = usia; }
42  }
```

Program 1. Akses modifier `private` pada encapsulation

Variabel-variabel yang terdapat pada class `Karyawan` di atas tidak bisa diakses oleh class lain, meskipun class lain tersebut adalah class turunannya. Sehingga dalam hal ini untuk memanipulasi data, harus memanfaatkan **getter** dan **setter** yang disediakan oleh class tersebut. Contoh penggunaan method untuk memodifikasi data suatu class ditunjukkan Program 2 berikut ini.

```
43  public class KaryawanDemo {
44    public static void main(String[] args) {
45          Karyawan k1 = new Karyawan();
46          k1.setNama("Luffy");
47          k1.setGaji(2500000);
48          k1.setUsia(21);
49
50          System.out.println(k1.getNama());
51          System.out.println(k1.getGaji());
52          System.out.println(k1.getUsia());
53    }
54  }
```

Program 2. Pemanfaatan getter dan setter

Encapsulation juga bisa diterapkan pada class turunannya. Program 3 di bawah ini terdapat class `Manager` yang merupakan turunan dari class `Karyawan`. Tulis kembali class tersebut untuk

menambah pemahaman.

```
1   public class Manager extends Karyawan{
2     private float jamKerja = 7.5f;
3
4     public int jamKerja() {
5           return jamKerja;
6     }
7
8     public float getGajiManager( ) {
9           return getGaji() * 2;
10    }
11  }
```

Program 3. Contoh program single inheritance

7.2. LATIHAN

1. Buatlah class dengan fungsi `main()` dan 5 objek berbeda yang dihasilkan dari class `Manager`. Kemudian melalui object tersebut tampilkan nilai private variable dari class `Manager` dan class `Karyawan`.
2. Tambahkan **setter** pada class `Manager` untuk memodifikasi nilai baru `jamKerja` menjadi 8.5!

BAB 8
POLYMORPHISM

Salah satu kekuatan dari pemrograman berorientasi objek selain inheritance dan encapsulation, adalah polymorphism. Polymorphism merupakan salah satu kemampuan pemrograman berorientasi objek untuk membuat berbagai jenis bentuk dari suatu objek. Salah satu penggunaan polymorphism yang paling sering dipakai adalah ketika suatu induk class (*superclass*) melakukan referensi terhadap anak class (*subclass*).

Perlu diketahui bahwa salah satu cara untuk mengakses suatu objek adalah dengan membuat *reference variable*. Ketika suatu reference variable diciptakan maka hal itu tidak bisa dirubah menjadi bentuk yang lain. Reference variable itu sendiri dideklarasikan sebagai interface. Interface sama seperti class, namun hanya berisi konstanta dan metode abstract. Uraian lebih jelas mengenai interface akan dijelaskan pada Bab 10.

Perhatikan contoh berikut ini.

```
public interface Herbivora {}
public class Hewan{}
public class Kuda extends Hewan implements
Herbivora{}
```

Penggalan kode di atas menunjukkan bahwa class `Kuda` merupakan suatu bentuk polymorphism karena class tersebut memiliki multiple inheritance diantaranya,

1. Seekor Kuda adalah seekor Hewan
2. Seekor Kuda adalah Herbivora

3. Seekor Kuda termasuk class Kuda
4. Seekor Kuda merupakan sebuah objek

Sehingga kita bisa membuat objek reference seperti berikut :

```
Kuda k = new Kuda();
Hewan h = k;
Herbivora hb = k;
Object ob = k;
```

8.1. RUNTIME POLYMORPHISM

8.1.1. Overriding

Runtime Polymorphism merupakan proses pemanggilan method dengan cara melakukan overriding pada saat kode program berjalan. Overriding berhubungan dengan pewarisan (inheritance). Overriding pada program digunakan untuk mendefinisikan kembali metode yang sudah ada pada superclass. Sehingga memungkinkan metode pada subclass dan supeclass mempunyai cara kera yang berbeda. Program 1 sebagai contoh superclass yaitu class `Parentclaz`. Program 2 merupakan contoh subclass yaitu class `Childclaz`.

```
1  class Parentclaz{
2      void printOut(){
3          System.out.println("Ini adalah
   SuperClass");
4      }
5  }
```

Program 1. Superclass yang akan diterapkan konsep overriding

```
1  class Childclaz extends Parentclaz{
2      void printOut(){
3          System.out.println("Ini adalah
   SubClass");
4      }
5      void hello(){
```

```
6           System.out.println("Hello World");
7       }
8   }
```

Program 2. Penerapan overriding pada subclass

Program 3 class `DemoOverriding()` terdapat method `main()` untuk menjalankan kedua program di atas.

```
1   class DemoOverriding{
2     public static void main(String[] ovr){
3       Parentclaz pc = new Parentclaz();
4       Childclaz cc = new Childclaz();
5       pc.printOut();
6       cc.printOut();
7       }
8   }
```

Program 3. Menjalankan konsep overriding

Overriding berbeda dengan Overloading, Overriding hanya bisa diterapkan pada class yang memiliki hubungan inheritance. Perhatikan contoh penggunaan overriding method pada Program 4. Jalankan kode program di bawah ini dan pahami!

```
1   class Hewan{
2     public void jalan() {
3           System.out.println("Hewan bisa
    berjalan");
4     }
5   }
```

```
1   class Kucing extends Hewan{
2     public void jalan() {
3         System.out.println("Kucing bisa berjalan
    dengan"
4                       + "dan berlari");
5     }
6   }
```

Program 4. Contoh overriding method

Jika kita perhatikan kode Program 4 diatas, kita akan menemukan method `gerak()` pada class `Hewan` dan juga method `gerak()` pada class `Kucing`, akan tetapi implementasi yang diterapkan berbeda. Hewan bisa berjalan tetapi belum tentu bisa berlari namun Kucing yang merupakan inheritance dari Hewan bisa berjalan sekaligus berlari.

8.1.2. Upcasting

Program 5 dan Program 6 merupakan contoh penerapan konsep upcasting. Dalam hal ini class pertama adalah class `Sepeda` dan class kedua adalah class `SepedaAir`.

```
1   class Sepeda{
2       void run(){
3           System.out.println("Bisa berjalan");
4       }
5   }
```

Program 5. Superclass yang akan diterapkan konsep upcasting

```
1   class SepedaAir extends Sepeda{
2     void run(){
3        System.out.println("Hanya bisa berjalan di
    atas"
4     + "Air");
5      }
6      public static void main(String[] args) {
7        Sepeda s = new SepedaAir();//upcasting
8        s.run();
9      }
10  }
```

Program 6. Penerapan upcasting pada subclass

Sifat overriding diterapkan pada method yang dimiliki oleh `Sepeda`. Ketika kita memanggil method pada induk class dengan cara membuat objek reference, metode ini dilakukan oleh JVM bukan compiler. Sehingga ini disebut dengan runtime polymorphism.

Pada class `SepedaAir` terdapat method run yang merupakan method overriding dari class `Sepeda`. Namun pada saat implementasinya objek dari class `Sepeda` melakukan referensi ke class `SepedaAir` bukan ke class `Sepeda` itu sendiri, dinamakan upcasting.

8.1.3. Overloading

Overloading merupakan suatu mekanisme untuk mengembangkan fungsi dari suatu method, dimana beberapa method memiliki nama yang sama tetapi parameter yang dimiliki berbeda-beda. Overloading juga bisa berlaku terhadap constructor karena pada dasarnya constructor merupakan sebuah method. Overloading terhadap constructor merupakan suatu mekanisme pembuatan constructor yang memiliki bentuk lebih dari satu. Dalam hal ini pembeda antara satu constructor dengan constructor yang lain berupa jumlah parameter atau tipe parameter. Ketika kita melakukan overloading, java compiler akan membedakan masing-masing constructor berdasarkan jumlah parameter dan tipe datanya. Sehingga sangat mungkin jika kita memiliki constructor dengan jumlah parameter yang sama tetapi memiliki tipe data yang berbeda atapun sebaliknya.

```
1   import java.util.Date;
2   public class OverloadingConstructor{
3      int idUser;
4      String userName;
5      int level;
6      Date lastLogin;
7      public OverloadingConstructor() {
8
9      }
10     public OverloadingConstructor(int idUser,
11  String userName) {
12       this.idUser = idUser;
13       this.userName = userName;
14     }
15  }
```

Program 7. Overloading pada constructor

Program 7 memperlihatkan dua constructor yang memiliki jumlah parameter berbeda. Ketika kita membuat object dari class diatas maka nilai default dari object tersebut bisa memiliki dua kemungkinan yaitu `NULL` atau, `idUser` dan `userName`.

8.2. LATIHAN

Silahkan tulis dan jalankan Program 8 berikut ini! Kemudian kerjakan soal setelahnya!

```
16  public class Pet{
17      private String nama;
18
19      public void beriNama(String beriNama){
20          this.nama = beriNama;
21      }
22
23      public String panggilNama(){
24          return this.nama;
25      }
26
27      public String perilaku(){
28          return "Hewan Penurut";
29      }
30  }
```

Program 8. Latihan menerapkan konsep polymorphism

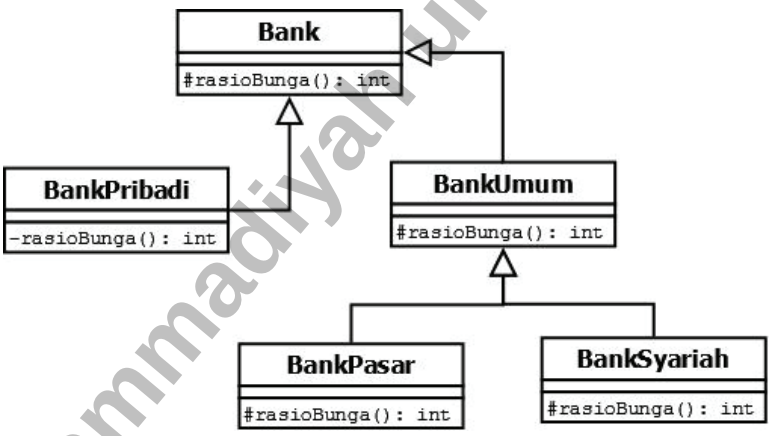
1. Buatlah class `Kucing` dan `Anjing` dimana kedua class tersebut melakukan overriding terhadap method `perilaku()`!
2. Tambahkan satu method pada masing-masing class yang secara khusus hanya berlaku pada masing-masing class tersebut.
3. Buat class `TestPolymorphism` sehingga keluaran program seperti berikut ini!

```
1   TOM
2   Menyukai Ikan
3   Meeooww... Meeooww
4   BULL
```

```
5   Menyukai Daging dan Tulang
6   Guk..Guk..Guk..
```

8.3. TUGAS

1. Lihat kembali Program 4 mengenai overriding, buatlah class `Elang` yang memiliki method `jalan()` namun implementasinya berbeda dari kedua class lainnya!
2. Buatlah class baru dengan nama `CustomerData`, tambahkan variabel nama, alamat, tanggal lahir, pekerjaan dan gaji. Selanjutnya buatlah overloading constructor dari class tersebut.
3. Buatlah class baru dengan method `main()` yang disertai 10 object customer dari class `CustomerData`.
4. Buatlah class berdasarkan diagram UML berikut ini! Terapkan teknik polymorphism dan tampilkan hasil output program (*screenshot*)!



BAB 9
ABSTRACT CLASS

Suatu saat class dibuat untuk menyelesaikan program dengan kerumitan yang tinggi. Class tersebut berisi metode dan variable, serta tidak pernah digunakan untuk membuat objek, objek dapat dibuat dari class lain yang menjadi turunannya. Class inilah yang disebut dengan class abstract, yaitu class yang tidak dapat dibuat objeknya.

Java menyediakan suatu mekanisme yang memungkinkan suatu method ditentukan didalam class tetapi tidak disertai dengan definisinya. Method seperti ini dikenal dengan method abstract dan classnya dinamakan class abstract. Definisi class ditentukan masing-masing subclass. Dalam hal ini setiap subclass dari class abstract harus mendefinisikan seluruh method abstract pada super class. Class abstract dapat dibentuk apabila subclass mempunyai operasi yang sesuai dengan method tersebut tetapi antara satu subclass denga subclass yang lain memiliki tindakan yang berbeda.

Class abstract di dalam Java selalu ditandai dengan kata kunci abstract. Ciri-ciri class abstract adalah sebagai berikut :

1. Class abstract tidak bisa dibuat objek (*instance of object*).
2. Class abstract bisa memiliki abstract method maupun tidak.
3. Jika kita membuat method abstract maka kita harus membuatkan implementasi dari method tersebut.
4. Method abstract tidak memiliki body

9.1. OBJECT CLASS ABSTRACT

Seperti uraian sebelumnya, class abstract tidak bisa digunakan untuk membuat objek. Sehingga untuk menggunakan fitur dari class abstract, perlu membuat class yang merupakan class turunan dari class abstract tersebut.

```
1   abstract class AbstrakClass {
2       int a;
3       int b;
4       int c;
5       public AbstrakClass(int a, int b, int c){
6           this.a = a;
7           this.b = b;
8           this.c = c;
9       }
10      public int kali(){
11          return a*b*c;
12      }
13  }
```

Program 1. Contoh class abstract

Kode Program 1 di atas merupakan sebuah class abstract yang memiliki konstruktor dengan tiga parameter. Program 2 di bawah ini membuktikan bahwa suatu class abstract tidak dapat dibuat objek.

```
1   public class ObjekAbstrakClass{
2       public static void main(String [] args){
3           AbstrakClass ac =new AbstrakClass(2, 3,
    4);
4       }
5   }
```

Program 2. Class abstract tidak dapat dibuat object-nya

Jika kita menjalankan kode Program 2, maka bisa dipastikan terjadi kesalahan (error). Error yang muncul disebabkan karena ada pembuatan objek dari class `AbstrakClass`. Sehingga untuk menggunakan class abstract agar bisa diimplentasikan kita perlu

membuat kode program yang merupakan turunan dari class abstract tersebut seperti pada Program 3 berikut ini.

```
1   public class TurunanAbstrakClass extends
    AbstrakClass {
2       int x;
3
4       public TurunanAbstrakClass (int a, int b,
    int c){
5           super(a,b,c);
6           x = a+b+c;
7       }
8
9       public void printX(){
10          System.out.println(x);
11      }
12  }
```

Program 3. Implementasi class abstract melalui subclass

Kode `TurunanAbstrakClass` merupakan turunan dari `AbstrakClass`, dengan cara seperti ini kita bisa menggunakan variabel atau method dari class abstract. Tulis dan amati kode berikut!

```
1   public class ObjekAbstrakClass {
2     public static void main(String[] args) {
3       TurunanAbstrakClass tac =
4   new TurunanAbstrakClass(2, 3, 2);
5       tac.printX();
6       System.out.println(tac.kali());
7     }
8   }
```

Program 4. Penggunaan propertis class abstract melalui object subclass

9.2. METHOD ABSTRACT

Sebuah class abstract dapat memiliki method abstract maupun method **bukan** abstract. Method abstract merupakan method

yang hanya memiliki nama method saja tanpa menjelaskan rutin atau fungsi detail implementasi dari method tersebut. Sedangkan method bukan abstract merupakan method yang memiliki detail implementasi didalamnya. Program 5 memperlihatkan method abstract pada suatu class.

```
1   public abstract class methodAbstract {
2       public abstract int luas();
3       public abstract int keliling();
4
5       public int getLuas (){
6           return luas();
7       }
8
9       public int getKell (){
10          return keliling();
11      }
12  }
```

Program 5. Class yang mengandung method abstract

Class abstract pada Program 5 memiliki dua method abstract yaitu luas dan keliling, dimana method tersebut tidak memiliki detail implementasi tentang bagaimana menentukan nilai luas dan keliling. Untuk mengimplementasikannya, cara yang dilakukan adalah sama seperti pada contoh sebelumnya, yaitu kita harus membuat suatu class yang merupakan class turunan dari class abstract tersebut. Dengan demikian kita dapat menentukan detail dari method abstract. Misalkan dalam hal ini kita akan membuat class `Persegi` pada Program 6. Selanjutnya Program 7 merupakan class untuk membentuk objek dari class `Persegi` dan memanggil method abstract.

```
1   public class Persegi extends methodAbstract {
2       int sisi=5;
3
4       public int luas() {
5           return sisi*sisi;
6       }
```

```
7
8       public int keliling() {
9           return 4*sisi;
10      }
11  }
```

Program 6. Implementasi method abstract oleh subclass

```
1   public class MethodMain {
2    public static void main(String[] args) {
3     Persegi psg = new Persegi();
4     System.out.println("Keliling = "+psg.
    getKell());
5     System.out.println("Luas = "+psg.getLuas());
6    }
7   }
```

Program 7. Penggunaan propertis method abstract melalui object subclass

9.3. LATIHAN

Dengan menggunakan class `MethodAbstrak` pada Program 5 di atas, buatlah class `PersegiPanjang`, `JajarGenjang`, `Lingkaran`, dan `Segitiga`! Selanjutnya implementasikan method `luas()` dan `keliling()` yang sesuai dengan perhitungan masing-masing class.

9.4. TUGAS

Buatlah class `abstract` untuk bangun ruang, dengan ketentuan memiliki method `abstract` untuk menghitung volume, dan luasSelimut/luasPermukaan. Selanjutnya buatlah class `Balok`, `Kubus`, `Bola`, `Kerucut`, dan `PrismaSegitiga` untuk mengimplementasikan method `abstract` tesebut!

BAB 10
INTERFACE

Java menyediakan teknik untuk berbagi konstanta atau menentukan bentuk method yang dapat digunakan pada beberapa class. Teknik ini disebut dengan interface. Sekilas interface mirip dengan class abstract, namun interface seperti signature atau kontrak kerja – berisikan operas-operasi / kumpulan methods dan konstanta yang dapat dikerjakan oleh suatu class tertentu dan harus diimplementasikan.

Akan tetapi interface memiliki beberapa perbedaan dengan class pada umumnya seperti :

1. Interface tidak bisa digunakan untuk membuat objek
2. Interface tidak bisa memiliki konstruktor
3. Secara default semua method di dalam interface bersifat abstract
4. Variabel pada interface harus bertipe `static final`
5. Interface tidak bisa diturunkan kepada sebuah class, tetapi interface bisa diimplementasikan menggunakan `implements`
6. Sebuah interface bisa diturunkan ke sesama interface menggunakan `extends`

10.1. DEKLARASI INTERFACE

Secara default interface bersifat abstract sehingga kita tidak perlu menuliskan kata kunci `abstract` ketika membuat interface. Semua method pada interface juga bersifat abstract sehingga tidak

perlu menuliskan kata kunci `abstract` dan semua method bersifat `public`. Deklarasi sebuah interface ditunjukkan pada Program 1. Contoh sebuah interface sederhana diperlihatkan Program 2.

```
1   public interface NamaInterface{
2       //variabel static final
3       //method abstract
4   }
```

Program 1. Deklarasi interface

```
1   public interface ActivityAnimal {
2       public void eat();
3       public void travel();
4   }
```

Program 2. Contoh sebuah interface sederhana

10.2. IMPLEMENTASI INTERFACE

Ketika sebuah class mengimplementasikan interface, kita bisa berasumsi bahwa class tersebut memiliki kontrak kerja dengan interface untuk mengimplementasikan seluruh method yang ada pada interface. Jika class tersebut tidak mengimplementasikan method dari interface maka akan terjadi *error*. Agar interface bisa digunakan, kita perlu menggunakan kata kunci `implements` yang mengindikasikan bahwa class tersebut mengimplementasikan sebuah interface seperti diperlihatkan Program 3.

```
1   public class Mamalia implements ActivityAnimal{
2
3       public void eat() {
4           System.out.println("Mammal eats");
5       }
6
7       public void travel() {
8           System.out.println("Mammal travels");
9       }
10
```

```
11      public int noOfLegs(){
12          return 0;
13      }
14
15      public static void main(String[] args) {
16          Mamalia m = new Mamalia();
17          m.eat();
18          m.travel();
19      }
20  }
```

Program 3. Class yang mengimplementasikan sebuah interface

10.2.1. Percobaan

Buat interface dengan nama `ActivityLampu` seperti pada program 4 berikut ini!

```
1   public interface ActivityLampu {
2       public static final int LAMPU_HIDUP=1;
3       public static final int LAMPU_MATI=0;
4       public abstract void matikanLampu();
5       public abstract void hidupkanLampu();
6   }
```

Program 4. Interface `ActivityLampu` dengan method dan variabelnya

Selanjutnya buat class `Lampu` yang merupakan implementasi dari interface `IntLampu` seperti Program 5 berikut ini!

```
1   public class Lampu implements ActivityLampu{
2     public  int statusLampu;
3
4     @Override
5     public void matikanLampu() {
6        if(statusLampu==0){
7          System.out.println("Lampu sudah dalam
    kondisi mati");
8        }else if(statusLampu==1){
9          statusLampu=-1;
```

```
10          System.out.println("Lampu sudah
    dimatikan");
11        }
12     }
13
14     @Override
15     public void hidupkanLampu() {
16        if(statusLampu==1){
17          System.out.println("Lampu sudah
    dinyalakan\n***");
18        }else {
19          statusLampu=+1;
20          System.out.println("Lamu sudah dalam
    kondisi menyala");
21        }
22     }
23     public int setSaklar(int saklar){
24        return statusLampu = saklar;
25     }
26  }
```

Program 5. Class `Lampu` mengimplementasikan interface

```
1   public class AplikasiLampu {
2       public static void main(String[] args) {
3           Lampu lampu = new Lampu();
4           Scanner sc = new Scanner(System.in);
5           lampu.statusLampu = lampu.setSaklar(0);
6           System.out.println("Status Lampu =
    "+lampu.statusLampu+"\nketikkan");
7           System.out.println("1 untuk menyalakan
    lampu\n0 untuk mematikan lampu");
8
9           if(lampu.setSaklar(sc.nextInt())==0){
10              lampu.matikanLampu();
11          }else {
12              lampu.hidupkanLampu();
13          }
14      }
15  }
```

Program 6. Fungsi `main()` untuk menjalankan program interface

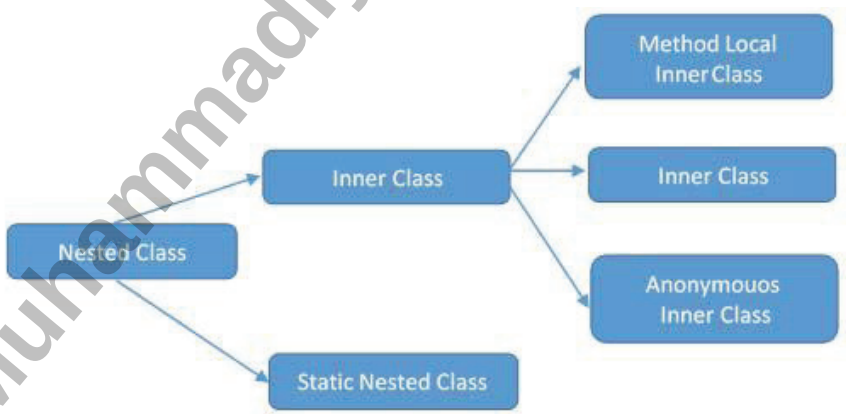
10.3. TUGAS

Modifikasi class `Lampu` di atas dengan menambahkan satu variabel `static final LAMPU_REDUP`, dan tambahkan method untuk meredupkan lampu. Selanjutnya buat class dengan fungsi `main()` untuk menjalankannya!

BAB 11
NESTED CLASS

Nested class merupakan fasilitas yang disediakan oleh Java untuk mengelompokkan class secara virtual sehingga lebih mudah dibaca dan dikelola. Selain itu nested class juga memungkinkan outer class bisa mengakses variabel `private` dan method `private` dari inner class.

Suatu class sangat memungkinkan untuk kita membuat class lain, dimana class-class tersebut memiliki kaitan fungsi atau kegunaan. Nested class sendiri dikategorikan menjadi dua bagian yaitu : Inner class (Non-Static Nested class) dan Static Nested class. Struktur dari pengelompokan inner class dapat dilihat dari Gambar 11.1 berikut ini.



Gambar 11.1 Kategori nested class

11.1. INNER CLASS (Non-Static Nested class)

Non-static nested class lebih sering disebut sebagai inner class. Secara umum class tidak boleh memiliki tipe akses `private`. Tetapi jika menggunakan inner class maka class tersebut dapat bertipe akses `private`. Inner class dapat dibuat secara sederhana dengan membuat sebuah class di dalam class lain. Program 1 adalah contoh inner class.

```
1   public class Outer_Demo {
2       class Nested_Demo{
3       }
4   }
```

Program 1. Contoh inner class

Program 2 memperlihatkan penggunaan inner class dapat dilakukan dengan cara mengakses method yang ada didalamnya. `OuterDemo` merupakan class paling luar yang di dalamnya memiliki inner class yaitu `InnerDemo`, dan satu method dengan nama `displayInner()` untuk mengakses variabel dari class `InnerDemo`. Method `displayInner()` merupakan method yang akan kita gunakan untuk mengakses method dari inner class, dan terlihat bahwa kita harus membuat object dari inner class untuk mengakses method di dalamnya. Program 3 merupakan class dengan method `main()` untuk mengetahui output atau menjalankan (run) inner class.

```
1   public class OuterDemo {
2     int num;
3
4     //inner class
5     private class InnerDemo{
6       private void print(){
7       System.out.println("Ini merupakan method
    inner class");
8       }
9     }
10
```

```
11   //akses method inner class dari method outer
   class
12    void displayInner(){
13      InnerDemo inner = new InnerDemo();
14      inner.print();
15    }
16  }
```

Program 2. Pengaksesan properties pada inner class

```
1   public class Myclass {
2       public static void main(String[] args) {
3
4           //membuat object outer class
5           OuterDemo outer = new OuterDemo();
6
7           //mengakses method outer class
8           outer.displayInner();
9       }
10  }
```

Program 3. Menjalankan (run) inner class melalui metode `main()`

11.2. MENGAKSES PRIVATE MEMBER

Seperti yang sudah disebutkan sebelumnya bahwa inner class bisa juga memiliki tipe akses `private`. Untuk mengakses variabel `private` tersebut, kita perlu membuat object dari outer class, diikuti object dari inner class seperti di contoh pada potongan kode program di bawah ini.

```
OuterDemo outer = new OuterDemo();
OuterDemo.InnerDemo inner = outer.new
InnerDemo();
```

Program 4 merupakan cara untuk mengimplementasikan pengaksesan variabel `private`. Program 5 merupakan class yang berisi method `main()` untuk menjalankan (*run*) inner class.

```
1   public class OuterDemo2 {
2    private int num = 175;
3    public class Inner_Demo{
4      public int getNum(){
5         System.out.println("Ini adalah nilai dari
    variable private outerDemo");
6         return num;
7      }
8    }
9   }
```

Program 4. Mengakses variable `private`

```
1   public class OuterDemoMain {
2     public static void main(String[] args) {
3        //Membuat Object OuterDemo
4        OuterDemo2 outer = new OuterDemo2();
5
6        //Membuat Object InnerDemo
7        OuterDemo2.Inner_Demo inner =
8                                  outer.new Inner_
    Demo();
9        System.out.println(inner.getNum());
10    }
11  }
```

Program 5. Class dengan method main() untuk mengakses variable `private`

11.3. STATIC NESTED CLASS

Static nested class merupakan nested class yang bertipe `static`. Class dengan tipe ini bisa diakses oleh outer class tanpa harus membuat objectnya. Static nested class hanya dapat mengakses static member (variable / method) dari outer class. Static nested class seperti terlihat pada kerangka program di bawah ini.

```
public class MyOuter {
    static class Nested_Demo{
```

```
    }
}
```

Membuat object static nested class sedikit berbeda dengan membuat object dari inner class. Program 6 adalah contoh membuat *instance* dari static nested class.

```
1   public class Outer {
2       static class NestedDemo{
3           public void myMethod(){
4               System.out.println("Ini adalah
    method "
5   + "static nested class");
6           }
7       }
8
9       public static void main(String[] args) {
10          Outer.NestedDemo nested =
11  new Outer.NestedDemo();
12          nested.myMethod();
13      }
14  }
```

Program 6. Membuat instance dari static nested class

11.4. LATIHAN

1. Perhatikan Program 7 di bawah, dan isikan Nama Saudara dan NIM pada variabelnya! Buatlah method di dalam class **`StaticNestedClass`** untuk mengakses method `printNama()`!
2. Buatlah method di dalam **`InnerClass`** untuk mengakses variabel `jurusan`!
3. Buatlah class dengan fungsi **`main()`** untuk menampilkan hasil dari kode program saudara!

```
1   class NestedClass {
2       String nama=""; //lengkapi dengan nama
    saudara
3       String nim=""; //lengkapi dengan nim
    saudara
4
5       public void printNama(){
6           System.out.println(nama+" : "+nim);
7       }
8
9       static class StaticNestedClass {
10          static String jurusan="informatika";
11      }
12
13      class InnerClass{
14
15      }
16  }
```

Program 7. Soal latihan nested class

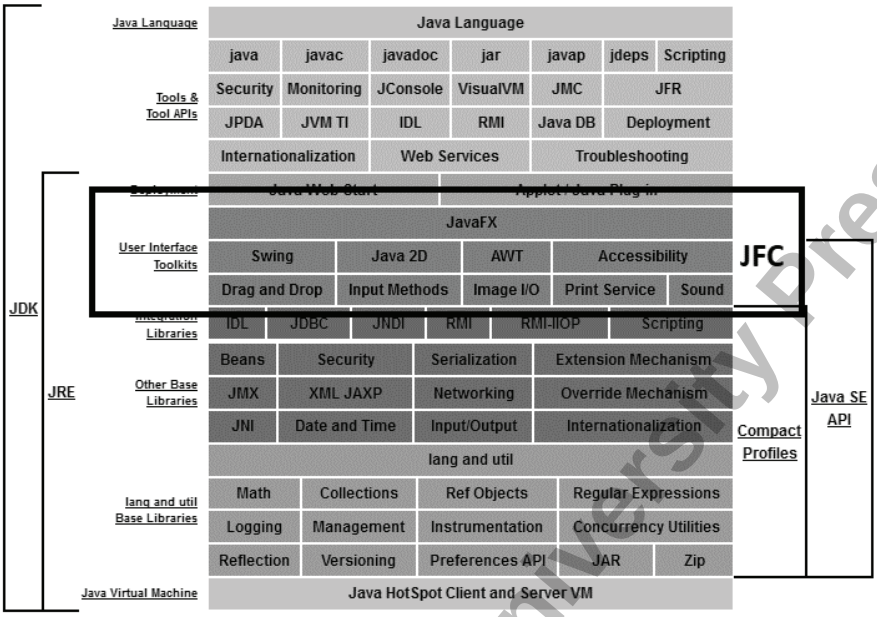
BAB 12
GRAPHICAL USER INTERFACE (GUI)

Graphical User Interface (GUI) adalah interaksi yang dilakukan oleh user melalui menu dan simbol / *icon* yang ditampilkan dalam grafis. GUI dalam Java digunakan untuk memudahkan penggunaan komponen yang sulit dilakukan ketika pemrograman berbasis teks seperti tombol, dialog box, menu atau gambar.

Java memiliki mempunyai dua *graphical library*, yaitu AWT (*Abstract Windows Toolkit*) dan Swing. AWT dibuat pada tahun 1994 merupakan library untuk membangun grafis secara sederhana. Tahun 1997 Java Foundation Class (JFC) muncul sebagai pengembangan dari AWT dan didalamnya terkandung kode Swing. Sehingga Java menyediakan dua graphical library, yaitu AWT dengan `package java.awt.*` dan Swing dengan `package javax.swing.*`. *Library* yang termasuk dalam JFC atau *user interface* seperti pada Gambar 11.1.

Tidak seperti beberapa komponen AWT yang menggunakan *native code*, keseluruhan Swing ditulis menggunakan bahasa pemrograman Java. Swing menyediakan implementasi *platform-independent* dimana aplikasi yang dikembangkan dengan platform yang berbeda dapat memiliki tampilan yang sama.

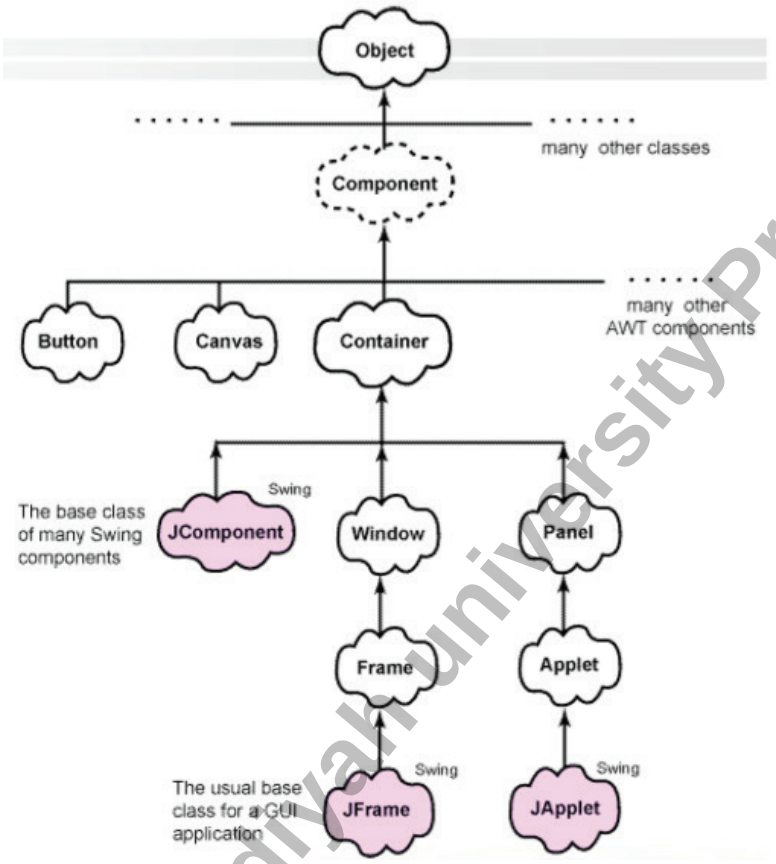
Nama dari komponen GUI milik Swing hampir sama persis dengan komponen GUI milik AWT. Perbedaan jelas terdapat pada penamaan komponen. Pada dasarnya, nama komponen Swing sama dengan nama komponen AWT tetapi dengan tambahan huruf J pada awalannya. Sebagai contoh, satu komponen dalam AWT adalah button class. Sedangkan pada Swing, nama komponen tersebut menjadi class.`Jbutton`



Gambar 11.1. Java Foundation Class pada platform Java SE

Program GUI terdiri dari tiga bagian yaitu (1) *Graphical Components*, yang merupakan sarana membentuk antar muka pengguna dalam bentuk grafis (objek dari Swing). (2) *Listener methods*, yaitu menerima event (tindakan) dan meresponnya. (3) *Application methods*, yaitu melakukan komputasi sesuai aksi yang telah dilakukan pengguna. Komputasi dalam hal menerima data dari GUI, hingga mengirim hasil olah kembali ke GUI untuk ditampilkan.

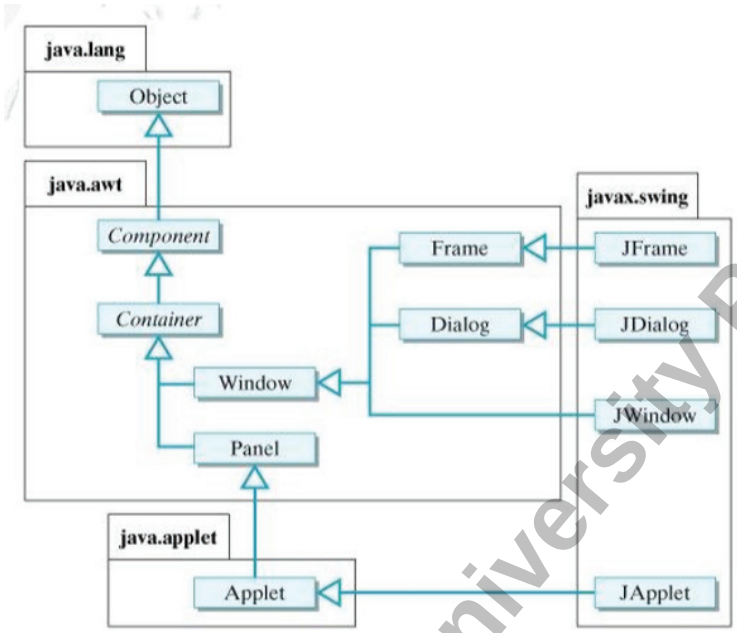
Gambar 11.2. menunjukkan hirarki dari AWT. Class `Component` merupakan class dasar (fundamental) untuk AWT. Sedangkan `JComponent` merupakan turunan dari class `Container` dan salah satu class dasar (fundamental) dari Swing.



Gambar 11.2. Hirarki dari Graphical Object

12.1. AWT (*Abstract Window Toolkit*)

Java AWT terdiri dari beberapa komponen GUI yang telah lama dikembangkan oleh Java. Program GUI adalah kumpulan komponen grafis yang ditempatkan kedalam satu atau lebih window. Container adalah objek yang dapat diisi komponen-komponen GUI tersebut. Gambar 11.3. menunjukkan *library* yang ada pada AWT.

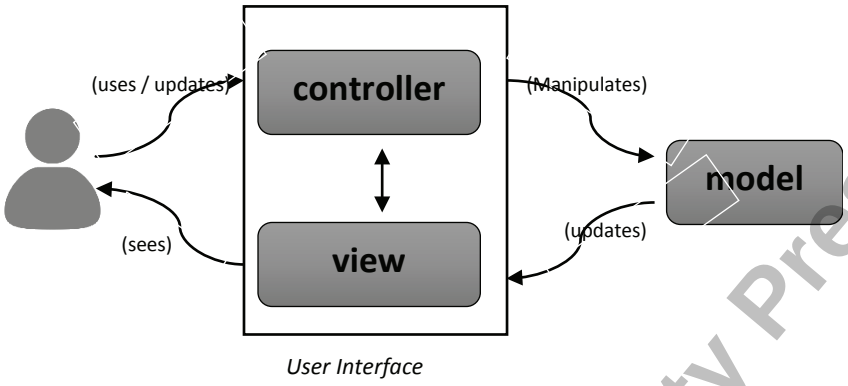


Gambar 11.3. AWT library

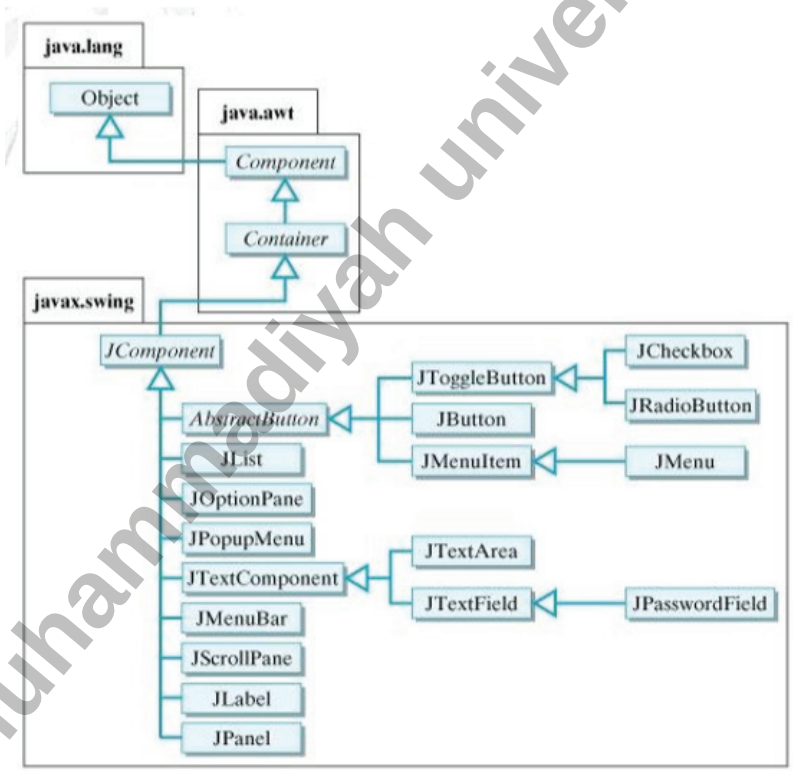
12.2. SWING

Swing menggunakan system interaksi dengan *Model View Controller* (MVC) seperti terlihat pada Gambar 11.4. Class-class *library* yang terkandung dalam Swing ditunjukkan pada Gambar 11.5.

- *Model* yang berarti representasi data pada aplikasi
- *View* yang berarti memberikan tampilan output dan input. Sebuah model yang sama dapat mempunyai tampilan yang berbeda.
- *Controller* yang berarti untuk melakukan manipulasi pada model menggunakan interaksi user. Model dapat menentukan tampilan sesuai perintah Controller.



Gambar 11.4. Model View Controller



Gambar 11.5. Swing library

12.3. KOMPONEN GUI

Komponen GUI merupakan objek grafis yang digunakan untuk menampilkan data, menerima masukan, atau menunjukkan kondisi tertentu. Berikut ini beberapa contoh komponen GUI yang akan dicoba dalam praktikum kali ini. Selain komponen di bawah, masih banyak komponen lain yang mendukung user interface.

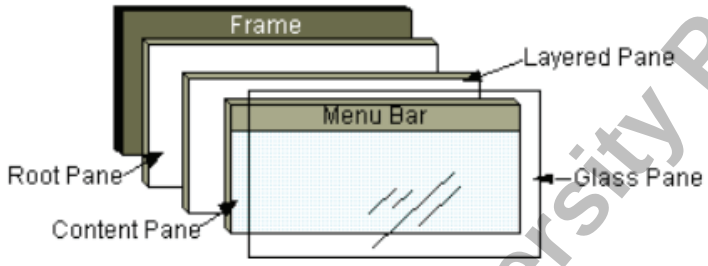
a. **JFrame** merupakan komponen terbawah dalam tampilan grafik Swing. Frame dapat ditampilkan dengan cara membuat class baru yang diturunkan JFrame.
b. **JPanel** merupakan turunan JComponent, class *Container* sederhana tetapi bukan top-level. Korespondensi class dalam package AWT.
c. **JCheckBox** adalah item yang dapat dipilih atau tidak oleh pengguna. Korespondensi pada
d. **JTextField** merupakan area yang dapat menampung tulisan dalam satu baris yang diketik oleh pengguna.
e. **JTextArea**, merupakan tempat pengeditan teks yang dapat menampung lebih dari satu baris.
f. **JButton** merupakan komponen GUI yang menyerupai tombol. Ketika tombol ini di klik (pilih) maka perintah tertentu akan dijalankan.
g. **JLabel,** merupakan komponen grafik yang dapat menampung tulisan dan/atau icon.

12.4. PENGATURAN CONTAINER TINGKAT ATAS (TOP-LEVEL CONTAINER)

Seperti disebutkan diatas, *top-level containers* seperti `JFrame` dan `JApplet` dalam Swing sangat tidak cocok dengan AWT. Ini adalah syarat menambahkan komponen ke dalam kontainer. Jika Anda ingin menambahkan langsung sebuah komponen kedalam kontainer sebagai kontainer AWT, pertama-tama Anda telah mendapatkan content pane dari kontainer. Untuk melakukan hal tersebut, Anda akan menggunakan metode `getContentPane()`

dari kontainer. Urutan container diperlihatkan pada Gambar 11.6.

Perlu diperhatikan pada package `java.awt` masih saja diimpor karena layout manager yang digunakan terdapat pada package tersebut. Juga, memberi judul pada frame dan mengepack komponen di dalam frame dapat juga dilakukan untuk frame AWT.



Gambar 11.6 . Urutan level rootpane

12.5. LATIHAN

Hal yang perlu dilakukan pertama kali adalah melakukan `import java.awt.*` dan `javax.swing.*`

12.5.1. Frame

Hal-hal yang berkaitan dalam membangun `JFrame` adalah :

- Judul frame dibuat dengan menjalankan konstruktor pada JFrame menggunakan perintah super("Judul Frame").
- Ukuran frame diatur dengan metode `setSize()`.
- Ketika frame ditutup dan aplikasi Java juga akan tertutup/berhenti maka dapat menggunakan metode `setDefaultCloseOperation()`.
- Agar frame ditampilkan pada layar computer (output) maka perlu mengatur metode `setVisible()` dengan parameter `TRUE`.

Implementasi frame dapat dilihat pada Program 1 berikut ini.

```
1   public class Utama extends javax.swing.JFrame{
2       public Utama(){
3           super("Belajar mengenal GUI");
4           setSize(300,100);
5           setDefaultCloseOperation(EXIT_ON_CLOSE);
6           setVisible(true);
7           setLocationRelativeTo(null);
8       }
9       public static void main(String[] args) {
10          Utama u = new Utama();
11      }
12  }
```

Program 1. Implementasi JFrame

a. Jalankan dan perlihatkan hasil program saudara!

b. Lengkapi pengertian implementasi class `JFrame` pada tabel berikut ini!

Konstruktor	Keterangan
`JFrame()`	...
`JFrame(String Judul)`	...

Metode	Keterangan
`void setSize(int lebar, int tinggi)`	...
`void setLocation(int x, int y)`	...
`void setVisible(Boolean)`	...
`void setLocationRelativeTo(Component)`	...

12.5.2. Button

Program 2 adalah kode untuk menampilkan tombol dengan ukuran tertentu. Tombol ini diletakkan dalam class `FrameA` yang diturunkan dari `JFrame`.

```
1   import java.awt.Container;
2   import javax.swing.JButton;
3   public class FrameA extends javax.swing.JFrame{
4       public FrameA(){
5           super("Frme dan Button");
6           setSize(100, 50);
7           setDefaultCloseOperation(EXIT_ON_CLOSE);
8           setVisible(true);
9           setLocationRelativeTo(null);
10      }
11      public static void main(String[] args) {
12          FrameA fa = new FrameA();
13          Container kontainer =
    fa.getContentPane();
14          JButton jbtOK = new JButton("OK");
15          kontainer.add(jbtOK);
16      }
17  }
```

Program 2. Implementasi JButton

a. Jalankan dan perlihatkan hasil program saudara!

b. Lengkapi pengertian implementasi class `JButton` pada tabel berikut ini!

Konstruktor	Keterangan
`JButton()`	…
`JButton(String teks)`	…
`JButton(Icon icon)`	…
`JButton(String teks, Icon icon)`	…

12.5.3. Container

`JFrame` dapat diisi dengan komponen-komponen GUI setelah diletakkan container. Container adalah komponen khusus yang berguna untuk menampung komponen GUI lainnya. Berikut ini

contoh dua buah tombol dengan container `JPanel`. Tombol kedua menggunakan gambar icon. Saudara siapkan gambar terlebih dahulu dan letakkan di dalam folder yang sama dengan file Java (`FrameB.java`) yang sedang dikompilasi dan coba melakukan pemrograman seperti pada Program 3 berikut ini.

```
1   import java.net.URL;
2   import javax.swing.ImageIcon;
3   import javax.swing.JButton;
4   import javax.swing.JPanel;
5   public class FrameB extends javax.swing.JFrame{
6       public FrameB(){
7           super("Frame dan Button");
8           setSize(500,500);
9           setDefaultCloseOperation(EXIT_ON_CLOSE);
10          setLocationRelativeTo(null);
11      }
12      public static void main(String[] args) {
13          FrameB fb = new FrameB();
14          JPanel panel = new JPanel();
15          URL img =
16  FrameB.class.getResource("shakehand2.png");
17          JButton jbtOK = new JButton("OK");
18          JButton jbtImg = new JButton(new
    ImageIcon(img));
19          panel.add(jbtOK);
20          panel.add(jbtImg);
21          fb.add(panel);
22          fb.setVisible(true);
23      }
24  }
```

Program 3. Implementasi container

Jalankan dan perlihatkan hasil program saudara!

12.5.4. Label

Label merupakan komponen grafik yang dibuat untuk menampung tulisan atau icon. Program 4 merupakan implementasi

JLabel dengan menggunakan class Utama yang telah dicoba sebelumnya pada Program 1.

```
1   public class DemoLabel {
2       public static void main(String[] args) {
3           Utama u = new Utama();
4           u.setSize(500,500);
5           URL img = FrameB.class.
    getResource("shakehand.png");
6           ImageIcon ikon = new ImageIcon(img);
7           JLabel label = new JLabel("Label", ikon,
    SwingConstants.CENTER);
8           JPanel panel = new JPanel();
9           panel.add(label);
10          u.add(panel);
11      }
12  }
```

Program 4. Implementasi JLabel

a. Jalankan dan perlihatkan hasil program saudara!

b. Lengkapi pengertian implementasi class JLabel pada table berikut ini

Konstruktor	Keterangan
JLabel (String teks)	...
JLabel (String teks, int i)	...
JLabel (String teks, Icon ic, int i)	...

12.5.5. TextField dan Password Field

JTextField merupakan komponen GUI yang dapat menampung Tulisan yang diketik pengguna program. JPasswordField digunakan untuk menampung masukan namun isi tulisan disembunyikan. Program 5 adalah kode untuk implementasi keduanya.

```
1   import javax.swing.JLabel;
2   import javax.swing.JPanel;
3   import javax.swing.JPasswordField;
4   import javax.swing.JTextField;
5   public class DemoTextField {
6       public static void main(String[] args) {
7           Utama u = new Utama();
8           JLabel nama = new JLabel("Nama : ");
9           JLabel password = new JLabel("Password :
    ");
10          JTextField inputNama = new
    JTextField(15);
11          JPasswordField inputpsw = new
12  JPasswordField(15);
13          JPanel panel = new JPanel();
14          panel.add(nama);
15          panel.add(inputNama);
16          panel.add(password);
17          panel.add(inputpsw);
18          u.add(panel);
19
20      }
21  }
```

Program 5. Implementasi JTextField dan JPasswordField

a. Jalankan dan perlihatkan hasil program saudara!

b. Lengkapi pengertian implementasi class `JTextField` pada tabel di bawah ini.

Konstruktor	Penjelasan
`JTextField()`	...
`JTextField(int i)`	...
`JTextField(String i)`	...
`JTextField(String teks, int i)`	...

Parameter dalam class JTextField	Penjelasan
String Text	...
Boolean Editable	...
Int columns	...
Int horizontalAlignment	...

12.5.6. Radio Button dan CheckBox

JRadioButton digunakan untuk membuat satu pilihan dari sekian banyak pilihan yang tersedia. JCheckBox digunakan untuk membuat beberapa pilihan sekaligus. JRadioButton mempunyai konstruktor yang mirip dengan JCheckBox. Program 6 merupakan implementasi dari penggunaan komponen GUI JRadioButton.

```
1   public class DemRadioButton {
2       public static void main(String[] args) {
3           Utama u = new Utama();
4           u.setSize(1000, 100);
5
6           JRadioButton[] teams = new
    JRadioButton[4];
7           teams [0] = new JRadioButton("Sangat
    tidak setuju");
8           teams [1] = new JRadioButton("Tidak
    setuju");
9           teams [2] = new JRadioButton("Kurang
    setuju");
10          teams [3] = new JRadioButton("Setuju",
    true);
11          teams [3] = new JRadioButton("Sangat
    Setuju");
12
13          JPanel panel = new JPanel();
```

```
14          JLabel Pernyataan = new JLabel("Modul
    Praktikum Pemrograman Berorientasi Objek jelas
    dan mudah.");
15          panel.add(Pernyataan);
16
17          ButtonGroup group = new ButtonGroup();
18          for(int i = 0; i<teams.length; i++){
19              group.add(teams[i]);
20              panel.add(teams[i]);
21          }
22
23          u.add(panel);
24          u.setVisible(true);
25      }
26  }
```

Program 6. Implementasi JRadioButton

a. Jalankan dan perlihatkan hasil program saudara!

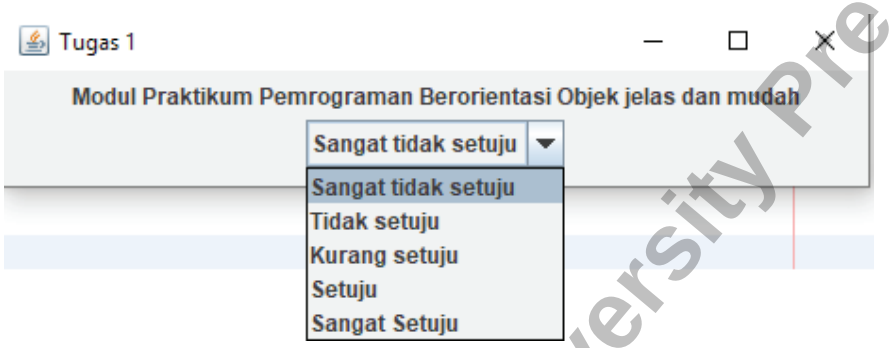
b. Lengkapi pengertian implementasi class `JCheckBox` pada table berikut ini

Konstruktor	Penjelasan
`JCheckBox(String teks)`	...
`JCheckBox(String, Boolean)`	...
`JCheckBox(icon)`	...
`JCheckBox(icon, Boolean)`	...
`JCheckBox(String, Icon)`	...
`JCheckBox(String, Icon, Boolean)`	...

12.6. TUGAS

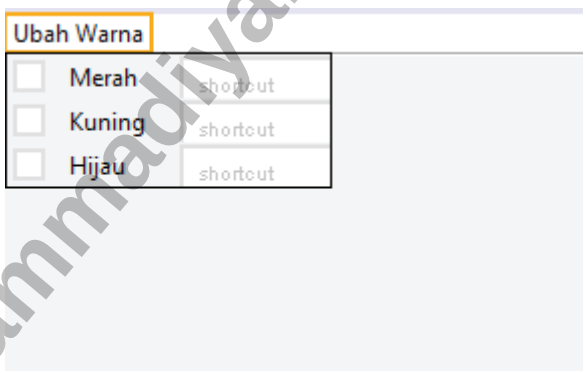
Buatlah program GUI berdasarkan output berikut ini!

1. Gunakan class `JComboBox` untuk membuat program GUI berdasarkan Gambar !



Gambar 11.6 Tampilan GUI menggunakan `JComboBox`

2. Buatlah program GUI untuk menambah menu seperti Gambar 11.7 berikut ini! Apabila salah satu menu dipilih maka akan mengubah warna background.



Gambar 11.7 Tampilan GUI menggunakan `JMenu`

DAFTAR PUSTAKA

Hakim, Rachmad. *Mastering Java, Konsep Pemrograman Java dan Penerapannya untuk Membuat Software Aplikasi*. PT. Elex Media Komputindo, Jakarta. 2009.

Horstmann, Cay S. Core Java Volume I – Fundamental. Prentice Hall, New York. 2016

Informatika. *Modul Praktikum Pemrograman Brorientasi Objek Update 2017*. Informatika, Universitas Muhammadiyah Surakarta. 2017.

Kurniawan, Agus. *Pemrograman Java Tingkat Lanjut*. Penerbit Andi, Yogyakarta. 2014

ISBN:978-602-361-238-3