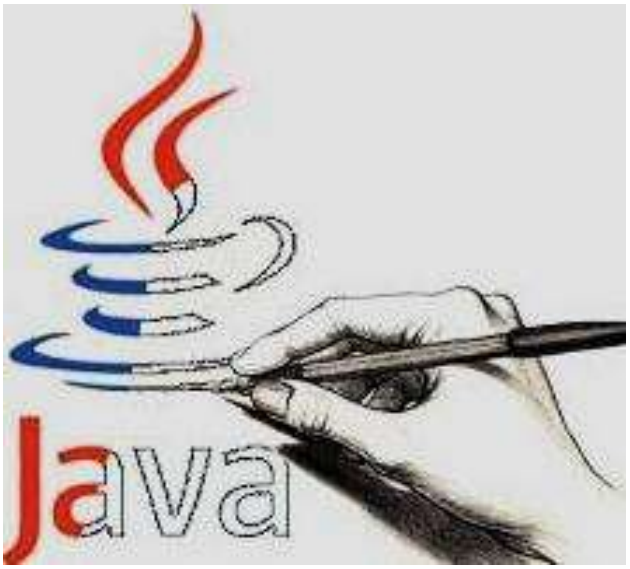
PEMROGRAMAN BERORIENTASI OBJEK DENGAN JAVA



Mingsep Rante Sampebua

PEMROGRAMAN BERORIENTASI OBJEK DENGAN JAVA

Cetakan Pertama, September 2018
76 hlm; 17.6 x 25.0 cm
ISBN: 978-602-5866-12-8

Penulis : Mingsep Rante Sampebua, Dr.

Editor : Anas Arfandi, Dr.
Desain Cover : Anas Arfandi, Dr.
Setting/Layout : Gunadarma Ilmu
Diterbitkan Oleh : Gunadarma Ilmu, Samata - Gowa

Perpustakaan Nasional RI: Data Katalog Dalam Terbitan (KDT)

PEMROGRAMAN BERORIENTASI OBJEK

KATA PENGANTAR

Buku ini merupakan materi kuliah Pemrograman Berorientasi Objek. Program Komputer adalah kumpulan dari instruksi-instruksi yang memandu komputer untuk menjalankan tugas tertentu, dalam kehidupan nyata dapat dicontohkan seperti sebuah resep yang akan digunakan untuk memasak masakan tertentu. Didalam resep terdapat daftar bahan yang dibutuhkan yang kita sebut variabel atau data, dan juga langkah-langkah untuk membuatnya yang memandu komputer untuk mengolah data atau variabel yang telah ada. Jadi pemrograman adalah teknik untuk membuat komputer melakukan apa yang kita inginkan. Karakteristik utama dari sistem berorientasi objek adalah abstraksi, encapsulation, inheritance, polymorphisme, reusable, generalization, dan komunikasi antar objek. Paradigma berorientasi objek adalah cara memandang sistem sebagai satu kesatuan yang terdiri dari berbagai objek. Objek-objek tersebut dikapsulasi dalam bentuk kelas. Akhir kata penulis mengucapkan terima kasih kepada rekan-rekan kerja di UNCEN, istri tercinta, dan anak anak saya Cahaya, Ratih, Ratu dan Nuh yang telah memberi semangat dan dorongan agar selalu berserah dan bersyukur kepada Tuhan Yang Maha Esa dalam mengerjakan pekerjaan. Penulis sampaikan terima kasih kepada Direktorat Riset dan Pengabdian Masyarakat atas hibah Penelitian Strategis Nasional pendanaan tahun 2018.

Jayapura, Agustus 2018

Penulis

DAFTAR ISI

BAB 1
PENDAHULUAN
PEMROGRAMAN BERORIENTASI OBJEK

Konsep Dasar Pendekatan Objek

- Suatu teknik atau cara pendekatan baru dalam melihat permasalahan dari suatu sistem (sistem perangkat lunak, sistem informasi, atau sistem lainnya).
- Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.
- Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut, mulai dari abstraksi objek, kelas, hubungan antar kelas sampai abstraksi sistem.
- Saat mengabstraksikan dan memodelkan objek ini, data dan proses-proses yang dipunyai oleh objek akan dienkapsulasi (dibungkus) menjadi satu kesatuan.

Contoh:
Tinjau aktivitas kuliah pada suatu sistem akademik sebagai berikut:

Dari aktivitas kuliah tersebut, secara eksplisit ada 3 objek yang langsung dapat dikenali yaitu **Dosen** yang memberikan kuliah, **Mahasiswa** yang mengikuti kuliah, dan **Materi Kuliah** yang disampaikan. Secara implisit, ada 2 objek lain yang bisa dikenali lagi yaitu **Jadwal** kapan kuliah diadakan dan **Nilai** yang didapat mahasiswa dari kuliah yang sudah diikutinya. Abstraksi dan pemodelan untuk salah satu dari kelima objek tersebut.
Misalnya untuk objek Dosen diabstraksikan menjadi:

Kelas Dosen
Atribut
Kode_Dosen
Nama_Dosen

Pendidikan
Operasi
Mengajar
Menilai
Memberi_tugas

Objek dan Kelas

Apakah yang disebut objek? Apakah yang disebut kelas? Adalah sangat penting untuk membedakan antara objek dengan kelas.

Objek

- Objek adalah abstraksi dari sesuatu yang mewakili dunia nyata seperti benda, manusia, organisasi, tempat, kejadian, struktur, status atau hal-hal lain yang bersifat abstrak.
- Suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya.
- Dalam konteks OOP, objek adalah instansiasi (yang dibentuk secara seketika) dari kelas pada saat eksekusi (seperti halnya deklarasi variabel pada pemrograman prosedural). Jadi semua objek adalah instan dari kelas.
- Objek mempunyai siklus hidup: diciptakan, dimanipulasi, dan dihancurkan

Kelas

- Kelas adalah kumpulan dari objek-objek dengan karakteristik yang sama.
- Kelas adalah definisi statik dari himpunan objek yang sama yang mungkin lahir atau diciptakan dari kelas tersebut.
- Sebuah kelas akan mempunyai sifat (atribut), kelakuan (operasi), hubungan (relationship) dan arti.
- Suatu kelas dapat diturunkan dari kelas yang lain, dimana atribut dari kelas semula dapat diwariskan ke kelas yang baru.

Kesimpulan:

- Objek adalah model eksekusi, sementara kelas adalah deskripsi statik dari objek yang mungkin lahir pada saat eksekusi.
- Pada saat eksekusi yang kita punyai adalah objek, sementara dalam pemodelan (analisis dan perancangan) dan teks program yang kita lihat adalah kelas.

Property Objek

Sebuah objek pada dasarnya mempunyai property sebagai berikut:

Atribut

- Nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek.
- Merupakan ciri dari sebuah objek.
- Dipunyai secara individual oleh sebuah objek.
- Contoh: berat, warna, jenis, nama, dan sebagainya

Layanan (Service)

- Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri.
- Fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek.
- Dapat berasal dari:
  - model objek
  - event
  - aktivitas atau aksi keadaan
  - fungsi
  - kelakuan dunia nyata
- Contoh: Read, Write, Move, Copy, dan sebagainya

Klasifikasi Objek

Menurut [BOO95] objek dapat dibedakan menjadi:

- ADT (Abstract Data Type). Definisi dari kelas dimana komponen type menjadi atribut dan fungsi primitive menjadi operasi/metode/layanan.
- Mesin. Objek pasif yang punya status yang akan diaktifkan oleh objek lain. Fungsi primitive pada mesin merupakan mekanisme transisi yang mengubah suatu status ke status lain.
- Proses. Objek aktif yang mempunyai "urutan kendali" (*thread of control*).

Sistem Berorientasi Objek

Definisi

- Sebuah sistem yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah sistem yang komponennya dibungkus (dienkapsulasi) menjadi kelompok data dan fungsi.
- Setiap komponen dalam sistem tersebut dapat mewarisi atribut dan sifat dari komponen lainnya, dan dapat berinteraksi satu sama lainnya

Karakteristik Sistem Berorientasi Objek

Karakteristik yang dipunyai sebuah sistem berorientasi objek adalah:

1. **Abstraksi**
   Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan. Abstraksi dalam konteks objek adalah suatu cara melihat suatu objek dalam bentuk yang lebih sederhana. Contoh mobil sebagai kumpulan ratusan komponen elektronik, mekanik dengan 4 ban atau lebih, kursi, stir kemudi, dan lain sebagainya.
2. **Enkapsulasi**
   Pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek, untuk menyembunyikan implementasi dari objek sehingga objek lain tidak mengetahui cara kerjanya.

3. **Pewarisan (Inheritance)**
   Mekanisme yang memungkinkan satu objek (baca: kelas) mewarisi sebagian atau seluruh definisi dari objek lain sebagai bagian dari dirinya.
4. **Reusability**
   Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada
   permasalahan lainnya yang melibatkan objek tersebut.
5. **Generalisasi dan Spesialisasi**
   Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus.
6. **Komunikasi Antar Objek**
   Komunikasi antar objek dilakukan lewat pesan (*message*) yang dikirim dari satu objek ke objek lainnya.
7. **Polymorphism**
   Kemampuan suatu objek untuk digunakan di banyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

BAB 2
DASAR-DASAR PEMROGRAMAN BERORIENTASI OBJEK DENGAN JAVA

2.1 Dasar Program Java

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

Pada kode diatas, kita telah membuat sebuah program sederhana yang menampilkan tulisan "Hello World" pada console. Terdapat beberapa aturan dalam membuat program dalam Java yaitu :

1. Nama file harus sama dengan nama kelas program. Misal pada kode diatas nama kelasnya adalah ProgPertama, maka nama file harus ProgPertama.java
2. Hanya boleh terdapat satu kelas public pada sebuah file.
3. Kelas yang menjadi program yang akan dijalankan memiliki metode public static void main(String[] args)

2.2 Tipe Data

Terdapat beberapa tipe data primitive yang ada di Java yaitu :

Tipe Data	Keterangan
boolean	true atau false
char	Karakter
byte	-128 – 127
short	-32768 – 32768
int	-2147483648 – 2147483647
long	-9223372036854775808 – 9223372036854775807
double	4.9E-324 – 1.7976931348623157E308
float	1.4E-45 – 3.4028235E38

String bukanlah merupakan tipe data di Java, String merupakan Object. Namutn string memiliki keunikan yaitu String dapat langsung dibuat tanpa harus membuat Object.

2.3 Variabel

Variabel merupakan sesuatu yang digunakan untuk menampung sebuah data. Sebuah variabel harus ada dalam sebuah kelas atau metode. Pembuatan sebuah variabel di Java terlihat pada kode dibawah ini:

Tipevariabel namavariabel;

Tipe variabel dapat berupa tipe data atau kelas, misal :

int nilai;
char indexNilai;
Untuk menambahkan nilai ke sebuah variabel, maka dapat menggunakan tanda = (sama dengan) , misal jika kita akan menambahkan nilai 100 pada variabel nilai dan A pada variabel indexNilai, maka dapat terlihat pada kode dibawah ini.

```
int nilai;
char indexNilai;
nilai = 100;
indexNilai = 'A';
```

Atau dapat juga langsung saat pembuatan sebuah variabel.

```
int nilai = 100;
char indexNilai = ‘A’;
```

Syarat-syarat penamaan variabel adalah :

1. Harus diawalai dengan huruf
2. Tidak boleh terdapat karakter unik seperti @, #,% dan lain-lain
3. Tidak boleh mengandung karakter putih (spasi, enter, tab)

2.4 Operator

Operator merupakan sebuah karakter khusus yang digunakan untuk menghasilkan suatu nilai.

2.4.1 Operator Aritmatika

Operator	Keterangan
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
%	Sisa pembagian

Contoh :

```
int a = 10;
int b = 3;
int c = a / b;
System.out.println(c);
```

Hasil dari kode program diatas adalah 3 bukan 3.333. Hal ini dikarenakan dalam Java jika kita melakukan operasi pembagian dengan tipe data integer, maka hasilnyapun akan integer, dan integer tidak dapat mengandung nilai koma dalam Java, sehingga jika akan melakukan perkalian yang menghasilkan nilai koma, maka harus menggunakan tipe data double atau float.

2.4.2 Operator Penugasan

Operator	Keterangan
=	Pemberian nilai
+=	Penambahan bilangan
- =	Pengurangan bilangan
*=	Perkalian bilangan
/=	Pembagian bilangan
%=	Perolehan sisa bagi

Contoh

```
int a = 10;
a += 5;
System.out.println(a);
```

Hasil dari operasi += tersebut adalah 15. Hal ini dikarenakan a += 5 sama dengan
a = a + 5, dikarenakan a sebelumnya adalah 10, maka itu berarti a = 10 + 5.
Jika akan melakukan penambahan atau pengurangan dengan nilai 1, maka dapat
dengan mudah menggunakan karakter ++ untuk penambahan atau -- untuk pengurangan, misal :

```
int a = 10;
a--;
System.out.println(a);
```

Maka hasilnya adalah 9.

2.4.3 Operator Pembanding

Operator	Keterangan
==	Sama dengan
!=	Tidak sama dengan
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama dengan
>	Lebih besar
<	Lebih kecil

Hasil dari operasi pembanding adalah boolean. True jika operasi pembanding tersebut benar, dan false jika operasi pembanding tersebut salah, misal :

```
boolean a = 10 == 100;
System.out.println(a);
```

Hasil dari program diatas adalah false, karena memang 10 tidak sama dengan 100.

2.4.4 Operator Logika

Operator logika digunakan untuk membentuk suatu keadaan dari dua atau lebih kondisi tertentu, operator logika biasanya digabungkan dengan operator pembanding.

Operator	Keterangan
&&	Dan
||	Atau

Hasil dari operator logika adalah boolean. Hasil operasi logika dengan menggunakan && adalah sebagai berikut.

Operasi1	Operasi 2	Hasil
False	False	False
False	True	False
True	False	False
True	True	True

Hasil operasi logika dengan menggunakan || adalah sebagai berikut.

Operasi1	Operasi 2	Hasil
False	False	False
False	True	True
True	False	True
True	True	True

Contoh.

```
boolean hasil = 10 == 100 || 100 == 100;
System.out.println(hasil);
```

Maka hasilnya adalah true.

2.5 Percabangan (Kontrol Alur Eksekusi Program)

Secara garis besar, kontrol alur eksekusi dapat dikelompokkan kedalam 3 tiga bagian besar yaitu: seleksi, iterasi/perulangan, dan jump.

2.5.1. Seleksi

Terdapat dua kata kunci yang dapat digunakan untuk melakukan seleksi dalam java, yaitu if dan switch. Keduanya menentukan alur eksekusi program mana yang akan dijalankan pada suatu percabangan.

a. If

Percabangan if-else merupakan percabangan yang sama dengan percabangan if namun memiliki kondisi false, artinya jika kondisi pada if tidak terpenuhi maka perintah pada else akan dijalankan. Secara umum, penggunaan if mengikuti ketentuan sbb:

```
if(kondisi)
  pernyataan;
```

```
  atau menggunakan if dan else
if(kondisi)
  pernyataan_1;
else
  pernyataan_2;
```

kondisi haruslah berupa ekspresi yang menghasilkan nilai Boolean (true atau false), sedangkan pernyataan_1 dan pernyataan_2 merupakan code atau blok code yang akan dieksekusi sesuai dengan nilai yang dihasilkan oleh kondisi. Jika kondisi bernilai true maka pernyataan_1 yang akan dieksekusi. Jika kondisi bernilai false maka pernyataan_2 yang akan dieksekusi.

Contoh program:

```
package bab2;
class TesSeleksi {
    public static void main(String[] args) {
        int a, b;
        a = 8;
        b = a / 2;
        if (b > 5) {
            System.out.println("Benar atau Memenuhi");
        } else {
            System.out.println("Tidak Memenuhi");
        }
    }
}
```

Penggunaan if …… else bertingkat (bersarang).

Percabangan if bersarang merupakan gabungan beberapa if dan dapat pula digabung dengan if-else. Bentuk pernyataan if bersarang adalah sebaga berikut :

```
if(kondisi1){
  // perintah kondisi1
  }else if(kondisi2){
  // perintah kondisi2
  }else if(kondisi3){
  // perintah kondisi3
 }else{
// perintah jika semua kondisi tidak ada yang benar
 }
```

Contoh program:

```
package bab2;
class If_else_bersarang {
    public static void main(String[] args) {
        int nilai = 6;
        char index;
```

```
        if (nilai >= 8) {
            index = 'A';
        } else if (nilai >= 7) {
            index = 'B';
        } else if (nilai >= 6) {
            index = 'C';
        } else if (nilai >= 5) {
            index = 'D';
        } else {
            index = 'E';
        }
        System.out.println(index);
    }
}
```

Program java yang berfungsi untuk membaca data yang diinput pengguna pada keyboard

```
package bab2;
import java.io.*;
public class InputConsole {

    /**Membaca string dari keyboard*/
    public static String readString() {
        BufferedReader bfr = new BufferedReader(new
InputStreamReader(System.in), 1);
        // Menginisialisasi string
        String string = " ";
// Mengambil string dari keyboard
        try {
            string = bfr.readLine();
        } catch (IOException ex) {
            System.out.println(ex);
        }

// Mengembalikan string hasil pembacaan dari keyboard
        return string;
    }
/**Mengambil nilai int dengan parsing string input dari
Keyboard */
    public static int readInt() {
        return Integer.parseInt(readString());
    }
//Mengambil nilai byte dengan parsing string input dari
keyboard
    public static byte readByte() {
```

```
        return Byte.parseByte(readString());
    }
//Mengambil nilai short dengan parsing string input dari
keyboard
    public static short readShort() {
        return Short.parseShort(readString());
    }
 //Mengambil nilai long dengan parsing string input dari
keyboard
    public static long readLong() {
        return Long.parseLong(readString());
    }
//Mengambil nilai float dengan parsing string input dari
keyboard
    public static float readFloat() {
        return Float.parseFloat(readString());
    }
//Mengambil nilai double dgn parsing string input dari
keyboard
    public static double readDouble() {
        return Double.parseDouble(readString());
    }
}
```

Contoh:

```
package bab2;
public class Demo_if {
/**Main Method*/
    public static void main(String[] args) {
        double nilaiInput;
        System.out.println("\nPROGRAM DEMO IF");
        System.out.println("---------------\n");
        //  Menginstruksikan  user  untuk  menginputkan
nilai
        System.out.print("Masukkan Nilai [0 - 100]: ");
        nilaiInput = InputConsole.readDouble();

        // Menyeleksi kondisi dan mencetak hasil ke
console
        if (nilaiInput >= 60) {
    System.out.println("\nSelamat,   anda   lulus   mata
kuliah ini.");
        }
    }
}
```

Contoh:

```
package bab2;
class TestIfGrade {
    public static void main(String[] args) {
        double nilaiInput;
        //Menginstruksikan user untuk menginputkan nilai
        System.out.print("Masukkan Nilai [0 - 100]: ");
        nilaiInput = InputConsole.readDouble();

        //Menyeleksi range 0 -100
        if ((nilaiInput < 0) | (nilaiInput > 100)) {
            System.out.println("Data input invalid !");
        } //Kalau input valid, lakukan instruksi berikut
        else {
            //Menyeleksi kondisi
            if (nilaiInput >= 60) {
      System.out.println("Selamat, anda lulus mata
kuliah ini.");
                if (nilaiInput >= 85) {
                    System.out.println("Nilai anda :
A");
                } else if (nilaiInput >= 70) {
                    System.out.println("Nilai anda :
B");
                } else if (nilaiInput >= 60) {
                    System.out.println("Nilai anda :
C");
                }
            } else {
       System.out.println("Maaf anda harus mengulang
kembali.");
            }
        }
    }
}
```

Contoh Prak_ifelse_nilai

```
package bab2;
import java.util.Scanner;
public class Prak_ifelse_nilai {
    public static void main(String[] args) {
        Scanner masukan = new Scanner(System.in);
        System.out.print("Masukan nilai matakuliahnya =
");
        int nilai = masukan.nextInt();
        if (nilai >= 80) {
```

```
            System.out.println("NIlainya A");
        } else if (nilai >= 70) {
            System.out.println("Nilainya B");
        } else if (nilai >= 55) {
            System.out.println("Nilainya C");
        } else if (nilai >= 40) {
            System.out.println("Nilainya D");
        } else {
            System.out.println("Nilainya E");
        }
    }
}
```

b. Switch

Percabangan switch-case merupakan percabangan yang kondisinya hanya dapat menggunakan perbandingan = (sama dengan). Bentuk pernyataan percabangan switch case adalah sebagai berikut :

```
switch(ekspresi){
    case nilai1:
    // jalankan instruksi
    break; // hentikan
    case nilai2:
    // jalankan instruksi
    break; // hentikan
    case nilai2:
    // jalankan instruksi
    break; // hentikan
    case nilai4:
    // jalankan instruksi
    break; // hentikan
    default:
    // jalankan instruksi
    break; // hentikan
 }
```

Penggunaan break adalah untuk memberitahu bahwa pernyataan didalam blok switch telah selesai. Penggunaan default adalah mirip dengan penggunaan else dalam if artinya jika tidak ada nilai dalam case yang cocok dengan nilai ekspresi, maka pernyataan default yang akan dieksekusi. Pada percabangan switch pertama, diperlukan sebuah variable (ekspresi), setelah itu pada bagian case dibandingkan, jika sama, maka instruksi akan dijalankan sampai menemui tanda break.

Contoh:

```
package bab2;
class Hitung{
```

```
    public static void main(String[] args){
      int angka = 4;
      switch(angka%2){
      case 0:
         System.out.println(“BilanganGenap”);
         break;
       case 1:
          System.out.println(“Bilangan Ganjil”);
          break;
       }
     }
}
```

Contoh:

```
package bab2;
class DemoSwitch {
    public static void main(String[] args) {
        int hari = 5;
        switch (hari) {
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
                System.out.println("Bukan Hari Libur");
                break;
            case 7:
                System.out.println("Hari Libur");
                break;
            default:
                System.out.println("Hari            Tidak
Diketahui");
                break;
        }
    }
}
```

Contoh:

```
package bab2;
public class DemoSwitchCase {
    /**Main Method*/
    public static void main(String[] args) {
        int pilihan;
        System.out.println("\nPROGRAM DEMO SWITCH
CASE");
```

```
        System.out.println("----------------------\n");

        // Menginstruksikan user untuk memasukkan
pilihan
     System.out.print("Masukkan pilihan anda [1-3] : ");
        pilihan = InputConsole.readInt();
        switch (pilihan) {
            case 1:
       System.out.println("\nAnda memilih Tom Cruise.");
                break;
            case 2:
       System.out.println("\nAnda memilih Jacky Chan.");
                break;
            case 3:
        System.out.println("\nAnda memilih Van Damme.");
                break;
            default:
      System.out.println("\nHanya pilihan [1-3] yang " +
                         "diperbolehkan.");
        } // akhir switch
    } // akhir main method
}
```

2.5.2. Perulangan (Iterasi)

a. Perulangan While

Pernyataan while berguna untuk melakukan proses perulangan untuk sebuah kondisi, selama kondisi tersebut bernilai benar (true), maka perulangan akan terus berjalan, dan terhenti ketika kondisi bernilai salah (false). Bentuk pernyataan while seperti berikut ini:

```
while(kondisi){
// isi instruksi
}
```

Contoh:

```
package bab2;
public class DemoWhile {
    /**Main Method*/
    public static void main(String[] args) {
        int jumlah = 1;
        while (jumlah <= 10) {
            System.out.println(jumlah);
            jumlah++; // menaikkan jumlah
        }
    }
}
```

Jika program tersebut dijalankan, maka hasilnya adalah tulisan dari no 1 sampai 10. Dan saat jumlah bernilai 11, maka perulangan akan terhenti dikarenakan kondisi bernilai false (11 ≤ 10).

b. Perulangan do-while

Perulangan do-while merupakan perulangan yang hampir mirip dengan perulangan while namun perbedaannya, pada perulangan do-while, maka minimal instruksi akan dijalankan sekali. Bentuk pernyataan do-while sebagai berikut :

```
do{
// insturksi
}while(kondisi);
```

Contoh:

```
package bab2;
public class DemoDoWhile {

    /**Main Method*/
    public static void main(String[] args) {

        int jumlah = 100;
        do {
            System.out.println(jumlah);
            jumlah++; // naikkan jumlah
        } while (jumlah <= 10);
    }
}
```

Jika program tersebut dijalankan, maka akan menghasilkan keluaran 100, artinya walaupun kondisi salah, namun minimal isi instruksi akan dijalankan sekali, hal ini dikarenakan proses do-while berbeda dengan while, dimana do-while pertama melakukan instruksi baru mengecek kondisi, sedangkan while pertama mengecek kondisi baru melakukan instruksi.

Contoh:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class loop_do_while {
    public static void main(String[] args) {
        int Data;
        do {
            System.out.print("Masukkan angka Anda : ");
            BufferedReader bfr = new BufferedReader(new
InputStreamReader(System.in));
            String angkaInput = null;
            try {
```

```
                angkaInput = bfr.readLine();
            } catch (IOException e) {
                e.printStackTrace();
            }
            Data                                                =
Integer.valueOf(angkaInput).intValue();
        } while (Data != 99);
        System.out.println("Anda keluar dari LOOP");
    }
}
```

c. Perulangan for

Perulangan for merupakan perulangan yang memiliki variabel untuk melakukan pengkondisian, berbeda dengan while dan do-while yang kita harus membuat sebuah variabel diluar untuk melakukan penkondisian, pada perulangan for, ditempatkan sebuah blok untuk membuat variabel dan melakukan proses pengkondisian. Bentuk pernyataan for seperti berikut :

```
for(inisialisasi; kondisi; penaikan/penurunan){
instruksi
}
```

Misal kita akan menampilkan angka dari 1 = 100, maka dapat menggunakan perulangan for.

```
public class Contoh_for {
  public static void main(String[] args) {
   for(int i = 1; i <= 100; i++){
   System.out.println(i);
   }
  }
}
```

Contoh:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class loop_FOR {
     public static void main(String[] args) {
          System.out.println("Masukkan angka Anda : ");
         BufferedReader bfr = new BufferedReader
         (new InputStreamReader(System.in));
         String angkaInput = null;
          try {
               angkaInput = bfr.readLine();
          } catch (IOException e) {
```

```
                e.printStackTrace();
            }
          int Data =
Integer.valueOf(angkaInput).intValue();
          for(int ulang = 1; ulang<=Data; ulang++) {
            System.out.println("Bilangan : " + ulang);
          }
      }
}
```

Contoh Nested Loop For:

```
package bab2;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class NestedLoopFor {

    public static void main(String[] args) {
        int ulang = inputData();
        buatSegitiga(ulang);
    }

    private static int inputData() {
    System.out.print("Masukkan jumlah segitiga yang Anda
mau: ");
        BufferedReader bfr = new BufferedReader(new
InputStreamReader(System.in));
        String angkaInput = null;
        try {
            angkaInput = bfr.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }
        int Data =
Integer.valueOf(angkaInput).intValue();
        return Data;
    }

    private static void buatSegitiga(int ulang) {
        for (int i = 1; i <= ulang; i++) {
            for (int j = 1; j <= (ulang - i); j++) {
                System.out.print(" ");
            }
            for (int k = 1; k <= i; k++) {
                System.out.print("#");
```

```
            }
            System.out.println("");
        }
    }
}
```

Contoh Nested Loop For:

```
package bab2;
public class NestedLoopFor_2 {
    public static void main(String[] args) {
        for (int x = 0; x < 5; x++) {
            for (int y = 0; y < 5; y++) {
                System.out.println("x= " + x + " y= " +
y + " ");
            }
            System.out.println();
        }
    }
}
```

2.5.3. Jump

a. Break

Perintah break merupakan perintah yang dapat digunakan untuk menghentikan proses
perulangan, misal jika kita membuat program seperti berikut:

```
package bab2;
public class Break {

    public static void main(String[] args) {
        for (int i = 1; i <= 100; i++) {
            System.out.println(i);
            if (i == 50) {
                break;
            }
        }
    }
}
```

Maka program hanya akan menampilkan angka dari 1 sampai 50, karena pada saat i mencapai 50, program dihentikan oleh perintah break.

Contoh:

```
package bab2;
public class Break_2 {
    public static void main(String[] args) {
        for (int nilai = 0; nilai < 20; nilai += 2) {
            if (nilai == 10) {
                break;
```

```
            }
            System.out.print(nilai + " ");
        }
        System.out.println();
    }
}
```

Contoh:

```
public class BreakLabel{
public static void main(String[] args){
      test:
      for(int i = 1; i < 100; i++){
            int j = 1;
      System.out.println("Loop tingkat pertama i= " + i);
            while(true){
      System.out.println("Loop tingkat kedua j=" + j);
                  if(j++ == 4){
                  break test;
                  }
            }
      }
      System.out.println("Keluar dari loop");
   }
}
```

b. Continue

Perintah continue dapat digunakan untuk meloncati sebuah perulangan, maksudnya adalah instruksi yang seharusnya dapat dilewat, hal ini berarti instrukti tidak akan dijalankan. Contoh:

```
package bab2;
public class DemoContinue {
    public static void main(String[] args) {
        for (int i = 1; i <= 60; i++) {
            if (i % 2 == 0) {
            //if (i % 2 == 1) {
                continue;
            }
            System.out.println(i);
        }
    }
}
```

Jika program diatas dijalankan, maka hasilnya akan menampilkan angka-angka ganjil saja, hal ini dikarenakan saat nilai i merupakan angka genap, maka perintah continue membuat program tidak menampilkan angka genap.

Contoh:

```
public class DemoContinue{
public static void main(String[] args){
      for (int nilai = 0; nilai <20; nilai += 2){
      if (nilai == 10)
                  continue;
            System.out.print(nilai + " ");
       }
       System.out.println();
  }
}
```

Contoh:

```
public class DemoContinueLabel{
public static void main(String[] args){
      wow:
      for(int i = 0; i < 3; i++){
      System.out.println("Loop tingkat pertama i=" + i);
      for(int j = 0; j < 10; j++){
            System.out.println("Loop tingkat kedua j=" +
j);
            if(j == (i * 2)){
            continue wow;
            }
        }
      }
      System.out.println("Keluar dari loop");
  }
}
```

2.6 Array

Array merupakan objek yang dapat digunakan untuk menyimpan sejumlah data. Data yang dapat ditampung pada array dapat berupa tipe data ataupun kelas (objek).

2.6.1. Mendeklarasikan Array

Untuk membuatan variabel array pun berbeda dengan membuat variable biasanya yaitu sebagai berikut:

```
TipeArray namaArray[];
```

Dimana tipe array dapat berupa tipe data biasa seperti int, char, short atau juga kelas seperti String dan yang lainnya.

2.6.2. Membuat Array

Setelah mendeklarasikan Array, maka perlu dibuat arraynya terlebih dahulu, sebelum array tersebut digunakan, caranya dengan menggunakan perintah new.

```
TipeArray namaArray[];
```

```
namaArray = new TipeArray[jumlah];
```

Dimana jumlah array merupakan jumlah data yang dapat ditampung oleh array.

2.6.3. Memanipulasi Data dalam Array

Setelah membuat Array, maka untuk melakukan proses manipulasi seperti menambahkan data ke Array, mengubah data di Array ataupun mengakses data dalam Array, maka diperlukan sebuah indeks, dimana saat membuat sebuah array dengan jumlah data 5, maka hasilnya akan terlihat seperti ini.

No	Index
1	0
2	1
3	2
4	3
5	4

Artinya data ke 1 dapat diakses menggunakan indeks 0 dan seterusnya. Dan untuk melakukan proses manipulasi data dalam array menggunakan indeks dapat digunakan dengan perintah :

```
namaArray[indeks];
```

Contoh:

```
int a[] = new int[5];
a[0] = 234;
a[1] = 6867;
a[2] = 4234;
a[3] = 564;
a[4] = 2423;
System.out.println(a[0]);
System.out.println(a[1]);
System.out.println(a[2]);
System.out.println(a[3]);
System.out.println(a[4]);
```

2.6.4. Array Multidimensi

Java mendukung pembuatan array multidimensi maksudnya kita dapat menambahkan data array ke dalam sebuah array, cara pembuatannya adalah sebagai berikut :

```
TipeArray namaArray[][] = new TipeArray[jumlah][jumlah];
```

Contoh :

```
package bab1;
public class Matriks {
    /**Main method*/
    public static void main(String[] args) {
   // Membuat dua matriks berdimensi dua (m x n) di mana
m = n =3
        System.out.print("Masukkan orde matriks : ");
        int ordeMatriks = InputConsole.readInt();
        int[][]          matriks1          =          new
int[ordeMatriks][ordeMatriks];
        int[][]          matriks2          =          new
int[ordeMatriks][ordeMatriks];
        //   Menugaskan   nilai   dengan   mendeskripsikan
secara langsung
    /************************************************
        int[][] matriks1 = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
        };

        int[][] matriks2 = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
        };
     ************************************************/

  //Menginstruksikan user untuk menginputkan tiap elemen

        // Memasukkan entry matriks 1
        for (int i = 0; i < matriks1.length; i++) {
            for (int j = 0; j < matriks1[i].length; j++)
              {
                int indeksBaris = i + 1;
                int indeksKolom = j + 1;
     System.out.print("Masukkan elemen Matriks A pada
                      baris ke " +
       indeksBaris + " kolom ke " + indeksKolom + "\t");
                matriks1[i][j] = InputConsole.readInt();
            }
        }

        System.out.println("-----------------------");
```

```
        // Memasukkan entry matriks 2
        for (int i = 0; i < matriks1.length; i++) {
            for (int j = 0; j < matriks1[i].length; j++)
{
                int indeksBaris = i + 1;
                int indeksKolom = j + 1;
System.out.print("Masukkan elemen Matriks B pada baris
ke " + indeksBaris + " kolom ke " + indeksKolom + "\t");
//"\t"=tab
                matriks2[i][j] = InputConsole.readInt();
            }
        }

 // Menambahkan dua matriks & mencetak hasilnya di
monitor
        int[][] matriksHasil = jumlahMatriks(matriks1,
matriks2);
        System.out.println("Penjumlahan Matriks ");
        cetakHasil(matriks1,   matriks2,   matriksHasil,
'+');

// Mengalikan dua matriks dan mencetak hasilnya di
monitor
        matriksHasil = kaliMatriks(matriks1, matriks2);
        System.out.println("\nPerkalian Matriks ");
     cetakHasil(matriks1, matriks2, matriksHasil, '*');
    }

    /**Method penjumlahan dua matriks*/
    public    static    int[][]    jumlahMatriks(int[][]
matriks1, int[][] matriks2) {
        int[][]          hasil          =          new
int[matriks1.length][matriks1[0].length];
        for (int i = 0; i < hasil.length; i++) {
            for (int j = 0; j < hasil[0].length; j++) {
                hasil[i][j]    =    matriks1[i][j]    +
matriks2[i][j];
            }
        }

        return hasil;
    }
    /**Method perkalian dua matriks*/
    public static int[][] kaliMatriks(int[][] matriks1,
int[][] matriks2) {
        int[][]          hasil          =          new
int[matriks1.length][matriks2[0].length];
```

```
        for (int i = 0; i < hasil.length; i++) {
            for (int j = 0; j < hasil[0].length; j++) {
          for (int k = 0; k < matriks1[0].length; k++) {
                    hasil[i][j]   +=   matriks1[i][k]   *
matriks2[k][j];
                }
            }
        }

        return hasil;
    }
    /**Method untuk mencetak hasil di monitor*/
    public static void cetakHasil(
            int[][] matriks1, int[][] matriks2, int[][]
matriks3, char op) {
        for (int i = 0; i < matriks1.length; i++) {
          for (int j = 0; j < matriks1[0].length; j++) {
                System.out.print(" " + matriks1[i][j]);
            }
            if (i == matriks1.length / 2) {
                System.out.print("  " + op + "  ");
            } else {
                System.out.print("     ");
            }
          for (int j = 0; j < matriks2[0].length; j++) {
                System.out.print(" " + matriks2[i][j]);
            }
            if (i == matriks1.length / 2) {
                System.out.print("  =  ");
            } else {
                System.out.print("     ");
            }
          for (int j = 0; j < matriks3[0].length; j++) {
                System.out.print(" " + matriks3[i][j]);
            }
            System.out.println();
        }
    }
}
```

BAB 3
PEMROGRAMAN BERORIENTASI OBJEK

Pemrograman Berorientasi Objek (PBO) merupakan pemrograman yang menjadikan objek sebagai komponen utama dalam sistem. Objek merupakan gabungan data dan fungsi, dimana sebuah objek dibuat dari

sebuah kelas. Untuk dapat menguasai pemrograman Java, harus mengerti dengan baik konsep pemrograman berorientasi objek, karena Java merupakan bahasa pemrograman berorientasi objek. Pada bagian ini akan dibahas konsep-konsep penting dalam pemrograman berorientasi objek, sehingga diharapkan akan lebih mudah dalam mempelajari bahasa Java.

3.1 Object

Pada dasarnya semua benda yang ada di dunia nyata dapat dianggap sebagai sebuah objek. Jika diperhatikan lebih lanjut, pada dasarnya ada dua karakteristik yang utama pada sebuah objek , yaitu :

- Setiap objek memiliki atribut sebagai status yang kemudian akan disebut sebagai ***state.***
- Setiap objek memiliki tingkah laku yang kemudian akan disebut sebagai ***behaviour.*** Contoh sederhananya adalah : objek sepeda
- Sepeda memiliki atribut (***state***) : pedal, roda, jeruji, dan warna.
- Sepeda memiliki tingkah laku (***behaviour***) : kecepatannya menaik, kecepatannya menurun, dan perpindahan gigi sepeda.

Dalam pengembangan perangkat lunak berorientasi objek, objek dalam perangkat lunak akan menyimpan ***state***-nya dalam variable dan menyimpan informasi tingkah laku (***behaviour***) dalam ***methodmethod*** atau fungsi-fungsi/prosedur.

Objek merupakan hasil dari sebuah kelas, jika diibaratkan Objek adalah kue, maka kelas adalah cetakan kuenya, dimana kue dibuat menggunakan cetakan tersebut. Dan sebuah cetakan kue dapat membuat beberapa kue, artinya sebuah kelas dapat membuat beberapa object. Untuk membuat objek dalam Java diperlukan sebuah perintah new, dimana cara pembuatannya sama dengan pembuatan variabel.

```
Kelas objek = new Kelas();
```

Jika dalam kelas yang dibuat objek tersebut terdapat atribut, maka dapat dipanggil menggunakan . (titik)

```
// mengubah atribut
objek.namaAtribut = value;
```

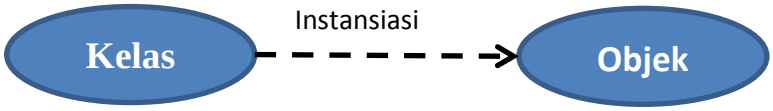
Jika dalam kelas tersebut memiliki sebuah fungsi (metode), maka dapat dipanggil menggunakan . (titik) dan diakhiri dengan ()

```
// memanggil metode
objek.namaMetode();
```

3.2 Class

Class berbeda dengan objek. Class merupakan prototype yang mendefinisikan variabel-variabel dan method-method secara umum. Sedangkan objek merupakan instansiasi dari suatu kelas. Dalam paradigma pemrograman berorientasi objek dikenal kelas dan objek. Kelas merupakan *blue print* dari objek-objek yang akan dibuat. Analogi kelas dan objek seperti

rancangan model rumah dan *pembangunan* rumah-rumah, adapun proses pembuatan objek dari kelas dikenal dengan *instantiasi.*



Dalam bahasa pemrograman Java, kelas didefinisikan menggunakan kata kunci class. Contoh kelas sederhana adalah sebagai berikut :

```
class Manusia {
String nama;
}
```

Sebagai contoh kita ambil kelas manusia. Kelas manusia mempunyai atribut : nama. Selain itu kelas manusia juga mempunyai metodh: tampilkanNama, kerja, makan. Kasus diatas diimplementasikan dalam bahasa Java sebagai berikut:

```
/*Disimpan dalam file "manusia.java"*/
public class Manusia {
   public String nama;

   public Manusia(String n){
   this.nama = n;
   }
   public String tampilkanNama(){
   return nama;
   }
   public void makan(){
   System.out.println("Nyam...              nyam...
nyam...");
   }
   public void kerja(){
   System.out.println("Kerja...kerjaaa...");
   }
}
```

Adapun kode untuk menginstantiasi kelas manusia menjadi objek Andi yang mengimplementasikan metodh: tampilkanNama dan makan adalah sebagai berikut:

```
/*
Disimpan dalam file "Implementasi_Manusia_1.java"
*/
package bab3_2;
public class Implementasi_Manusia_1 {
```

```
  public static void main(String arg[]){
    Manusia_1 andi= new Manusia_1("Andi");
    System.out.println("Nama=                    "+
  andi.tampilkanNama());
    andi.makan();
    }
  }
```

Hasil eksekusi terhadap *class* andi adalah sebagai berikut:

```
Nama= Andi
Nyam... nyam... nyam...
```

- Penjelasan program:
  /* Disimpan dalam file "andi.java"*/
  Atau
  // Disimpan dalam file "andi.java"
  Bagian di atas adalah komentar
- `class Implementasi_Manusia_1` → deklarasi sebuah kelas dengan nama kelasnya adalah andi
  `public static void main(String arg[]){……}` → mendeklarasikan suatu metode dengan nama main sebagai titik awal eksekusi program
- `Manusia andi= new Manusia("Andi");` → kelas manusia di instansiasi menjadi objek andi.
- `System.out.println("Nama=  "+ andi.tampilkanNama());` → untuk menampilkan data / informasi di layar

Tanda ; menunjukkan akhir suatu stateme/pernyataan/perintah

3.2.1 Metode

Dalam java terdapat dua buah metode

1. Fungsi merupakan metode yang memiliki nilai balik jika metode tersebut dipanggil, cara pembuatan sebuah metode adalah dengan cara menentukan nilai baliknya, lalu membuat nama metodenya.
2. Prosedur, merupakan metode yang tidak memiliki nilai balik, cara pembuatan prosedur sama dengan metode namun bedanya, nilai baliknya menggunakan kata kunci void.

Contoh :

```
package bab3;
class Manusia {
    String nama;
// metode
    String getNama() {
```

```
    // untuk mengembalikan nilai gunakan kata kunci
    return
            return nama;
        }
    // prosedur
        void alamat() {
            String alamatnya = "UNCEN Jayapura";
            System.out.println("Alamat=     "     +
    alamatnya);
        }
    }
```

Pada kode diatas, kelas manusia memiliki 2 buah metode yaitu getNama() dan alamat(). Dimana getNama() merupakan sebuah metode karena mengembalikan nilai String, sedangkan alamat() merupakan prosedur karena tidak mengembalikan nilai. Saat membuat sebuah metode maka untuk mengembalikan nilainya, harus menggunakan kata kunci return, diikuti nilai yang akan dikembalikannya. Untuk mengambil nilai balik dari fungsi dapat dilihat pada contoh sebagai berikut.

```
package bab3;
public class AplikasiManusia {

    public static void main(String arg[]) {
        Manusia manusia = new Manusia();
        manusia.nama = "Eko Kurniawan Khannedy";
// mengambil nilai dari metode
       System.out.println("Nama dia siapa ya= " +
                           manusia.getNama());

        manusia.alamat();
    }
}
```

3.2.2 Parameter

Parameter merupakan data yang dapat ditambahkan dari luar metode, misal jika kita membuat sebuah metode untuk mengubah nama pada kelas Manusia, maka pasti kita memerlukan nama baru untuk menggantikan nama lama, oleh karena itu diperlukan sebuah parameter nama baru untuk menggantikan nama tersebut. Contoh parameter dapat terlihat pada kelas dibawah ini. Sebuah metode dapat memiliki satu atau lebih parameter, untuk menambahkan parameter, dipisahkan dengan menggunakan tanda , (koma).

Contohnya pada metode `void ubahData(String namaBaru, int umurnya){..}`

```
    package bab3;
    class Manusia_2 {
        String nama;
        int umur;
        // metode dengan 1 parameter
        void ubahNama(String namaBaru) {
            nama = namaBaru;
        }
        // metode dengan 2 parameter
        void ubahData(String namaBaru, int umurnya) {
            nama = namaBaru;
            umur = umurnya;
        }
    // metode
        String getNama() {
    //  untuk mengembalikan nilai gunakan kata kunci
    return
            return nama;
        }
        int getUmur() {
            return umur;
        }
    // prosedur
        void alamat() {
            String alamatnya = "UNCEN Jayapura";
            System.out.println("Alamat= " + alamatnya);
        }
    }
```

Contoh penggunaannya adalah sebagai berikut :

```
    package bab3;
    public class AplikasiManusia_2 {
        public static void main(String arg[]) {
            Manusia_2 manusia = new Manusia_2();
            manusia.ubahNama("Ahmad");
System.out.println("Nama    dia    siapa    ya:    "    +
manusia.getNama());
            manusia.ubahData("Ahmad Jaya", 25);
    // mengambil nilai dari metode
System.out.println("Nama    dia    siapa    ya:    "    +
manusia.getNama());
System.out.println("Nama    dia    siapa    ya:    "    +
manusia.getUmur());
        }
```

```
    }
```

Saat kode diatas dieksekusi, maka variabel nama akan bernilai "`Nama dia siapa ya: Ahmad`" sesuai dengan nama baru yang telah ditambahkan lewat metode ubahNama(namaBaru);

3.2.3 Kata Kunci this

Kata kunci this digunakan dalam sebuah kelas dan digunakan untuk menyatakan objek sekarang. Contoh misal saat kita membuat sebuah parameter yang sama dengan nama atribut yang ada dalam sebuah kelas, maka jika kita menggunakan parameter tersebut untuk mengubah atribut pada kelas, maka perubahan tidak akan terjadi.

```
package bab3;
public class Manusia_3 {
    String nama;
  //private String nama;
    int umur;
  //private int umur;

    // metode dengan 1 parameter
    void ubahNama(String nama) {
        nama = nama;
      //this.nama = nama;
    }
    // metode dengan 2 parameter
    void ubahData(String namaBaru, int umur) {
        nama = nama;
        umur = umur;
      //this.nama = nama;
      //this.umur = umur;
    }
    String getNama() {
// public String getNama() {
        return nama;
    }
    int getUmur() {
// public int getUmur() {
        return umur;
    }
    void alamat() {
        String alamatnya = "UNCEN Jayapura";
        System.out.println("Alamat= " + alamatnya);
    }
}
```

Kita akan menggunakan kelas Manusia diatas seperti pada program berikut:

```
package bab3;
```

```
public class AplikasiManusia_3 {
    public static void main(String arg[]) {
        Manusia_3 manusia = new Manusia_3();
        manusia.ubahNama("Ahmad");
  System.out.println("Nama    dia    siapa    ya:    "    +
manusia.getNama());
        manusia.ubahData("Ahmad Jaya", 25);
// mengambil nilai dari metode
  System.out.println("Nama    dia    siapa    ya:    "    +
manusia.getNama());
  System.out.println("Umurnya      berapa:      "      +
manusia.getUmur());
    }
}
```

Setelah dijalankan, maka program tersebut akan menghasilkan nilai null, yang artinya nama dan alamat objek manusia tidak berubah menjadi "`Nama dia siapa ya: Ahmad`", kenapa? Hal ini karenakan jika kita membuat sebuah parameter yang sama dengan nama atribut, lalu saat kita memanggil nama atribut tersebut, maka sebenarnya bukan atribut yang kita panggil melainkan parameter. Agar kesalahan tersebut tidak terjadi, maka diperlukan kata kunci this, yang digunakan untuk menyatakan objek tersebut. jadi untuk mengubah atribut yang namanya sama dengan parameter haruslah tambahkan kata this (Aktifkan //this.nama pada contoh di atas). Saat program sebelumnya dijalankan kembali, maka hasilnya tidak akan null lagi.

3.2.4 Visibilitas Private dan Public

Java mendukung 4 visibilitas yaitu :

Visibilitas	Keterangan
Private	Hanya dapat diakses oleh kelas itu sendiri
Public	Dapat diakses oleh seluruh kelas
Protected	Hanya dapat diakses oleh kelas itu sendiri dan kelas turunannya
Tanpa visibilitas	Hanya dapat diakses oleh kelas-kelas yang berada pada suatu paket

Visibilitas private merupakan visibilitas yang dapat digunakan pada atribut, metode ataupun kelas. Gunanya visibilitas private adalah untuk menyembunyikan atribut, metode atau kelas. Atribut, metode, atau kelas yang menggunakan visibilitas hanya dapat diakses oleh objek itu sendiri. Contoh atribut yang menggunakan visibilitas private dapat dilihat pada kelas Manusia_3 diatas (Aktifkan `private String nama;`)
Pada kode diatas, atribut nama menjadi private, sehingga hanya kelas Manusia_3 itu sendiri yang bisa mengakses atribut nama, sehingga saat kelas lain mengakses atribut tersebut, maka akan terjadi error.

Pada kode diatas, maka akan terjadi error ketike kelas “AplikasiManusia_3” mengakses atribut nama melalui instansiasi objek manusia pada kelas “Manusia_3”, dikarenakan atribut tersebut bersifat private. Visibilitas public merupakan visibilitas yang dapat diterapkan pada atribut, metode dan kelas. Dengan visibilitas public, maka atribut, metode atau kelas yang memiliki sifat public tersebut dapat diakses oleh kelas manapun dan dari package (folder) manapun. Contoh, pada kode sebelumnya, kita akan menambah sebuah metode public yang bernama getNama() yang mengembalikan nama.
Dengan demikian untuk mengakses atribut nama, sekarang kita dapat menggunakan metode getNama()

3.2.5 Konstruktor

Konstruktor merupakan metode yang secara otomatis dipanggil ketika sebuah objek dipanggil. Cara membuat metode konstruktor adalah, nama metode harus saya dengan nama kelas dan tidak mengembalikan nilai balik dan tidak pula menggunakan kunci void. Konstruktor mendukung penggunakan parameter, misal saat membuat sebuah objek manusia, maka nama namusia tersebut harus ditentukan, maka kita dapat menambahkan sebuah parameter nama di konstruktor. Overloading konstruktor merupakan mekanisme dimana kita dapat membuat lebih dari satu buah konstruktor pada sebuah kelas. Namun dengan ketentuan, setiap konstruktor harus memiliki parameter yang berbeda, bisa berbeda jumlah parameternya ataupun bisa berbeda tipe data parameternya.

Contoh: Konstruktor

```
package bab3;
public class Manusia_3 {
    String nama;
    int umur;
    String alamat;
    String hobi;
  //Konstruktor
    public Manusia_3() {
    }
    //Konstruktor 1 parameter
 public Manusia_3(String alamat) {
     this.alamat=alamat;
 }
    void ubahNama(String nama) {
        //nama = nama;
        this.nama = nama;
    }
    // metode dengan 2 parameter
    void ubahData(String nama, int umur) {
        //nama = nama;
        this.nama = nama;
```

```
        //umur = umur;
         this.umur = umur;
    }
// metode dengan 2 parameter
    void ubahData(String nama, String hobi) {
         //nama = nama;
         this.nama = nama;
        //umur = umur;
         this.hobi = hobi;
    }
    String getNama() {
// untuk mengembalikan nilai gunakan kata kunci return
         return nama;
    }
    int getUmur() {
         return umur;
    }
     String gatAlamat() {
         return alamat;
    }

    void alamat() {
         String alamatnya = "UNCEN Jayapura";
         System.out.println("Alamat= " + alamatnya);
    }
}
```

Kelas AplikasiManusia_3

```
package bab3;
public class AplikasiManusia_3 {

    public static void main(String arg[]) {
         Manusia_3 manusia = new Manusia_3();
         Manusia_3 ridwan = new Manusia_3("Jayapura");
         System.out.println("Nama  dia  siapa  ya:  "  +
ridwan.gatAlamat());
         manusia.ubahNama("Ahmad");
  System.out.println("Nama   dia   siapa   ya:   "   +
manusia.getNama());
         manusia.ubahData("Ahmad Jaya", 25);
// mengambil nilai dari metode
  System.out.println("Nama   dia   siapa   ya:   "   +
manusia.getNama());
  System.out.println("Umurnya      berapa:      "      +
manusia.getUmur());
    }
}
```

3.2.7 Overloading Metode

Selain pada konstruktor, overloading juga bisa dilakukan pada metode, misal kita akan membuat dua buah metode `void ubahData(String nama, int umur)` dan `void ubahData(String nama, String hobi)`, metode pertama menggunakan parameter nama, umur dan metode kedua menggunakan parameter nama, hobi seperti contoh class Manusia_3 di atas

```
// metode dengan 2 parameter
    void ubahData(String nama, int umur) {
        //nama = nama;
        this.nama = nama;
       //umur = umur;
        this.umur = umur;
    }
// metode dengan 2 parameter
    void ubahData(String nama, String hobi) {
        this.nama = nama;
        this.hobi = hobi;
       }
```

3.2.8 Pewarisan Kelas

Pewarisan merupakan mekanisme dimana sebuah kelas dapat mewarisi seluruh atribut atau metode milik kelas lain dengan ketentuan tertentu. Misal ada sebuah kelas Karyawan dengan atribut nama, alamat, NIP, umur. Kemudian ada kelas Karyawan_Lepas dengan atribut gajiHarian, totalHari, gajiLembur, dan totalLembur. Contoh program:

```
package bab3;
public class Karyawan {
    String nama;
    String alamat;
    String NIP;
    int umur;
    public Karyawan() {
    }
    public void setData(String nama) {
        this.nama = nama;
    }
    public void setData(String alamat, String NIP, int
umur) {
        this.alamat = alamat;
        this.NIP = NIP;
        this.umur = umur;
    }
    public String getNama() {
        return nama;
    }
```

```
    public String getAlamat() {
        return alamat;
    }
    public String getNIP() {
        return NIP;
    }
    public int getUmur() {
        return umur;
    }
}

package bab3;
public class Karyawan_Lepas extends Karyawan {
    float gajiHarian;
    float totalHari;
    float gajiLembur;
    float totalLembur;
    public   void   setGaji(float   gajiHarian,   float
totalHari, float
                        gajiLembur, float totalLembur) {
        this.gajiHarian = gajiHarian;
        this.totalHari = totalHari;
        this.gajiLembur = gajiLembur;
        this.totalLembur = totalLembur;
    }

    public float getGajiharian() {
        return gajiHarian;
    }
    public float getTotalhari() {
        return totalHari;
    }
    public float getGajilembur() {
        return gajiLembur;
    }
    public float getTotallembur() {
        return totalLembur;
    }
    public float hitungGajiTotal() {
    return  (gajiHarian  *  totalHari)  +  (gajiLembur  *
totalLembur);
    }
}
```

Kelas Implementasi

```
package bab3;
public class Aplikasi_Karyawan {
```

```
    public static void main(String[] arrrr) {
        System.out.println("Membuat    objek    karyawan
Lepas");
        System.out.println("---------------------------
");
        Karyawan_Lepas hakim = new Karyawan_Lepas();
        hakim.setData("Ahmad Hakim");
        hakim.setGaji(50000, 20, 100000, 7);
        System.out.println("Nama    Karyawan=    "    +
hakim.getNama());

    System.out.println("Gaji       Total=       "       +
hakim.hitungGajiTotal());

    }
}
```

Pada kode diatas bisa bandingkan antara kelas Karyawan dan kelas Karyawan_Lepas memiliki beberapa atribut yang sama, yaitu nama, alamat, NIP, dan umur ubahNama(), dan metode yang sama diperlukan untuk semua karyawan lain seperti setData(), getNama, getAlamat, getNIM, dan getUmur. Sehingga kelas Karyawan dapat diwarisken ke kelas karyawan yang lain tidak terbatas pada Karyawan_Lepas. Dimasing-masing kelas akan membuat atribut dan metode tertentu sesuai kebutuhan.

3.2.9 Visibilitas protected

Sebelumnya kita telah membahas tentang visibilitas private dan public, kali ini kita akan membahas tentang visibilitas protected. Atribut, metode atau kelas yang ditandai dengan visibilitas protected hanya dapat diakses oleh kelas itu sendiri dan turunannya. Misal pada kelas sebelumnya kita telah membuat kelas Karyawan dan kelas Karyawan_Lepas. Pada kelas Karyawan, visibilitas untuk atribut umur adalah private, artinya hanya kelas Karyawan tersebut yang dapat mengakses atribut tersebut, walaupun kelas Karyawan_lepas merupakan turunan dari kelas Karyawan, tetap saja kelas Karyawan_Lepas tidak dapat mengakses atribut umur dari kelas Karyawan.

```
package bab3;
public class Karyawan {
    String nama;
    String alamat;
    String NIP;
    //private int umur;
    protected int umur;
    public Karyawan() {
    }
    public void setNama(String nama) {
```

```
        this.nama = nama;
    }
    public void setUmur(int umur) {
        this.umur = umur;
    }
    public void setData(String alamat, String NIP) {
        this.alamat = alamat;
        this.NIP = NIP;
    }
    public String getNama() {
        return nama;
    }
    public String getAlamat() {
        return alamat;
    }
    public String getNIP() {
        return NIP;
    }
    public int getUmur() {
        return umur;
    }
}
```

Jika kita menggunakan kode diatas untuk mengakses atribut nama dari kelas Orang, maka pasti akan terjadi error. Namun jika kita mengubah visibilitas nama menjadi protected maka atribut nama dapat diakses oleh kelas turunannya, yaitu kelas Pegawai.

```
public class Karyawan_Lepas extends Karyawan {
……
……
public void contoh_protected() {
  int ambilUmur = umur;
  System.out.println("Umur Karyawan= " + ambilUmur);

}
```

3.2.10 Overriding

Overriding tidak sama dengan overloading, overriding merupakan mekanisme dimana sebuah metode dapat dideklarasikan ulang pada kelas turunannya. Misal ada dua kelas yaitu Bayi dan Dewasa, pada kelas bayi tesebut terdapat metode lari() yang memerintahkan untuk lari.

```
public class Bayi {
    public void lari() {
    System.out.println("Tidak Bisa Lari");
  }
}
```

Setelah itu kelas Dewasa merupakan kelas turunan dari kelas Bayi.

```
package bab3;
public class Dewasa extends Bayi {
    public void mainBola() {
        System.out.println("Andi    sedang    bermain
bola");
    }
}
```

Setelah itu jika kita coba buat sebuah objek kelas Dewasa dan menyuruhnya lari.

```
public class Aplikasi_Dewasa {
    public static void main(String[] args) {
      Dewasa dewasa = new Dewasa();
      dewasa.mainBola();
      dewasa.lari();
   }
}
```

Maka hasilNya adalah “Andi sedang bermain bola” dan “Tidak Bisa Lari” artinya metode lari( ) yang dipanggil sebenarnya milik kelas Bayi yang pastinya tidak dapat berlari. Sekarang jika dianggap kelas Dewasa dapat berlari, maka kita harus menggubah metode lari() tersebut agar dapat berlari, caranya adalah dengan melakukan pendeklarasian ulang (overriding). Caranya adalah dengan membuat metode yang sama dengan metode yang diwarisinya.

```
public class Dewasa extends Bayi {
    public void lari() {
       System.out.println("Lari!!!!");
    }
}
```

Maka jika program Test sebelumnya dijalankan kembali, maka kelauarannya pasti “Lari!!!!”, artinya metode lari( ) milik kelas Dewasa yang dipanggil.

3.2.11 Kata Kunci super

Kata kunci super merupakan kata kunci yang digunakan untuk mengakses kelas parent (yang diturunkan), misal jika kita menggunakan kata kunci super pada kelas Dewasa artinya super tersebut merujuk pada kelas Bayi.

```
package bab3;
public class Dewasa extends Bayi {
    public void mainBola() {
        System.out.println("Andi    sedang    bermain
bola");
```

```
    }
/*
    @Override
    public void lari() {
        System.out.println("Lari !!!!!");
    }
*/

/*
    public void lariBayi() {
// mengakses metode lari milik Bayi
        super.lari();
    }
 */
}
```

Selain itu, kata kunci super juga dapat digunakan untuk mengakses konstruktor milik kelas yang diwariskan.

```
package bab3;
public class Bernama {
    private String nama;
    public Bernama() {
    }
    public Bernama(String nama) {
        this.nama = nama;
    }
    public String getNama() {
        return nama;
    }
}
package bab3;
public class Berumur extends Bernama{
private int umur;
  public Berumur() {
  }
public Berumur(String nama, int umur){
  super(nama);
  this.umur = umur;
  }
public int getUmur(){
  return umur;
  }
}
package bab3;
public class Aplikasi_Berumur {
```

```
    public static void main(String[] args) {
        Berumur berumur = new Berumur("Awan",75);
        System.out.println("Nama:                    "
+berumur.getNama());
        System.out.println("Umur:                    "
+berumur.getUmur());
    }

}
```

3.2.12 Kata Kunci final

Kata kunci final merupakan kata kunci yang dapat digunakan untuk menandai bahwa suatu atribut, metode atau kelas sudah final, artinya tidak dapat diubah lagi.

Lokasi final	Keterangan
Atribut	Atribut tidak dapat dideklarasikan lagi
Variabel	Variabel tidak dapat dideklarasikan lagi
Metode	Metode tidak dapat dideklarasikan (overriding) lagi
Kelas	Kelas tidak dapat diturunkan lagi

Jika kita menambahkan sebuah atribut dengan kata kunci final, maka atribut tersebut harus langsung dideklarasikan, misal seperti ini.

```
public class Katakunci_final {
  private final int data_1 = 5;
}
```

Jika tidak dideklarasikan langsung, maka akan terjadi kesalahan (error). Atau jika kita melakukan pendeklarasian ulang atribut tersebut maka akan terjadi error.

```
package bab3;
public class AplikasiKatakunci_final {
    private final int data_1 = 5;
    private int data_2;
  /*
    public void ubahData_1(int data_1) {
    //Terjadi Error
    this.data_1 = data_1;
    }
  */

    public void ubahData_2(int data_2) {
        this.data_2 = data_2;
    }
```

```
public static void main(String[] args) {
AplikasiKatakunci_final pinal = new

AplikasiKatakunci_final();
   pinal.ubahData_2(7);
   System.out.println("Hasil Kali= " + pinal.data_1
* pinal.data_2);
    }
}
```

3.2.13 Kelas Abstract

Kelas abstract merupakan kelas dimana memiliki metode-metode namun tidak dideklarasikan, pendeklarasiannya terjadi pada kelas turunannya. Untuk membuat kelas abstract sama dengan membuat kelas biasanya, namun diawali dengan kunci abstract pada kelasnya dan diawali dengan kata kunci abstract pada metode yang akan dibuat namun tidak akan dideklarasikan. Metode yang abstract tidak perlu berisikan deklarasinya. Misal kita membuat kelas abstract Hewan, lalu turunannya; Kucing, Kambing dan Anjing. Kelas hewan tersebut memiliki metode bicara() yang menyuruh Hewan tersebut bicara(), namun karena setiap hewan biasanya berbeda nada bicaranya, maka kita buat metode bicara() tersebut menjadi abstract.

```
package bab3;
public abstract class Abstract_Hewan {
    public abstract void bicara();
}

package bab3;
public class Hewan_Anjing extends Abstract_Hewan {
    private String nama;
    public void setNama(String nama){
      this.nama=nama;
    }
    public String getNama(){
        return nama;
    }
    public void bicara() {
        System.out.println("Gog gog...");
    }
}

package bab3;
public class Hewan_Kambing extends Abstract_Hewan {
    public void bicara() {
```

```
            System.out.println("Kambing Mengembik...");
        }
    }

    public class Kucing extends Abstract_Hewan {
       public void bicara() {
          System.out.println("Meong...");
      }
    }
```

Jika kelas turunan dari Hewan tidak mendeklarasikan metode bicara() maka akan terjadi error, kecuali kelas tersebut juga kelas abstract.

3.2.14 Polimorfisme

Polimorfisme merupakan kemampuan untuk sebuah kelas memiliki banyak kelas turunan. Setiap kelas turunan memiliki deklarasi masing-masing yang unik dan dapat berbagi fungsionalitas yang sama dengan kelas parent (yang diturunkan). Contoh polimorfisme adalah kelas Hewan yang sebelumnya telah dibuat.

```
package bab3;
public class Aplikasi_Hewan {
    public static void main(String[] args) {
        Hewan_Anjing hewan1 = new Hewan_Anjing();
        hewan1.setNama("Dogii");
        System.out.println(""+hewan1.getNama());
        hewan1.bicara();
        Hewan_Kambing hewan2 = new Hewan_Kambing();
        hewan2.bicara();
/*Hewan_Kucing hewan3 = new Hewan_Kucing();
        hewan3.bicara();*/
    }
}
```

3.3 Paket

Dalam Java, beberapa kelas dapat digabungkan dalam sebuah unit bernama paket (package). Penggunaan paket sangat dianjurkan agar kelas-kelas terlihat lebih teratur. Untuk mendeklarasikan paket, hanya perlu menggunakan kunci package pada bagian atas file java diikuti nama paket. Nama paket tidak boleh diawali dengan nomor dan tidak boleh mengandung karakter unik dan spasi. Paket biasanya bertingkat, untuk memberikan tingkatan pada paket kita dapat mengunakan tanda . (titik), misal.

Paket Folder

aplikasi.data /aplikasi/data/
aplikasi.database /aplikasi/database/
aplikasi.form /aplikasi/form/

Dianjurkan jika kita membangun sebuah sistem yang besar, maka diperlukan pengelompokkan jenis-jenis kelas dalam paket. Misal untuk kelas-kelas tabel dapat di masukkan ke paket data, kelas-kelas form bisa dimasukkan ke paket form, dan lain-lain.

```
package aplikasi.data;
public class Karyawan {
public String nip;
public String nama;
public String alamat;
}
```

Jika kita akan menggunakan kelas dengan lokasi paket yang sama, kita dapat menggunakannya langsung, namun jika kita akan menggunakan kelas dengan paket yang berbeda, maka kita perlu menggunakan import disertai lokasi paket dan nama kelasnya.

```
package aplikasi.program;
import aplikasi.data.Karyawan;
public class Program {
public static void main(String[] args) {
Karyawan karyawan = new Karyawan();
}
}
```

3.3.1 Visibiliti Default

Sebelumnya telah dibahas tentang visibiliti private, public dan protected. Sebenarnya ada satu lagi visibility, yaitu default, namun tidak menggunakan kata kunci default, melainkan tidak perlu menggunakan kata kunci (kosong). Jika sebuah atribut, metode atau kelas ditandai dengan visibiliti default, maka itu artinya atribut, metode atau kelas tersebut hanya dapat diakses oleh kelas-kelas yang ada dalam satu paket. Jika akan diakses dari luar paket, maka akan terjadi error.

Contoh visibiliti default :

```
package aplikasi.data;
public class Mahasiswa {
String nim;
String nama;
}
```

3.4 Interface

Interface merupakan mekanisme dimana kita dapat menentukan metodemetode yang harus ada pada kelas. Interface hampir mirip dengan kelas abstrak, namun ada beberapa perbedaan pada interface dan kelas abstrak.

1. Kelas abstrak bisa mengandung metode abstrak dan metode tidak abstrak, sedangkan pada interface harus semua metode abstrak.
2. Kelas abstrak dapat memiliki atribut, sedangkan interface tidak boleh memiliki atribut.
3. Kelas abstrak digunakan oleh kelas lain menggunakan pewarisan (extends), sedangkan interface menggunakan implementasi (implements).

3.4.1 Mendeklarasikan Interface

Interface mirip dengan Kelas, hanya yang membedakan adalah kata kunci yang digunakan bukan class melainkan interface. Contoh sederhana sebuah interface.

```
package aplikasi.prototype;
public interface Aksi {
public abstract void beraksi();
}
```

Secara default, seluruh metode yang ada dalam interface itu bersipat abstract dan public, sehingga kita dapat menghapusnya menjadi lebih sederhana seperti berikut.

```
package aplikasi.prototype;
public interface Aksi {
void beraksi();
}
```

Perlu diingat bahwa metode dalam interface tidak dapat private.

3.4.2 Pewarisan Interface

Dalam hal pewarisan interface, sama dengan class, hanya yang membedakan adalah interface dapat mewarisi lebih dari satu interface, sedangkan class hanya dapat mewarisi satu kelas.

```
package aplikasi.prototype;
public interface Tendangan {
void tendang();
}
package aplikasi.prototype;
public interface Pukulan {
void pukul();
}
package aplikasi.prototype;
public interface Aksi extends Tendangan,
Pukulan{
}
```

3.4.3 Menggunakan Interface

Sebuah kelas dapat menggunakan interface melalui kata kunci implements, berbeda dengan extends, sebuah kelas dapat menggunakan beberapa interface menggunakan implements.

```
package aplikasi.program;
import aplikasi.prototype.Aksi;
public class ContohAksi implements Aksi{
public void tendang() {
System.out.println("Tendang");
}
public void pukul() {
System.out.println("Pukul");
}
}
```

Karena interface Aksi merupakan turunan dari interface Tendangan dan Pukulan, maka semua kelas yang mengimplementasi interface Aksi, harus mendeklarasikan seluruh metode yang ada pada interface Aksi, Tendangan dan Pukulan.

3.5 Inner Class

Java mendukung pembuatan kelas di dalam kelas. Cara membuat kelas di dalam kelas sama dengan membuat kelas seperti biasanya, hanya lokasinya berada dalam sebuah badan kelas, misal.

```
package aplikasi.program;
public class Luar {
private String data;
public void ubahData(String data) {
this.data = data;
}
public String ambilData() {
return data;
}
public class Dalam {
private String contoh;
public void ubahContoh(String contoh) {
this.contoh = contoh;
}
public String ambilContoh() {
return contoh;
}
}
}
```

3.5.2 Kata Kunci static

Kata kunci static sebenarnya merupakan penyelewengan dari konsep pemrograman berorientasi objek. Dengan menggunakan kata kunci static, kita dapat mengakses sebuah atribut atau metode dari kelas secara langsung tanpa harus membuat objek kelas tersebut. Sehingga ini menyalahi aturan pemrograman berorientasi objek yang menyatakan bahwa untuk mengakses sebuah atribut atau metode harus melalui objek. Kata kunci static biasanya digunakan jika kita akan membuat sebuah kelas utilitas, sehingga kita dapat dengan mudah menggunakan metode-metode yang ada dalam kelas tersebut tanpa membuat objeknya. Misal.

```
package aplikasi.program;
public class FungsiMatematika {
public static int tambah(int a, int b) {
int c = a + b;
return c;
}
public static int kali(int a, int b) {
int c = a * b;
return c;
}
}
```

Dengan begitu kita dapat langsung mengakses metode kali dan tambah tanpa membuat objek FungsiMatematika, seperti :

```
package aplikasi.program;
public class Program {
public static void main(String[] args) {
int a = 10;
int b = 10;
int c = FungsiMatematika.kali(a, b);
}
}
```

Perlu diingat jika metode static hanya dapat memanggil menggunakan atribut atau metode static lainnya, artinya jika kita memanggil metode non static dalam metode static secara langsung, maka akan terjadi error.

```
package aplikasi.program;
public class FungsiMatematika {
public static int kali(int a, int b) {
contoh();
int c = a * b;
```

```
return c;
}
public void contoh(){
// hanya contoh
}
}
```

BAB 4
PENANGANAN KESALAHAN

Ada dua jenis kesalahan, pertama kesalahan pada saat kompilasi ada pula kesalahan ada saat berjalan. Biasanya kesalahan kompilasi dapat langsung terjadi ketika proses kompilasi sehingga proses kompilasi akan dibatalkan. Namun jika kesalahannya tersebut adalah kesalahan saat berjalan, maka program akan berhasil berjalan, namun saat kesalahan tersebut terjadi, maka program akan menjadi error. Contoh kesalahan misalnya:

```
package aplikasi.program;
public class Program {
public static void main(String[] args) {
int a = 10;
int b = 0;
int c = 10 / b;
System.out.println(c);
}
}
```

Sekilas mungkin tidak ada yang salah dengan kode diatas, yup dan kode diataspun dapat dikompile dengan baik. Namun saat dijalankan, akan terjadi error, yaitu error karena terjadi pembagian 0, dimana hasil dari pembagian 0 adalah tidak terdefinisikan. Oleh karena itu perlu dilakukan penanganan kesalahan.

4.1 Menangkap Kesalahan

Agar kesalahan yang terjadi dapat di ditangkap, maka kita dapat menggunakan try catch.

```
try{
// isi yang memungkinkan error
}catch(jenis error){
// dijalankan jika terjadi error
}
```

Misal pada kode sebelumnya kita telah membuat sebuah program yang melakukan proses pembagian 0 yang menyebabkan error, maka kita dapat melakukan penanganan kesalahnnya.

```
package aplikasi.program;
public class Program {
public static void main(String[] args) {
try {
int a = 10;
int b = 0;
int c = 10 / b;
```

```
System.out.println(c);
} catch (Throwable e) {
System.out.print("Ups, terjadi error :");
System.out.println(e.getMessage());
}
}
}
```

Jika program diatas dijalankan, maka akan menghasilkan keluara"Ups, terjadi error :/by zero. Kelas Throwable merupakan kelas kesalahan yang paling tinggi, jadi kita dapat menangani seluruh kesalahan menggunakan kelas Throwable.

4.2 Penanganan Secara Bertingkat

Try catch tidak hanya dapat ditangani oleh satu kelas Exception, dapat juga ditangani oleh beberapa kelas Exception seperti berikut :

```
try{
// blok yang memungkinkan terjadi error
}catch(jenis error 1){
// jika jenis error 1 terjadi
}catch(jenis error 2){
// jika jenis error 2 terjadi
}catch(jenis error 3){
// jika jenis error 3 terjadi
}
```

Penanganan secara beringkat harus bertingkat pula jenis kelas error nya, paling bawah haruslah kelas Exception yang paling tinggi, misal kelas Throwable.

4.3 Melontarkan Exception

Kadang ada kalanya kita perlu membuat kesalahan. Misal saat kita membuat sebuah kelas Mahasiswa, maka nim wajib dimasukkan, lalu jika nim tidak dimasukkan, maka dianggap salah.

```
package aplikasi.data;
public class Mahasiswa {
private String nim;
public String getNim() {
return nim;
}
public void setNim(String nim) throws Throwable
{
if (nim == null) {
throw new Throwable("Nim Harus Diisi");
```

```
}
this.nim = nim;
}
}
```

Untuk melontarkan kesalahan kita harus menggunakan kunci throw dan metode yang memungkinkan melontarkan kesalahan harus memiliki throws diikuti dengan kelas Exception nya. Dengan begitu jika kita akan mengubah nim, maka harus ditangani kesalahannya menjadi seperti ini :

```
package aplikasi.program;
import aplikasi.data.Mahasiswa;
public class Program {
public static void main(String[] args) {
try {
Mahasiswa mahasiswa = new Mahasiswa();
mahasiswa.setNim(null);
} catch (Throwable e) {
System.out.print("Ups, terjadi error :");
System.out.println(e.getMessage());
}
}
}
```

4.4 Membuat Kelas Exception

Kelas Exception merupakan kelas error yang dapat digunakan untuk membangun kelas Exception yang harus ditangani. Error ini bisa dibilang compilation exception artinya wajib ditangani. Untuk membuat kelas Exception, kita hanya perlu membuat kelas turunan dari kelas Exception.

```
package aplikasi.error;
public class ErrorWajib extends Exception {
public ErrorWajib(String message) {
super(message);
}
}
```

4.5 Membuat Kelas RuntimeException

Kelas RuntimeException merupakan kelas error yang errornya terjadi ketika aplikasi berjalan, artinya error jenis ini tidak perlu langsung di catch. Mirip dengan pembagian dengan 0. Kita dapat tidak menangkap error tersebut. Untuk membuat error jenis ini, kita dapat membuat kelas turunan dari RuntimeException.

```
public class GakBolehKosong extends
RuntimeException {
```

```
public GakBolehKosong(String message) {
super(message);
 }
}
```

Misal kita ubah error pada nim kelas Mahasiswa menjadi error tersebut.

```
package aplikasi.data;
import aplikasi.error.GakBolehKosong;
public class Mahasiswa {
private String nim;
public String getNim() {
return nim;
}
public void setNim(String nim) throws
GakBolehKosong {
if (nim == null) {
throw new GakBolehKosong("Nim Harus Diisi");
}
this.nim = nim;
}
}
```

Dengan demikian, tanpa menangkap errorpun, kita dapat langsung memanggil metode setNim(nim)

```
package aplikasi.program;
import aplikasi.data.Mahasiswa;
public class Program {
public static void main(String[] args) {
Mahasiswa mahasiswa = new Mahasiswa();
mahasiswa.setNim("10106031");
  }
}
```

4.6 Blok Finally

Blok finally merupakan blok yang selalu dijalankan pada proses try catch, baik itu terjadi error ataupun tidak. Blok finally terdapat pada bagian akhir try catch. Contoh :

```
package aplikasi.program;
import aplikasi.data.Mahasiswa;
import aplikasi.error.GakBolehKosong;
public class Program {
public static void main(String[] args) {
try {
Mahasiswa mahasiswa = new Mahasiswa();
```

```
mahasiswa.setNim("10106031");
} catch (GakBolehKosong error) {
System.out.print("Terjadi Error : ");
System.out.println(error.getMessage());
} finally {
System.out.println("Pasti Dijalankan");
}
}
}
```

Jika program diatas dijalankan, maka akan keluar tulisan “Pasti Dijalankan” dan walaupun kita masukkan data salah :

```
package aplikasi.program;
import aplikasi.data.Mahasiswa;
import aplikasi.error.GakBolehKosong;
public class Program {
public static void main(String[] args) {
try {
Mahasiswa mahasiswa = new Mahasiswa();
mahasiswa.setNim(null);
} catch (GakBolehKosong error) {
System.out.print("Terjadi Error : ");
System.out.println(error.getMessage());
} finally {
System.out.println("Pasti Dijalankan");
}
}
}
```

Maka blok finally akan selalu dijalankan, walaupun program diatas error.

BAB 5
KELAS-KELAS JAVA

5.1 String

Seperti yang telah dibahas pada materi tipe data, String bukanlah tipe data, String adalah sebuah kelas. Namun kelas String memiliki keunikan yaitu kita dapat menggunakan String tanpa mendeklarasikannya terlebih dahulu.
String data = "hehehe";
Jadi tidak perlu membuat sebuah String dahulu.
String data = new String();
Dikarenakan String merupakan kelas, sehingga String pun memiliki banyak metode yang dapat kita gunakan untuk melakukan proses manipulasi String tersebut, seperti menjadikan seluruh hurufnya besar (upper), kecil (upper) dan lain-lain.
Contoh :

```
String data = "hehehe";
String hasil = data.toUpperCase();
System.out.println(hasil);
```

Hasil dari perintah diatas adalah upper dari “hehehe” yaitu “HEHEHE”.

5.1.1 Menggabungkan String

String merupakan objek yang unik, bahkan kita dapat menggabung dua buah string atau lebih. Ada dua cara menggabungkan String, yaitu menggunakan tanda + (tambah)
String hasil = "satu " + "dua " + "tiga " + "empat";
Atau dapat menggunakan metode concat agar lebih terlihat berorientasi objek.
String hasil = "satu ".concat("dua ").concat("tiga ").concat("empat");

5.1.2 Membandingkan String

Kadang ada kalanya kita melakukan perbandingkan string, misal :

```
package aplikasi.program;
public class Program {
public static void main(String[] args) {
String data1 = "a" + "b";
data1 = data1 + "c";
String data2 = "abc";
if (data1 == data2) {
System.out.println(data1 + " sama dengan " +
data2);
} else {
System.out.println(data1 + " tidak sama dengan "
+
```

```
data2);
}
}
}
```

“saat dijalankan, maka hasilnya adalah “abc tidak sama dengan abc”. Lho kok? Padahal abc pasti sama dengan abc :(
Kenyataanya adalah, perbandingan = = hanya dapat digunakan untuk membandingkan tipe data, tidak dapat digunakan untuk membandingkan kelas. String adalah kelas, maka tidak dapat dibandingkan menggunakan tanda = =.
Untuk membandingkan objek maka kita harus menggunakan metode equals( ) milik kelas tersebut, jadi seharusnya membandingkan string adalah sebagai berikut.

```
package aplikasi.program;
public class Program {
public static void main(String[] args) {
String data1 = "a" + "b";
data1 = data1 + "c";
String data2 = "abc";
if (data1.equals(data2)) {
System.out.println(data1 + " sama dengan " +
data2);
} else {
System.out.println(data1 + " tidak sama dengan "
+
data2);
}
}
}
```

Jika program diatas dijalankan, maka hasilnya adalah “abc sama dengan abc”.

5.2 Date

Date merupakan representasi untuk tanggal dalam Java. Kelas Date berapa pada paket java.util. Contoh membuat tanggal sekarang.

```
package aplikasi.program;
import java.util.Date;
public class Program {
public static void main(String[] args) {
Date date = new Date();
System.out.println(date);
```

```
}
}
```

Sayangnya walaupun Date merupakan representasi tanggal dalam Java, namun banyak metode-metode milik kelas Date yang sudah deprecated (tidak dianjurkan untuk digunakan), oleh karena itu diperlukan kelas lain untuk melakukan manipulasi Date, yaitu Calendar.

5.3 Calendar

Calendar hampir mirip dengan Date, kelas ini merupakan representasi tanggal dalam Java. Cara membuat Calendar tidak melalui sebuah konstruktor, melainkan menggunakan metode static :

```
package aplikasi.program;
import java.util.Calendar;
public class Program {
public static void main(String[] args) {
Calendar calendar = Calendar.getInstance();
}
}
```

5.3.1 Mengubah Calendar

Jika kita akan melakukan pengubah tanggal atau waktu sebuah calendar, baik itu menit, detik, jam, hari, bulan dan tahun, maka kita dapat menggunakan metode set() :

```
calendar.set(field, value);
```

Dimana field nya adalah :

Field	Keterangan
Calendar.MILLISECOND	Mengubah data milisekon
Calendar.SECOND	Mengubah data detik
Calendar.MINUTE	Mengubah data menit
Calendar.HOUR	Mengubah data jam
Calendar.DAY_OF_MONTH	Mengubah data hari dalam bulan
Calendar. DAY_OF_WEEK	Mengubah data hari dalam minggu
Calendar.DAY_OF_YEAR	Mengubah data hari dalam tahun
Calendar.MONTH	Mengubah data bulan
Calendar.YEAR	Mengubah data tahun

Contohnya :

```
package aplikasi.program;
```

```
import java.util.Calendar;
public class Program {
public static void main(String[] args) {
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.YEAR, 1988);
calendar.set(Calendar.MONTH, Calendar.DECEMBER);
calendar.set(Calendar.DAY_OF_MONTH, 29);
}
}
```

Untuk bulan, value yang dimasukkan bukanlah angka melainkan bulan yang ada dalam Calendar, misal Calendar.DECEMBER.

5.3.2 Menambah dan Mengurangi Calendar

Selain mengubah secara manual menggunakan set(). Calendar juga memiliki metode add() yang digunakan untuk menambah atau mengurangi data calendar tersebut, formatnya adalah sebagai berikut :
calendar.add(field, value);
Dimana field pada metode add() sama dengan field pada metode set(). Contoh :

```
package aplikasi.program;
import java.util.Calendar;
public class Program {
public static void main(String[] args) {
Calendar calendar = Calendar.getInstance();
// menambah 10 hari
calendar.add(Calendar.DAY_OF_MONTH, 10);
  }
}
```

Jika akan mengurai data, cukup memasukkan data negatif, misal :

```
package aplikasi.program;
import java.util.Calendar;
public class Program {
public static void main(String[] args) {
Calendar calendar = Calendar.getInstance();
// mengurangi 10 hari
calendar.add(Calendar.DAY_OF_MONTH, -10);
  }
}
```

BAB 6
PENERAPAN PEMROGRAMAN BERORIENTASI OBJEK UNTUK ADMINISTRASI SEKOLAH

6.1 Tahapan Pembuatan Program Administrasi Sekolah

Persiapan

a. Peralatan yang Diperlukan

Pada Pemrograman Java, peralatan yang diperlukan adalah :

1. Java Development Kit versi 1.6 keatas.
2. Java Runtime Environtment versi 1.6 keatas.
3. NetBeans IDE versi 6.9 keatas.

b. Java Development Kit

Java Development Kit merupakan perangkat lunak yang digunakan untuk melakukan proses kompilasi dari kode Java menjadi *bytecode* yang dapat dimengerti dan dapat dijalankan oleh Java Runtime Environtment. Java Development Kit wajib terinstall pada komputer yang akan melakukan proses pembuatan aplikasi berbasis Java. Namun Java Development Kit tidak wajib terinstall di komputer yang akan menjalankan aplikasi yang dibangun menggunakan Java.

c. Java Runtime Environtment

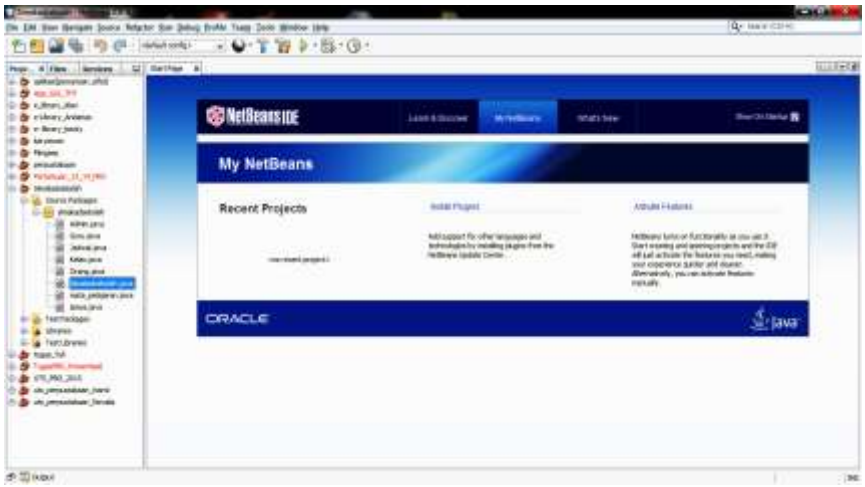
Java Runtime Environtment merupakan perangkat lunak yang digunakan untuk menjalankan aplikasi yang dibangun menggunakan java. Versi JRE harus sama atau lebih tinggi dari JDK yang digunakan untuk membangun aplikasi agar aplikasi dapat berjalan sesuai dengan yang diharapkan.

d. NetBeans IDE

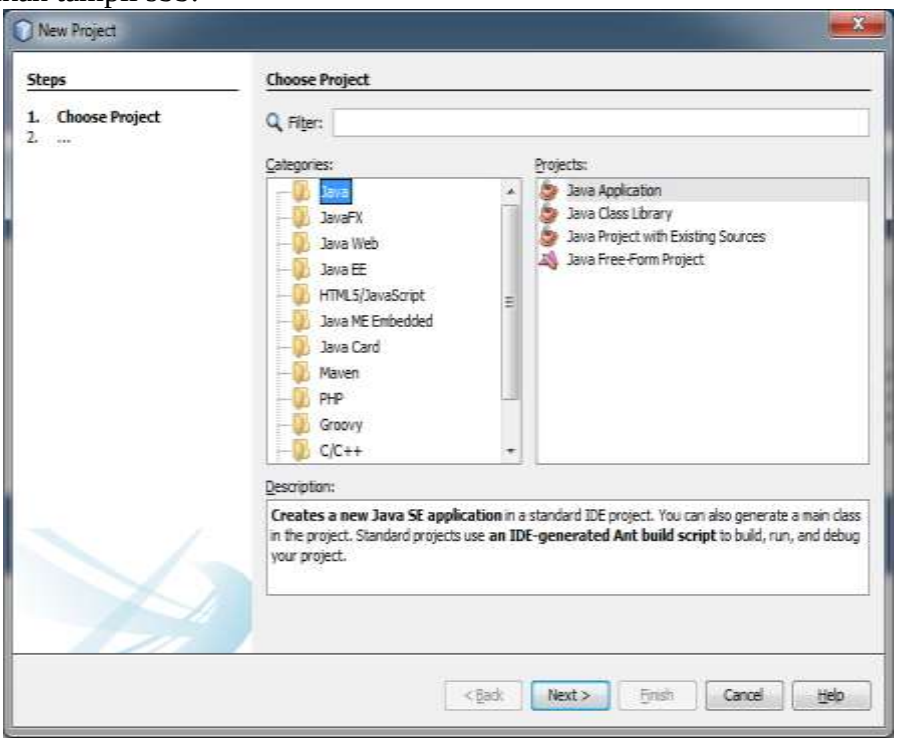
NetBeans IDE merupakan perangkat lunak yang digunakan untuk membangun perangkat lunak yang lain. NetBeans IDE dapat digunakan untuk membangun perangkat lunak berbasis Java Standard Edition, Java Enterprise Edition, Java Micro Edition, JavaFX, PHP, C/C++, Ruby, Groovy dan Python.

Setelah install NetBeans IDE 8.2, maka kita dapat memulai membuat program atau proyek administrasi sekolah. Tahapannya adalah sebagai berikut:

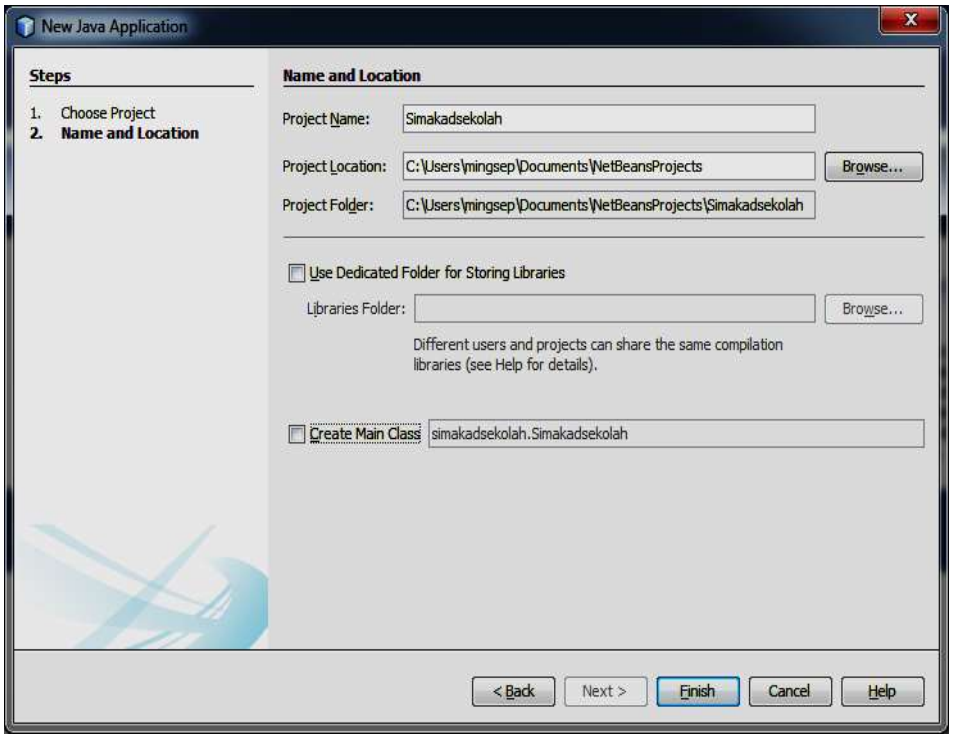
1. Jalankan aplikasi NetBeans IDE tunggu hingga muncul tampilan awal seperti gambar berikut:



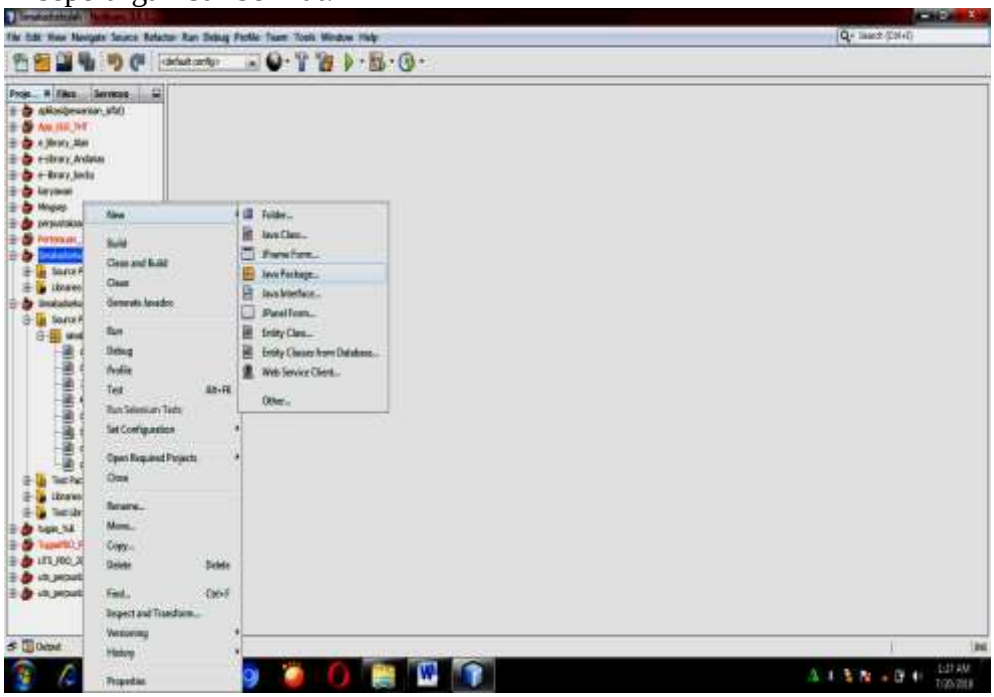
2. Buat proyek aplikasi administrasi sekolah yaitu pilih File → New Project akan tampil sbb:



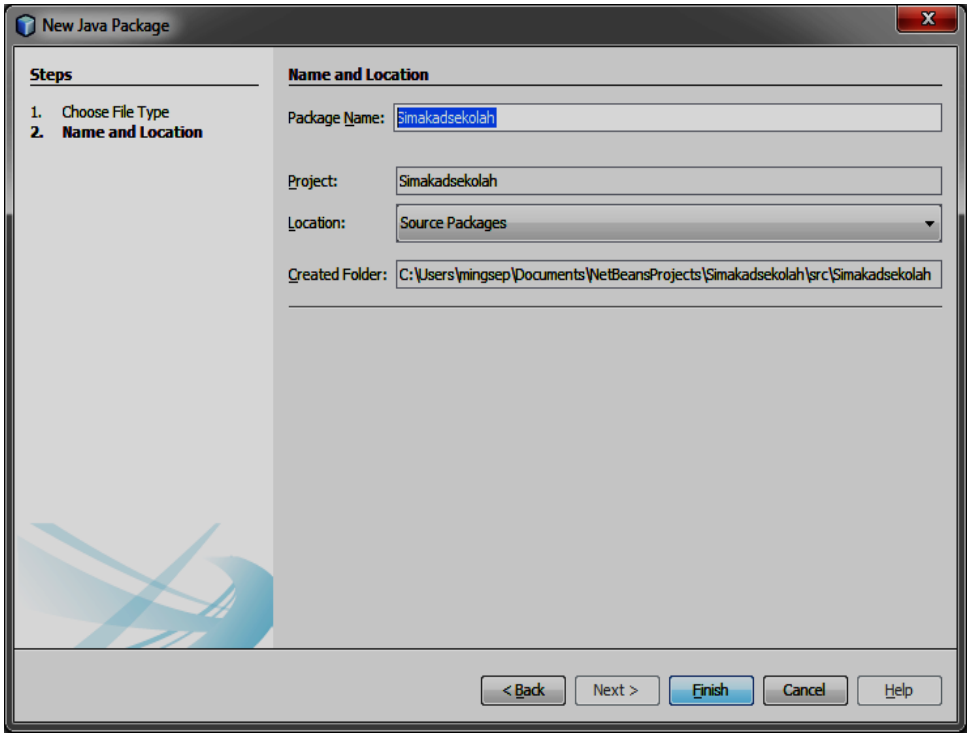
3. Pilih Next dan isi Project Name: Simakadsekolah, akan tampil gambar sbb:



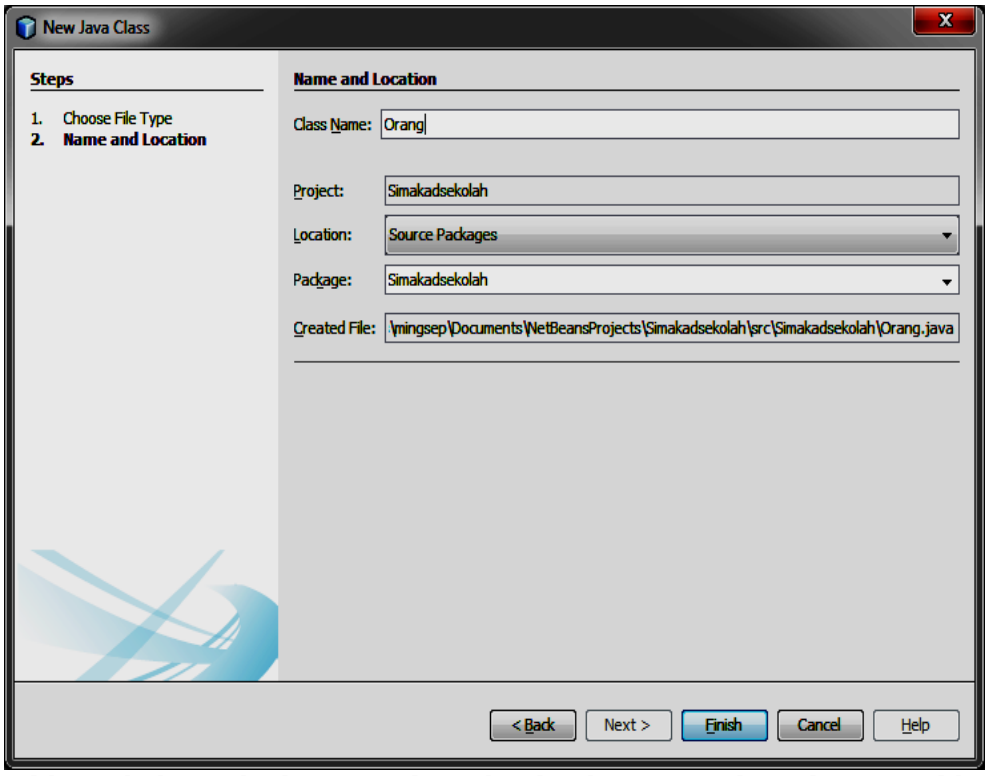
4. Pilih tombol Finish
5. Klik kanan pada project Simakadsekolah pilih New → Java Package seperti gambar berikut:



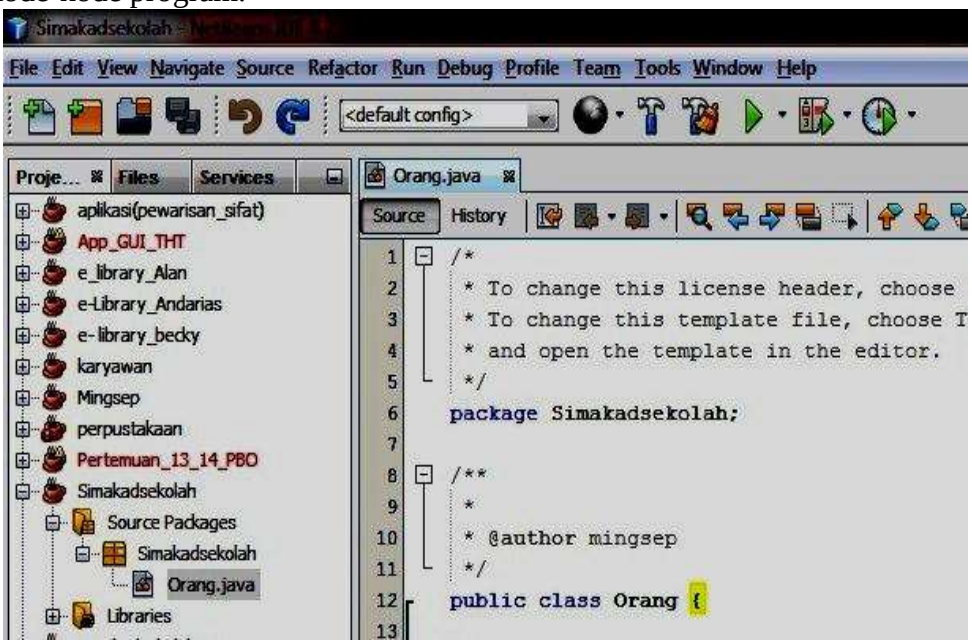
Akan tampil gambar berikut da nisi Package Name Simakadsekolah



Pilih tombol Finish, setelah itu klik kanan pada Package Simakadsekolah pilih New → Java Class akan muncul gambar berikut. Langkah ini akan dilakukan beberapa kali sejumlah kelas-kelas program untuk suatu proyek. Misalnya untuk aplikasi administrasi sekolah kita butuh 8 kelas yaitu Orang, Siswa, Guru, Admin, Jadwal, Kelas, Mata_pelajaran, dan Simakadsekolah.



Pilih tombol Finish akan tampil gambar berikut tempat kita akan menulikan kode-kode program.



6.2 Kode Program Administrasi Sekolah

6.2.1. Orang.java

```
package simakadsekolah;
import java.util.Date;
public class Orang {
    private String nama;
    private String nik;
    private String jeniskelamin;
    private Date tanggallahir;
    private String alamat;
    private String tempatlahir;
    private String agama;

    public void setOrang(String nama, String
jeniskelamin, String alamat, String agama) {
        this.nama = nama;
        this.jeniskelamin = jeniskelamin;
        this.alamat = alamat;
        this.agama = agama;
    }
    public String getNama() {
        return nama;
    }
    public void setNama(String nama) {
        this.nama = nama;
    }
    public String getNik() {
        return nik;
    }
    public void setNik(String nik) {
        this.nik = nik;
    }
    public String getJeniskelamin() {
        return jeniskelamin;
    }
    public void setJeniskelamin(String jeniskelamin) {
        this.jeniskelamin = jeniskelamin;
    }
    public Date getTanggallahir() {
        return tanggallahir;
    }
    public void setTanggallahir(Date tanggallahir) {
        this.tanggallahir = tanggallahir;
    }
    public String getAlamat() {
        return alamat;
```

```
    }
    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }
    public String getTempatlahir() {
        return tempatlahir;
    }
    public void setTempatlahir(String tempatlahir) {
        this.tempatlahir = tempatlahir;
    }
    public String getAgama() {
        return agama;
    }
    public void setAgama(String agama) {
        this.agama = agama;
    }
}
```

6.2.2. Siswa.java

```
package simakadsekolah;
public class siswa extends Orang {
    private String nim;
    private String nohp;
    private String email;
    private String username;
    private String password;

    public String getNim() {
        return nim;
    }
    public void setNim(String nim) {
        this.nim = nim;
    }
    public String getNohp() {
        return nohp;
    }
    public void setNohp(String nohp) {
        this.nohp = nohp;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getUsername() {
```

```
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

6.2.3. Guru.java

```
package simakadsekolah;
public class Guru extends Orang {
    private String nip;
    private String nohp;
    private String email;
    private String username;
    private String password;

    public String getNip() {
        return nip;
    }
    public void setNip(String nip) {
        this.nip = nip;
    }
    public String getNohp() {
        return nohp;
    }
    public void setNohp(String nohp) {
        this.nohp = nohp;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
```

```
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

6.2.4. Admin.java

```
package simakadsekolah;
public class Admin extends Orang {

    private String nohp;
    private String email;
    private String username;
    private String password;

    public String getNohp() {
        return nohp;
    }
    public void setNohp(String nohp) {
        this.nohp = nohp;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

6.2.5. Mata_pelajaran.java

```
package simakadsekolah;
public class mata_pelajaran extends Guru {
    private String kode_mapel;
    private String nama_mapel;

    public String getKode_mapel() {
        return kode_mapel;
    }

    public void setKode_mapel(String kode_mapel) {
        this.kode_mapel = kode_mapel;
    }

    public String getNama_mapel() {
        return nama_mapel;
    }

    public void setNama_mapel(String nama_mapel) {
        this.nama_mapel = nama_mapel;
    }
}
```

6.2.6. Kelas.java

```
package simakadsekolah;
public class Kelas extends mata_pelajaran {
   private String nama_kelas;
   private String ruang;
   private String wali_kelas;

    public String getNama_kelas() {
        return nama_kelas;
    }
    public void setNama_kelas(String nama_kelas) {
        this.nama_kelas = nama_kelas;
    }
    public String getRuang() {
        return ruang;
    }
    public void setRuang(String ruang) {
        this.ruang = ruang;
    }
    public String getWali_kelas() {
        return wali_kelas;
    }
    public void setWali_kelas(String wali_kelas) {
```

```
        this.wali_kelas = wali_kelas;
    }
   }
```

6.2.7. Jadwal.java

```
package simakadsekolah;
public class Jadwal extends Kelas {
    private String id_jadwal;
    private String tahun_akademik;
    private String semester;
    private String hari;
    private String jam;

    public String getId_jadwal() {
        return id_jadwal;
    }
    public void setId_jadwal(String id_jadwal) {
        this.id_jadwal = id_jadwal;
    }
    public String getTahun_akademik() {
        return tahun_akademik;
    }
    public void setTahun_akademik(String tahun_akademik)
{
        this.tahun_akademik = tahun_akademik;
    }
    public String getSemester() {
        return semester;
    }
    public void setSemester(String semester) {
        this.semester = semester;
    }
    public String getHari() {
        return hari;
    }
    public void setHari(String hari) {
        this.hari = hari;
    }
    public String getJam() {
        return jam;
    }
    public void setJam(String jam) {
        this.jam = jam;
    }
}
```

6.2.8. Simakadsekolah.java

```
package simakadsekolah;
public class Simakadsekolah {
    public static void main(String[] args) {
        Jadwal jadwal1 = new Jadwal();
        siswa siswa1 = new siswa();
        System.out.println("======Siswa 1=========");
        siswa1.setNim("201505111");
        System.out.println("Nim  :" + siswa1.getNim());
        siswa1.setOrang("Hikmah Awalia", "Perempuan",
"Jayapura", "Islam");
        System.out.println("Nama  :" +
siswa1.getNama());
        System.out.println("Jenis Kelamin  :" +
siswa1.getJeniskelamin());
        System.out.println("Alamat  :" +
siswa1.getAlamat());
        System.out.println("Agama  :" +
siswa1.getAgama());

System.out.println("================================");
        System.out.println("=======Jadwal 1==========");
        jadwal1.setNama_kelas("XII IPA");
        System.out.println("KELAS  :" +
jadwal1.getNama_kelas());
        jadwal1.setNama_mapel("Biologi");
        System.out.println("Mata Pelajaran  :" +
jadwal1.getNama_mapel());
        jadwal1.setRuang("Lab 1");
        System.out.println("Ruang  :" +
jadwal1.getRuang());
        jadwal1.setHari("Senin");
        System.out.println("Hari  :" +
jadwal1.getHari());
        jadwal1.setJam("09.00");
        System.out.println("Jam  :" + jadwal1.getJam());
        jadwal1.setNama("Siti Daryati");
        System.out.println("Guru Mapel  :" +
jadwal1.getNama());
System.out.println("==============================");
    }
}
```

BAB 7
DESAIN BERORIENTASI OBJEK MENGGUNAKAN UML

UML (Unified Modeling Language) menawarkan sebuah standar untuk merancang model sebuah sistem. UML dapat di definisikan sebagai bahasa visual untuk menjelaskan, memberikan spesifikasi, merancang, membuat model, dan mendokumentasikan aspek-aspek dari sebuah system. Karena tergolong bahasa visual, UML lebih mengedepankan penggunaan diagram untuk menggambarkan aspek dari system yang sedang dimodelkan. Fungsi UML yaitu :

a. Memberikan bahasa pemodelan yang bebas dari berbagai bahas pemrograman dan proses rekayasa.
b. Menyatukan praktek-praktek terbaik yang terdapat dalam pemodelan.
c. Memberikan model yang siap pakai, bahsa pemodelan visual yang ekspresif untuk mengembangkan dan saling menukar model dengan mudah dan dimengerti secara umum.
d. UML digambarkan dalam sketsa coretan-coretan dalam kertas atau whiteboard secara tidak normal. Biasanya digunaan dalam sesi dskusi tim untuk membahas aspek tertentu dalam tahap analisis dan perancangan
e. UML sendiri juga memberikan standart penulisan sebuah sistem blueprint yang meliputi konsep bisnis proses penulisan kelas-kelas dalam bahasa program yang spesifik, skema database dan komponen-komponen yang diperlukan dalam sistem.
f. UML berfungsi sebagai bahasa pemrograman mencoba melakukan semuanya dengan UML sampai kepada produk jadinya. Analisi dan perancangan dilakukan dengan diagram-diagram yang ada dalam UML, sementara sebuah tool atau generator bisa menghasilkan produk akhir dari diagram-diagram ini.yang ada dalam UML, sementara sebuah tool atau generator bisa menghasilkan produk akhir dari diagram-diagram ini.

Pemodelan (*modeling*) adalah proses merancang piranti lunak sebelum melakukan pengkodean (*coding*). Model piranti lunak dapat dianalogikan seperti pembuatan blueprint pada pembangunan gedung. Membuat model dari sebuah sistem yang kompleks sangatlah penting karena kita tidak dapat memahami sistem semacam itu secara menyeluruh. Semakin komplek sebuah sistem, semakin penting pula penggunaan teknik pemodelan yang baik. Dengan menggunakan model, diharapkan pengembangan piranti lunak dapat memenuhi semua kebutuhan pengguna dengan lengkap dan tepat, termasuk faktor- faktor seperti *scalability*, *robustness*, *security*, dan sebagainya. Kesuksesan suatu pemodelan piranti lunak ditentukan oleh tiga unsur, yang kemudian terkenal dengan sebuan segitiga sukses (*the triangle for success*). Ketiga unsur tersebut adalah metode pemodelan (*notation*), proses (*process*) dan *tool* yang digunakan. Memahami notasi pemodelan

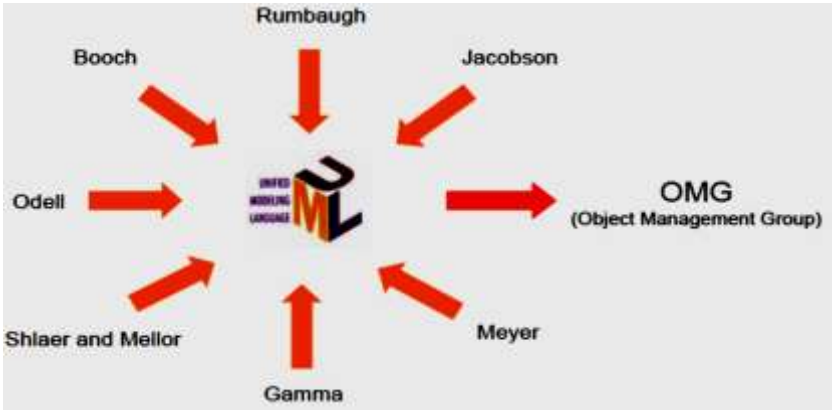
tanpa mengetahui cara pemakaian yang sebenarnya (proses) akan membuat proyek gagal. Pemahaman terhadap metode pemodelan dan proses disempurnakan dengan penggunaan *tool* yang tepat.

7.1. Pengertian UML

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem.

Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa- bahasa berorientasi objek seperti C++, Java, C# atau VB.NET, Framework Code Igniter (Apikasi web). Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (Object-Oriented Design), Jim Rumbaugh OMT (Object Modeling Technique), dan Ivar Jacobson OOSE (Object-Oriented Software Engineering).

Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: metodologi booch [1], metodologi coad [2], metodologi OOSE [3], metodologi OMT [4], metodologi shlaer-mellor [5], metodologi wirfs-brock [6], dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan.



Dimulai pada bulan Oktober 1994 Booch, Rumbaugh dan Jacobson, yang merupakan tiga tokoh yang metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease draft pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh **Object Management Group** (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. Booch, Rumbaugh dan Jacobson menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek.

7.2. Konsep Dasar UML

Dari berbagai penjelasan rumit yang terdapat di dokumen dan buku-buku UML Sebenarnya konsepsi dasar UML bisa kita rangkumkan dalam gambar:

Major Area	*View*	*Diagrams*	*Main Concepts*
structural	static view	class diagram	class, association, generalization, dependency, realization, interface
	use case view	use case diagram	use case, actor, association, extend, include, use case generalization
	implementation view	component diagram	component, interface, dependency, realization
	deployment view	deployment diagram	node, component, dependency, location
dynamic	state machine view	statechart diagram	state, event, transition, action
	activity view	activity diagram	state, activity, completion transition, fork, join
	interaction view	sequence diagram	interaction, object, message, activation
		collaboration diagram	collaboration, interaction, collaboration role, message
model management	model management view	class diagram	package, subsystem, model
extensibility	all	all	constraint, stereotype, tagged values

Abstraksi konsep dasar UML yang terdiri dari *structural classification*, *dynamic behavior*, dan *model management*, bisa kita pahami dengan mudah apabila kita melihat gambar diatas. *Main concepts* bisa kita pandang sebagai istilah-istilah yang akan muncul pada saat kita membuat diagram. *Diagrams* adalah nama diagram yang akan dibuat, dan *view* adalah kategori dari diagaram tersebut. Untuk mengerti UML, ada dua hal yang harus kita perhatikan:

1. Menguasai pembuatan diagram UML
2. Menguasai langkah-langkah dalam analisa dan pengembangan dengan UML

Diagram-diagram pada UML yaitu sebagai berikut:

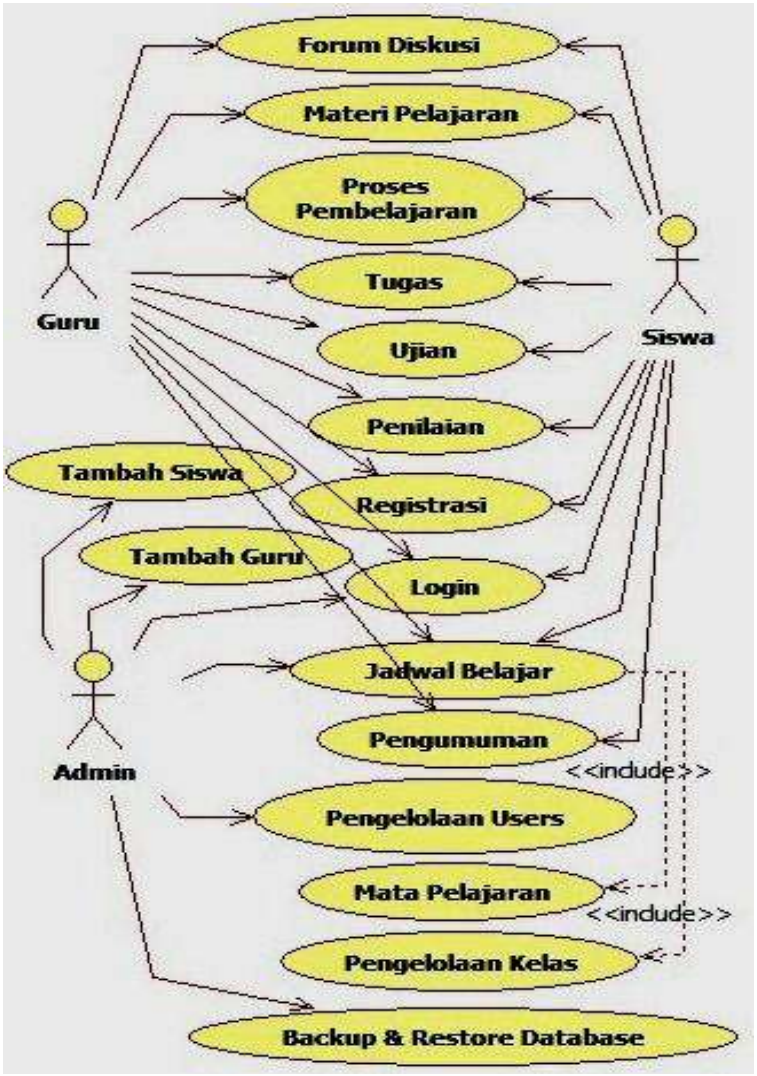
- *use case diagram*
- *class diagram*
- *activity diagram*
- *sequence diagram*
- *collaboration diagram*
- *component diagram*
- *deployment diagram*

7.3. Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan pekerjaan tertentu.

Use case diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. Contoh desain diagram *use case* aplikasi *smart school* seperti pada gambar dibawah ini menunjukkan fungsionalitas yang diharapkan dari sistem aplikasi *smart school*. *use cases* dalam aplikasi *smart school* merepresentasikan interaksi antara pengguna aplikasi dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu yang meliputi registrasi, login, forum diskusi, membuat materi pelajaran, melakukan proses pembelajarandan, tugas, ujian, penilaian, jadwal belajar, pengelolaan kelas, mata pelajaran, pengumuman, pengelolaan *users*, materi pelajaran, tambah siswa, dan tambah guru. Seorang aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. Aktor yang berperan pada aplikasi *smart school* yaitu admin, guru, dan siswa. Desain diagram *use case* aplikasi *smart school* dapat dilihat pada gambar berikut ini.



7.4. Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok :

1. Nama class
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

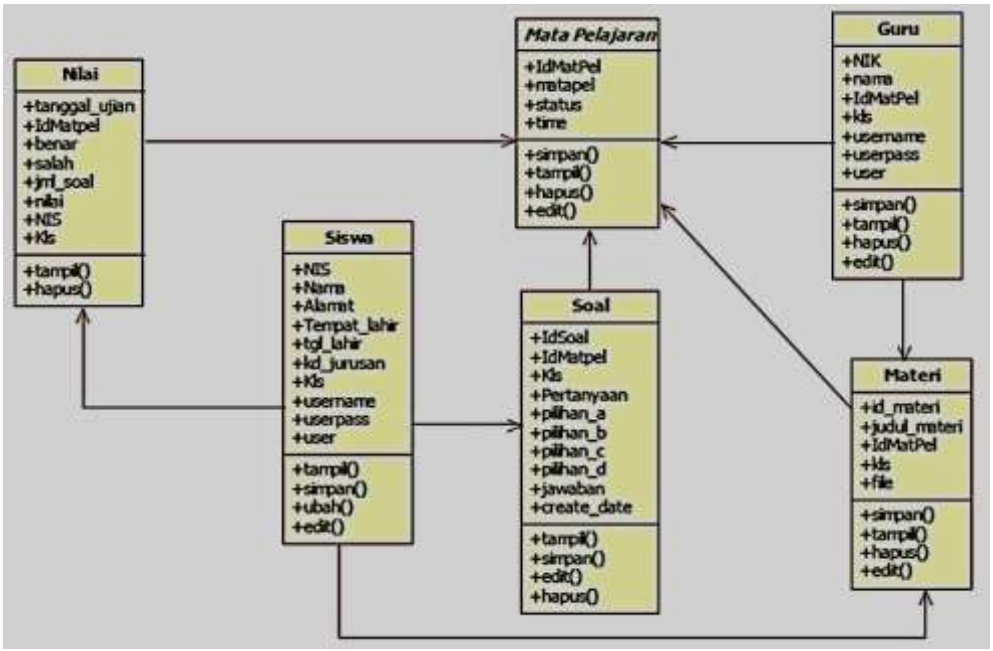
- *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
- *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak anak yang mewarisinya
- *Public*, dapat dipanggil oleh siapa saja

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan dalam satu *package*.

Hubungan Antar *Class*

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.

Contoh *class diagram* :



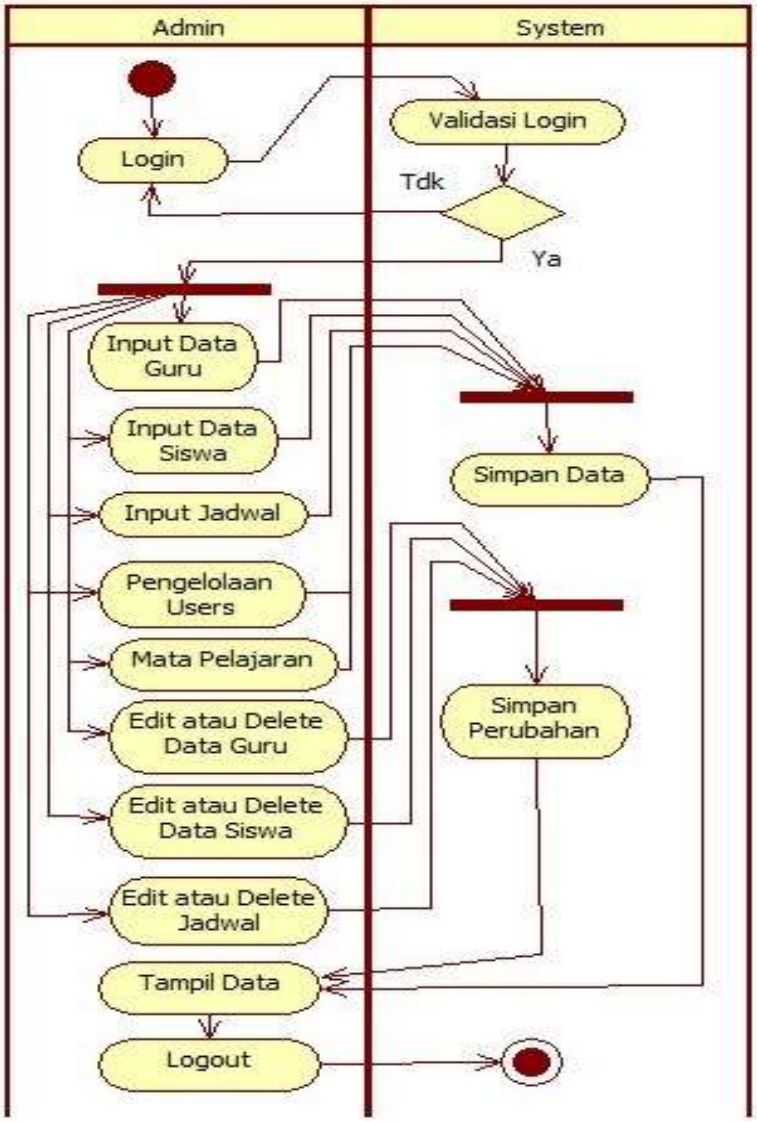
7.5. Activity Diagram

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

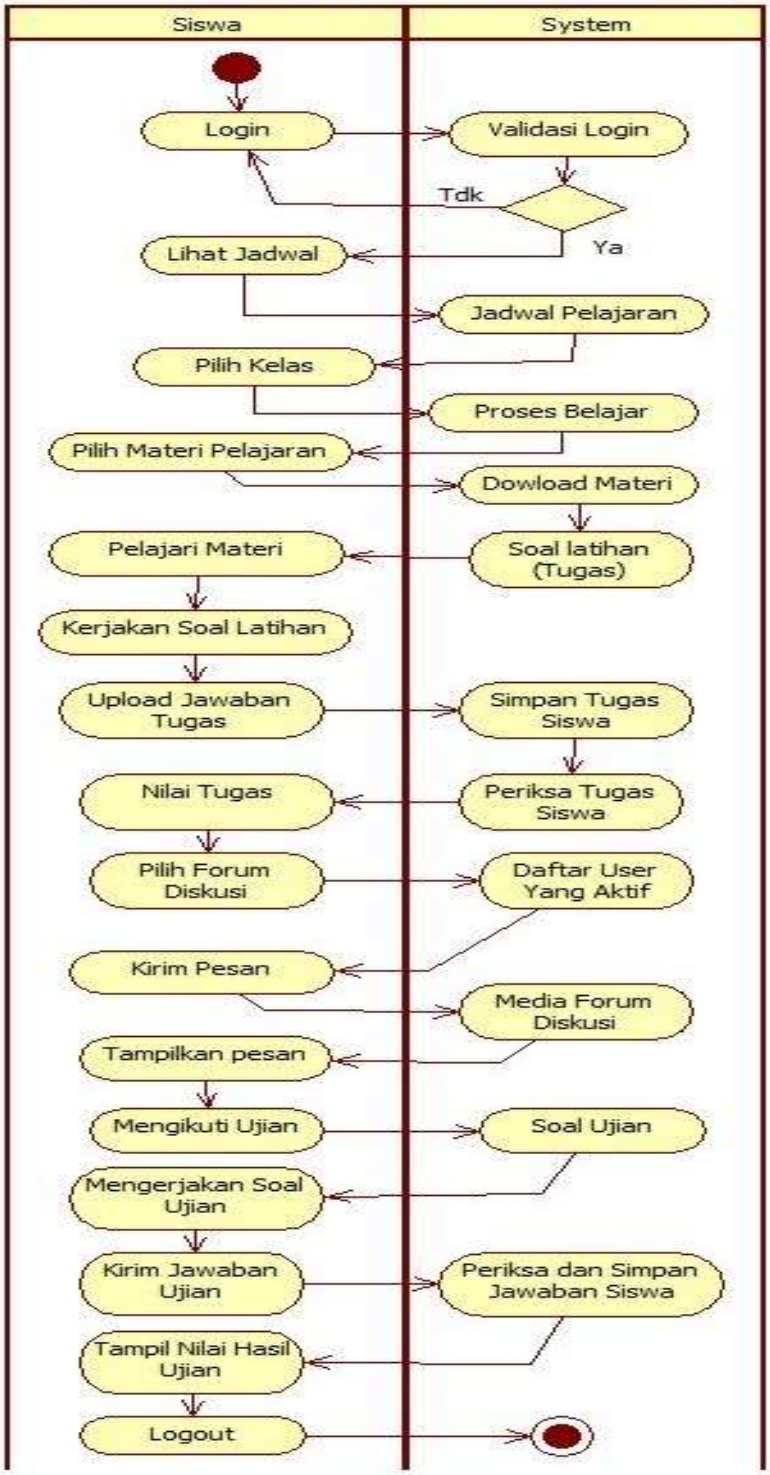
Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.

Desain diagram *activity* aplikasi *smart school* untuk admin dan siswa seperti diperlihatkan pada gambar berikut yang menggambarkan aliran

aktivitas yang ditunjukkan oleh garis dengan tanda panah dan proses-proses yang terjadi dalam aplikasi *smart school* yang ditunjukkan oleh bundaran lonjong, *decision* (belah ketupat) yang menggambarkan *behaviour* pada kondisi tertentu, dan proses paralel pada beberapa eksekusi yang terjadi secara bersamaan. *Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek yang bertanggung jawab untuk aktivitas tertentu (Admin dan *System*).



Gambar Diagram Activity Smart School Untuk Admin

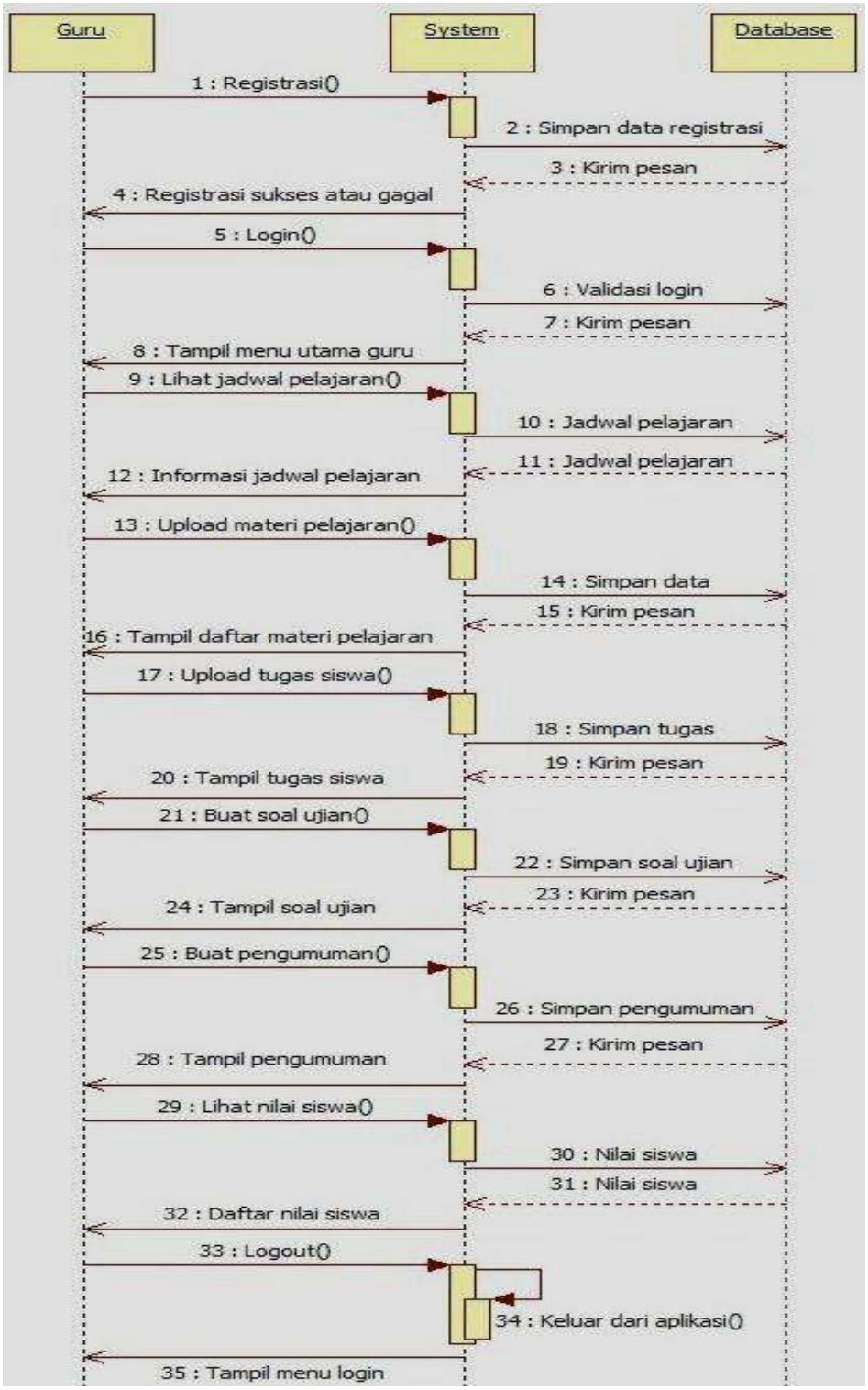


Gambar Diagram Activity Smart School Untuk Siswa

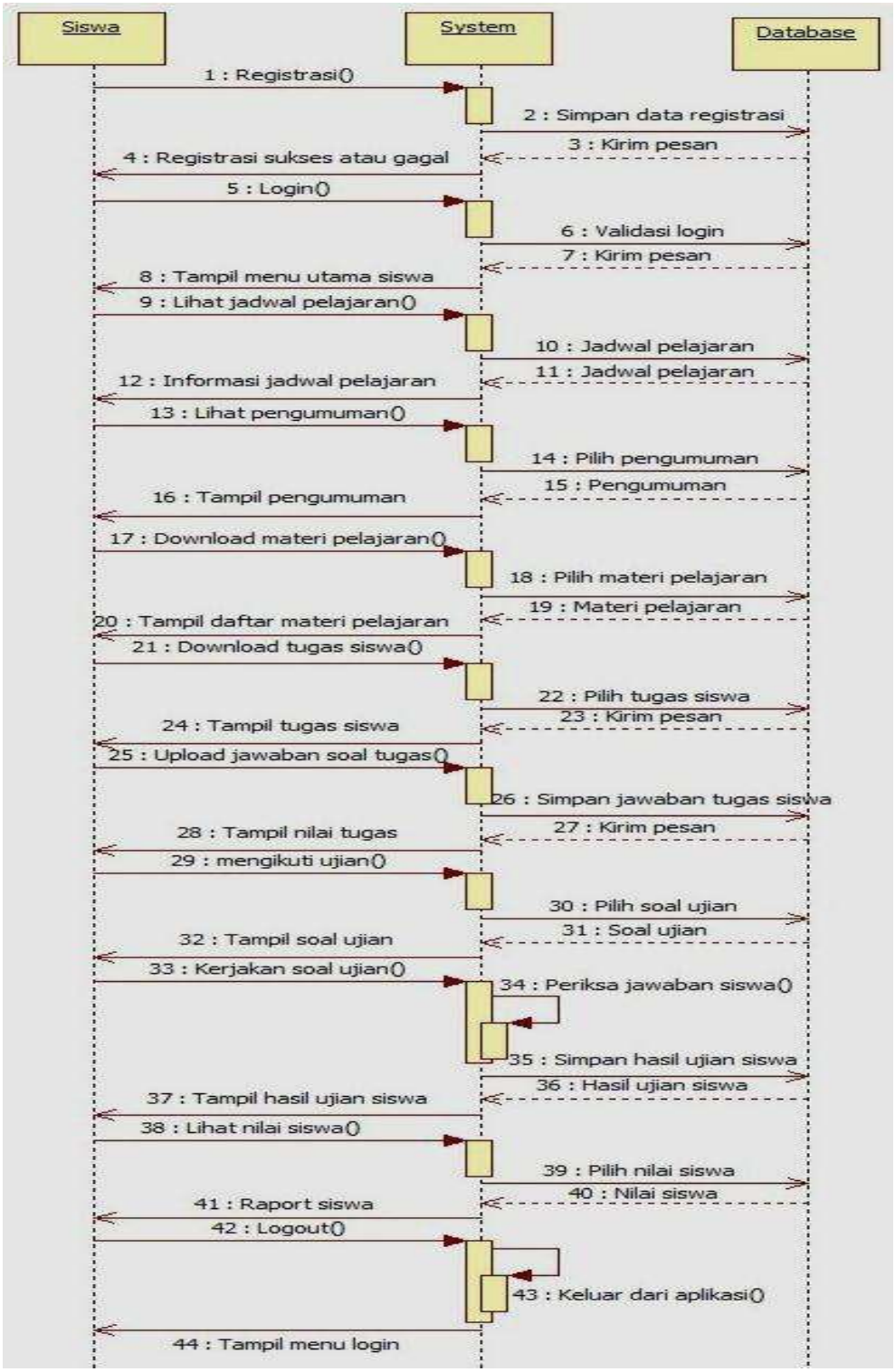
7.6. Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message.

Desain diagram *sequence* aplikasi *smart school* seperti pada gambar 8 dan gambar 9 di atas, menggambarkan interaksi antar objek di dalam dan di sekitar aplikasi (termasuk admin, guru, siswa, *display*, tabel database yang menyimpan data, dan *system*) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan dalam aplikasi *smart school* sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Gambar berikut ini memperliahatkan desain diagram *sequence* aplikasi *smart school* untuk guru dan siswa.



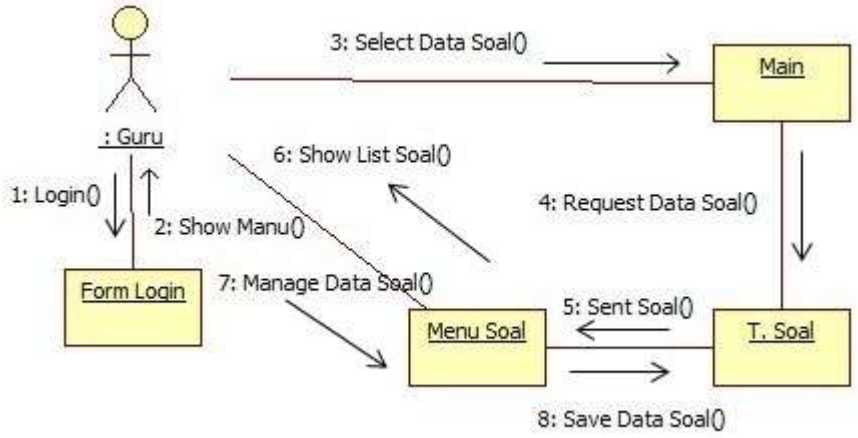
Gambar Desain Diagram Sequence Untuk Guru



Gambar Desain Diagram Sequence Untuk Siswa

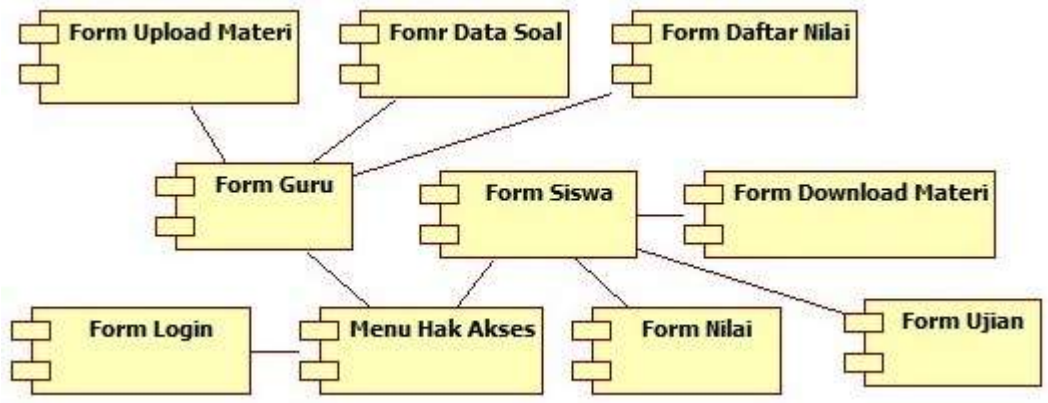
7.7. Collaboration Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap *message* memiliki *sequence number* dimulai dari nomor 1 dan seterusnya.



7.8. Component Diagram

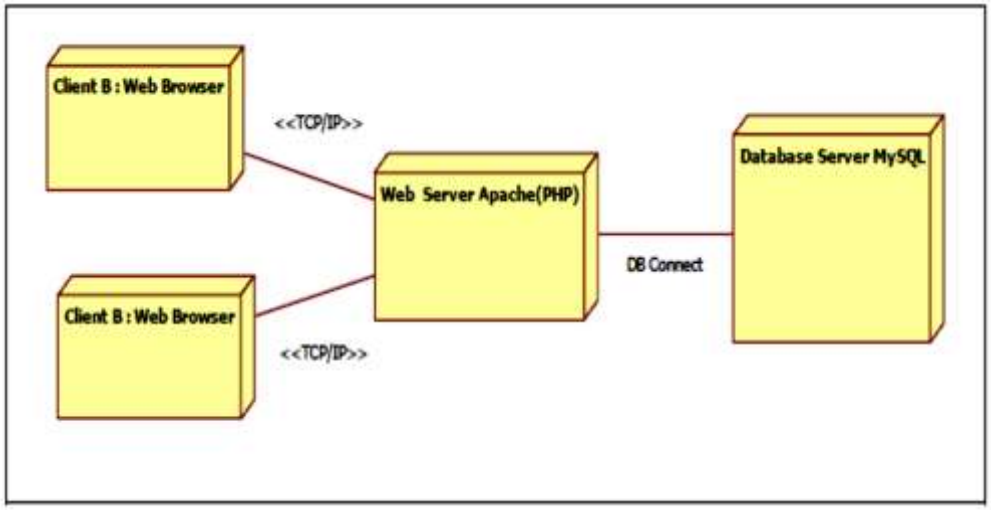
Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*. Umumnya komponen terbentuk dari beberapa *class* dan/atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain. Contoh *component diagram*:



7.9. Deploymen Diagram

Deployment / physical diagram menggambarkan detail bagaimana komponen di- *deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal Sebuah *node* adalah server, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar *node* (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.

Contoh *deployment diagram* :



DAFTAR PUSTAKA

Adi Nugroho, 2008, Algoritma dan Struktur Data Dalam Bahasa Java, Andi Yogyakarta.

Bambang Heriyanto, 2005, Esensi-esensi Bahasa Pemrograman Java, Informatika, Bandung.

Herbert Schildt, 2002, *Java2 : A beginner's Guide,* Second Edition, McGraw-Hill/Osborne

Ivar Jacobson, Grady Booch, and James Rumbaugh, 1999, The Unified Software Development Process, Addison-Wesley.

Sri Hartati, Herry Suharto, dan Soesilo Wijono, 2007, Pemrograman GUI Swing Java Dengan NetBeans 5, Andi Yogyakarta.

TENTANG PENULIS

Mingsep Rante Sampebua lahir di Bori Tana Toraja tahun 1975. Lulus Sarjana Teknik Elektronika (S1) pada Universitas Kristen Indonesia Paulus pada Tahun 1998. Pada tahun 2003 menyelesaikan pendidikan S2 pada Program Pasca Sarjana Universitas Gadjah Mada Jurusan Tenik Elektro dan Teknologi Informasi (S2) dengan bidang keahlian Teknologi Informasi. Tahun 2011 menyelesaikan pendidikan Doktor (S3) pada Program Pasca Sarjana Universitas Gadjah Mada Jurusan Tenik Elektro dan Teknologi Informasi dengan bidang keahlian Rekayasa Perangkat Lunak. Penulis adalah dosen Yayasan Universitas Kristen Indonesia Paulus pada Jurusan Teknik Informatika mulai tahun 2001 – 2014. Pada bulan april tahun 2014 diangkat menjadi dosen PNS dan aktif mengajar sampai sekarang pada Program Studi Sistem Informasi Fakultas MIPA Universitas Cenderawasih.