Java Dasar

Eko Kurniawan Khannedy

echo.khannedy@gmail.com

Daftar Isi

1 Persiapan

1.1 Peralatan yang Diperlukan

Pada pelatihan Java Dasar ini, 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.

1.2 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.

1.3 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.

1.4 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.

2 Dasar-Dasar Bahasa Pemrograman Java

2.1 Program Hello World

```
public class HelloWorld {

    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 HelloWorld, maka nama file harus HelloWorld.java.
2. Hanya boleh terdapat satu kelas public pada sebuah file.
3. Kelas yang menjadi program harus memiliki metode ***public static void main(String[] args)***
4. Terminal pada Java menggunakan tanda ; (titik koma).

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 - 32767
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
%=	Pemerolehan 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 dari sama dengan
<=	Kurang dari sama dengan
>	Lebih dari
<	Kurang dari

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	Keterangan
&&	Dan
||	Atau

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

Hasil operasi logika dengan menggunakan && adalah sebagai berikut.

Operasi 1	Operasi 2	Hasil
False	False	False
False	True	False
True	False	False
True	True	True

Hasil operasi logika dengan menggunakan || adalah sebagai berikut.

Operasi 1	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

2.5.1 Percabangan if

Pernyataan if merupakan salah satu bentuk pernyataan yang berguna untuk mengambil keputusan terhadap sebuah kemungkinan. Bentuk pernyataan if berupa :

```
if(kondisi){
     // yang akan dijalankan
}
```

Contoh :

```
int nilai = 10;

if(nilai == 10){
    System.out.println("Sepuluh");
}
```

Jika program diatas dijalankan, maka hasilnya adalah tulisan “Sepuluh” karena kondisi pada if bernilai true, jika kondisi bernilai salah, misal nilai == 100, maka program tidak akan menghasilkan tulisan apa-apa.

2.5.2 Percabangan if-else

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. Bentuk pernyataan if-else berupa :

```
if(kondisi){
    // jalankan jika kondisi true
}else{
    // jalankan jika kondisi false
}
```

Misal

```
int nilai = 8;

if(nilai == 10){
    System.out.println("Sepuluh");
}else{
    System.out.println("Bukan Sepuluh");
}
```

Jika program diatas dijalankan, maka hasilnya adalah tulisan “Bukan Sepuluh”, hal ini dikarenakan nilai bernilai 8, bukan 10.

2.5.3 Percabangan if 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
}
```

Misal.

```
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);
```

Jika program diatas dijalankan, maka hasilnya adalah ‘C’.

2.5.4 Percabangan switch-case

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

```
switch(variabel){
    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
}
```

Pada percabangan switch pertama, diperlukan sebuah variabel, setelah itu bada bagian case dibandingkan, jika sama, maka instruksi akan dijalankan sampai menemui tanda break. Misal :

```
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;
}
```

Jika program diatas dijalankan, maka program akan menghasilkan tulisan “Bukan Hari Libur”.

2.6 Perulangan

2.6.1 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
}
```

Misal :

```
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)

2.6.2 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);
```

Misal.

```
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.

2.6.3 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.

```
for(int i = 1; i <= 100; i++){
    System.out.println(i);
}
```

2.6.4 Perintah break

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

```
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.

2.6.5 Perintah continue

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

```
for(int i = 1; i <= 100; i++){
    if(i % 2 == 0){
        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.

2.7 Array

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

2.7.1 Mendeklarasikan Array

Untuk membuatan variabel array pun berbeda dengan membuat variabel 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.

2.7.2 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.

2.7.3 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	Indeks
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];
```

Misal :

```
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]);
```

2.7.4 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];
```

3 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek merupakan pemrograman yang menjadikan objek sebagai komponen utama dalam sistem. Objek merupakan gabungan data dan fungsi, dimana sebuah objek dibuat dari sebuah kelas.

3.1 Object

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 fungsi
objek.namaFungsi();
```

3.2 Class

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

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

Pada kode diatas, kelas yang telah dibuat adalah kelas Manusia. Dan nama merupakan atribut yang dimiliki kelas Manusia tersebut. Contoh pembuatan objek untuk kelas manusia adalah sebagai berikut :

```
// membuat objek manusia
Manusia manusia = new Manusia();

// mengubah nama objek manusia
manusia.nama = "Eko Kurniawan Khannedy";
```

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

Contoh :

```
class Manusia {

    String nama;

    // fungsi
    String ambilNama() {
        // untuk mengembalikan nilai gunakan kata kunci return
        return nama;
    }

    // prosedur
    void hapusNama() {
        nama = "";
    }
}
```

Pada kode diatas, kelas manusia memiliki 2 buah metode yaitu ambilNama() dan hapusNama(). Dimana ambilNama() merupakan sebuah fungsi karena

mengembalikan nilai String, sedangkan hapusNama() merupakan prosedur karena tidak mengembalikan nilai.

Saat membuat sebuah fungsi 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.

```
Manusia manusia = new Manusia();
manusia.nama = "Eko Kurniawan Khannedy";

// mengambil nilai dari fungsi
String nama = manusia.ambilNama();
```

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 :

```
class Manusia {

    String nama;

    // metode dengan parameter
    void ubahNama(String namaBaru){
        nama = namaBaru;
    }

    String ambilNama() {
        return nama;
    }

    void hapusNama() {
        nama = "";
    }
}
```

Contoh penggunaannya adalah sebagai berikut :

```
Manusia manusia = new Manusia();
manusia.ubahNama("Eko Kurniawan Khannedy");
```

```
String nama = manusia.ambilNama();
```

Saat kode diatas dieksekusi, maka variabel nama akan bernilai “Eko Kurniawan Khannedy” sesuai dengan nama baru yang telah tidambahkan lewat metode ubahNama(namaBaru);

Sebuah metode dapat memiliki satu atau lebih parameter, untuk menambahkan parameter, dipisahkan dengan menggunakan tanda , (koma). Contohnya :

```
class Manusia {

    String nama;
    String alamat;

    // metode dengan lebih dari satu parameter
    void ubahData(String namaBaru, String alamatBaru){
        nama = namaBaru;
        alamat = alamatBaru;
    }

    // metode dengan satu parameter
    void ubahNama(String namaBaru){
        nama = namaBaru;
    }

    String ambilNama() {
        return nama;
    }

    void hapusNama() {
        nama = "";
    }
}
```

Contoh penggunaannya adalah sebagai berikut :

```
Manusia manusia = new Manusia();
manusia.ubahData("Eko", "Subang");
```

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.

```
class Manusia {

    String nama;
    String alamat;

    void ubahData(String nama, String alamat){
        nama = nama;
        alamat = alamat;
    }

    void ubahNama(String nama){
        nama = nama;
    }
}
```

Saat kita menggunakan kelas Manusia diatas pada program.

```
Manusia manusia = new Manusia();
manusia.ubahData("Eko", "Subang");

System.out.println(manusia.nama);
System.out.println(manusia.alamat);
```

Setelah dijalankan, maka program tersebut akan menghasilkan nilai null, yang artinya nama dan alamat objek manusia tidak berubah menjadi “Eko” dan “Subang”, kenapa? Hal ini dikarenakan 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 sebagai berikut.

```
class Manusia {

    String nama;
    String alamat;

    void ubahData(String nama, String alamat){
        this.nama = nama;
        this.alamat = alamat;
    }
```

```
    void ubahNama(String nama){
        this.nama = nama;
    }
}
```

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 satu paket

Saat ini akan dibahas tentang visibilitas private dan public, untuk visibilitas protected akan dibahas setelah materi pewarisan dan untuk tanpa visibilitas akan dibahas setelah materi package.

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.

```
class Manusia {

    private String nama;
    String alamat;

    void ubahData(String nama, String alamat) {
        this.nama = nama;
        this.alamat = alamat;
    }

    void ubahNama(String nama) {
        this.nama = nama;
    }
}
```

Pada kode diatas, atribut nama menjadi private, sehingga hanya kelas Manusia

itu sendiri yang bisa mengakses atribut nama, sehingga saat kelas lain mengakses atribut tersebut, maka akan terjadi error.

```
public class HelloWorld {

    public static void main(String[] args) {
        Manusia manusia = new Manusia();
        manusia.ubahData("Eko", "Subang");

        System.out.println(manusia.nama);
        System.out.println(manusia.alamat);
    }
}
```

Pada kode diatas, maka akan terjadi error ketike kelas HelloWorld mengakses atribut nama objek manusia, 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 manapun.

Contoh, pada kode sebelumnya, kita akan menambah sebuah metode public yang bernama ambilNama() yang mengembalikan nama mahasiswa.

```
class Manusia {

    private String nama;
    String alamat;

    public String ambilNama() {
        return nama;
    }

    void ubahData(String nama, String alamat) {
        this.nama = nama;
        this.alamat = alamat;
    }

    void ubahNama(String nama) {
        this.nama = nama;
    }
}
```

Dengan demikian untuk mengakses atribut nama, sekarang kita dapat menggunakan metode ambilNama()

```
public class HelloWorld {

    public static void main(String[] args) {
        Manusia manusia = new Manusia();
```

```
        manusia.ubahData("Eko", "Subang");

        System.out.println(manusia.ambilNama());
        System.out.println(manusia.alamat);
    }
}
```

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. Contoh :

```
class Manusia {

    private String nama;
    String alamat;

    public Manusia() {
        System.out.println("Objek Mahasiswa Dibuat!!!");
    }

    public String ambilNama() {
        return nama;
    }

    void ubahData(String nama, String alamat) {
        this.nama = nama;
        this.alamat = alamat;
    }

    void ubahNama(String nama) {
        this.nama = nama;
    }
}
```

Dengan demikian, saat membuat sebuah objek Mahasiswa, maka konstruktor tersebut akan otomatis dipanggil. Misal jika kita membuat sebuah objek mahasiswa.

```
Manusia manusia = new Manusia();
```

Maka akan menampilkan tulisan “Objek Mahasiswa Dibuat!!!”.

Konstruktor juga mendukun penggunakan parameter, misal saat membuat sebuah objek manusia, maka nama namusia tersebut harus ditentukan, maka kita dapat menambahkan sebuah parameter nama di konstruktor seperti berikut.

```
class Manusia {

    private String nama;
    String alamat;

    public Manusia(String nama) {
        this.nama = nama;
    }

    public String ambilNama() {
        return nama;
    }

    void ubahData(String nama, String alamat) {
        this.nama = nama;
        this.alamat = alamat;
    }

    void ubahNama(String nama) {
        this.nama = nama;
    }
}
```

Dengan begitu, maka saat membaut objek manusia, maka kita harus menggunakan parameter nama

```
Manusia manusia = new Manusia("Eko Kurniawan");
```

3.2.6 Overloading Konstruktor

Overloading 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.

Misal kita akan mengubah kelas manusia tersebut menjadi memiliki dua konsturktor, dimana konstruktor pertama kita dapat membuat objek manusia tanpa harus menggunakan nama dan konstruktor kedua kita harus menggunakan nama untuk membuat objek mahasiswa.

```
class Manusia {

    private String nama;
    String alamat;

    public Manusia() {
        // tanpa parameter
    }

    public Manusia(String nama) {
        this.nama = nama;
```

```
    }

    public String ambilNama() {
        return nama;
    }

    void ubahData(String nama, String alamat) {
        this.nama = nama;
        this.alamat = alamat;
    }

    void ubahNama(String nama) {
        this.nama = nama;
    }
}
```

Dengan begitu, kita dapat menggunakan dua cara untuk membuat objek dari kelas Mahasiswa, yaitu tanpa parameter dan menggunakan parameter nama.

```
public class HelloWorld {

    public static void main(String[] args) {
        Manusia manusia1 = new Manusia();
        Manusia manusia2 = new Manusia("Eko Kurniawan Khannedy");
    }
}
```

3.2.7 Overloading Metode

Selain pada konstruktor, overloading juga bisa dilakukan pada metode, misal kita akan membuat dua buah metode ubah, metode pertama menggunakan parameter nama dan metode kedua menggunakan parameter nama dan alamat.

```
class Manusia {

    private String nama;
    private String alamat;

    public Manusia() {
        // tanpa parameter
    }

    public Manusia(String nama) {
        this.nama = nama;
    }

    public void ubah(String nama) {
        this.nama = nama;
    }

    public void ubah(String nama, String alamat){
        this.nama = nama;
        this.alamat = alamat;
```

```
    }
}
```

Dengan begitu, kita dapat menggunakan metode ubah, untuk mengubah nama ataupun untuk mengubah alamat.

```
public class HelloWorld {

    public static void main(String[] args) {
        Manusia manusia = new Manusia();
        manusia.ubah("Eko Salah");
        manusia.ubah("Eko Kurnaiwan", "Subang");
    }
}
```

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 Orang dengan atribut nama dan alamat. Lalu ada kelas Pegawai dengan atribut nip, nama dan alamat.

```
public class Orang {

    private String nama;
    private String alamat;

    public void ubahNama(String nama) {
        this.nama = nama;
    }

    public String ambilNama() {
        return nama;
    }

    public void ubahAlamat(String alamat) {
        this.alamat = alamat;
    }

    public String ambilAlamat() {
        return alamat;
    }
}
```

```
public class Pegawai {

    private String nip;
    private String nama;
    private String alamat;

    public void ubahNip(String nip) {
```

```
        this.nip = nip;
    }

    public String ambilNip() {
        return nip;
    }

    public void ubahNama(String nama) {
        this.nama = nama;
    }

    public String ambilNama() {
        return nama;
    }

    public void ubahAlamat(String alamat) {
        this.alamat = alamat;
    }

    public String ambilAlamat() {
        return alamat;
    }
}
```

Pada kode diatas bisa bandingkan antara kelas Orang dan kelas Pegawai memiliki beberapa atribut dan metode yang sama, yaitu nama, alamat, ubahNama(), ambilNama(), ubahAlamat() dan ambilAlamat(). Artinya banyak terjadi duplikasi kode, oleh karena itu lebih baik kelas tersebut digabungkan menggunakan pewarisan, yaitu Orang diturunkan menjadi Pegawai, karena semua atribut dan metode Orang ada di Pegawai namun tidak semua atribut dan metode Pegawai ada di kelas Orang.

Untuk mengatakan bahwa kelas X turunan dari kelas Y kita dapat menggunakan kata kunci extends. Dengan begitu kita hanya perlu mengubah kelas Pegawai menjadi sebagai berikut.

```
public class Pegawai extends Orang {

    private String nip;

    public void ubahNip(String nip) {
        this.nip = nip;
    }

    public String ambilNip() {
        return nip;
    }
}
```

Walaupun kelas Pegawai tidak memiliki atribut dan metode untuk nama dan alamat, namun sebenarnya Pegawai tersebut memilikinya, karena Pegawai merupakan turunan dari Orang, sehingga seluruh sifat dari Orang ada pada Pegawai.

```
public class Test {

    public static void main(String[] args) {
        Pegawai pegawai = new Pegawai();

        pegawai.ubahNama("Eko Kurniawan");
        pegawai.ubahAlamat("Subang");
        pegawai.ubahNip("10106031");
    }

}
```

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 Orang dan Pegawai.

Pada kelas Orang, visibilitas untuk atribut nama dan alamat adalah private, artinya hanya kelas Orang tersebut yang dapat mengakses atribut tersebut, walaupun kelas Pegawai merupakan turunan dari kelas Orang, tetap saja kelas Pegawai tidak dapat mengakses atribut nama dan alamat dari kelas Orang.

```
public class Pegawai extends Orang {

    private String nip;

    public void ubahNip(String nip) {
        this.nip = nip;
    }

    public String ambilNip() {
        return nip;
    }

    public void contoh(){
        String ambilnama = nama;
    }
}
```

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 Orang {

    protected String nama;
    protected String alamat;

    public void ubahNama(String nama) {
        this.nama = nama;
    }

    public String ambilNama() {
        return nama;
    }

    public void ubahAlamat(String alamat) {
        this.alamat = alamat;
    }

    public String ambilAlamat() {
        return alamat;
    }
}
```

```
public class Pegawai extends Orang {

    private String nip;

    public void ubahNip(String nip) {
        this.nip = nip;
    }

    public String ambilNip() {
        return nip;
    }

    public void contoh(){
        // berhasil
        String ambilnama = nama;
        String ambilalamat = alamat;
    }
}
```

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 :(");
    }
}
```

Setelah itu kelas Dewasa merupakan kelas turunan dari kelas Bayi.

```
public class Dewasa extends Bayi{

}
```

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

```
public class Test {

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

}
```

Maka hasilnya adalah “Tidak Bisa :(”, 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.

```
public class Dewasa extends Bayi {
```

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

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

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

```
public class Bernama {

    private String nama;

    public Bernama() {

    }

    public Bernama(String nama) {
        this.nama = nama;
    }

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

```
public class Berumur extends Bernama{

    private int umur;

    public Berumur() {

    }

    public Berumur(String nama, int umur){
        super(nama);
        this.umur = umur;
    }

    public int ambilUmur(){
        return umur;
    }

}
```

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

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

```
public class Contoh {

    private final String data = "Data";
}
```

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

```
public class Contoh {

    private final String data = "Data";

    public void ubahData(String data){
        // error
        this.data = data;
    }
}
```

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.

```
public abstract class Hewan {

    public abstract void bicara();

}
```

```
public class Anjing extends Hewan{

    public void bicara() {
        System.out.println("Gog gog...");
    }

}
```

```
public class Kambing extends Hewan{

    public void bicara() {
        System.out.println("Embe...");
    }

}
```

```
public class Kucing extends 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.

```
public class Test {

    public static void main(String[] args) {
        Hewan hewan1 = new Anjing();
        hewan1.bicara();
```

```
        Hewan hewan2 = new Kambing();
        hewan2.bicara();

        Hewan hewan3 = new 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 menganduk 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 metode-metode 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;
        }
    }
}
```

3.5.1 Anonymous Class

Kelas anonimous merupakan kelas yang dideklarasikan tanda nama, biasanya kelas ini dibuat ketika mendeklarasikan sebuah variabel. Contoh

```
package aplikasi.program;

import aplikasi.prototype.Pukulan;

public class Program {

    public static void main(String[] args) {

        Pukulan pukulan = new Pukulan() {

            public void pukul() {
                System.out.println("Pukul ah...");
            }
        };

        pukulan.pukul();
    }
}
```

Sekilas pada kode diatas, kita hanya memiliki kelas Program. Padahal sebenarnya terdapat dua kelas, yaitu dengan kelas Anonymous yang merupakan turunan dari interface Pukulan. Penggunaan kelas anonimous ini biasa dilakukan ketika kita hanya akan membuat sebuah kelas yang hanya sekali pakai saja.

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
    }
}
```

3.6 Kelas POJO / Java Bean

Kelas POJO atau Java Bean merupakan kelas dimana sebuah kelas memiliki atribut dan memiliki metode getter dan setter. Dimana atributnya bersifat private dan metode getter dan setter nya bersifat public. Metode getter digunakan untuk mendapatkan nilai atribut tersebut, sedangkan metode setter digunakan untuk mengubah nilai atribut.

Penamaan kelas POJO mirip seperti punuk unta. Misal :

Nama Atribut	Penamaan
Nama Depan	namaDepan
Nama Belakang	namaBelakang
Alamat	Alamat
Tanggal Lahir	tanggalLahir
Contoh Atribut Panjang Sekali	contohAtributPanjangSekali

Sedangkan untuk penamaan getter dan setternya pun mirip seperti punuk unta, misal :

Atribut	Getter	Setter
namaDepan	getNamaDepan	setNamaDepan
alamat	getAlamat	setAlamat
tanggalLahir	getTanggalLahir	setTanggalLahir

Namun ada beberapa pengecualian, jika tipe atributnya adalah boolean, maka getter nya bisa diganti dari get menjadi is. Sehingga seperti ini.

Atribut	Tipe Data	Getter
sudahMenikah	boolean	isSudahMenikah
nama	*semua tipe*	getNama

Contoh kelas POJO Mahasiswa yang memiliki nim, nama, dan alamat.

```
package aplikasi.data;
```

```
public class Mahasiswa {

    private String nim;
    private String nama;
    private String alamat;

    public String getAlamat() {
        return alamat;
    }

    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }

    public String getNama() {
        return nama;
    }

    public void setNama(String nama) {
        this.nama = nama;
    }

    public String getNim() {
        return nim;
    }

    public void setNim(String nim) {
        this.nim = nim;
    }
}
```

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 keluaran “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.

5 Kelas – Kelas

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. karena 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);

    }
}
```

Tentang Penulis

Penulis bernama **Eko Kurniawan Khannedy**, atau sering dipanggil *Usu*. Lahir di kota Bekasi tanggal 29 Desember 1988, dan besar di kota Subang. Penulis adalah Siswa lulusan SMA Negeri 1 Subang, dan saat ini sedang menempuh kuliah di Universitas Komputer Indonesia, jurusan Teknik Informatika angkatan tahun 2006.

Penulis aktif di berbagai komunitas teknologi, seperti *Java User Group Bandung*, *OpenSource University Meetup* dan *Unikom Programming Team*. Saat ini penulis menjabat sebagai **Leader** di Java User Group Bandung, Unikom Programming Team dan OpenSource University Meetup Unikom.

Penulis dapat dihubungi di :

- +6285292775999
- echo.khannedy@gmail.com
- http://eecchhoo.wordpress.com/
- http://twitter.com/echo_khannedy
- http://facebook.com/khannedy

:D