Pemrograman Berorientasi Obyek

Class dan Obyek 2

anton@ukdw.ac.id

Method **main** pada Java

- **public static void main(String[] args)**
- Merupakan bagian yang dieksekusi oleh program Java
- Sifat: public, static, void
- **Public**: dapat diakses dari class manapun
- **Static**: tidak perlu dibuat instance nya terlebih dahulu, bisa langsung dieksekusi
- **Void**: tidak mengembalikan nilai

Argumen pada method **main**

- **(String[] args)**
- Digunakan untuk menerima parameter dari program Java melalui command prompt
- Ditampung dalam array of String
- Cara mengakses:
  - args[0], args[1], args[2]….
  - Dapat juga menggunakan args.length
- Boleh diganti asal array of String!

```
java.lang.NoSuchMethodError: main
Exception in thread "main"
```

args[]

- Berupa array 1 dimensi bertipe String
- Cara pengaksesan:
  - Looping:

    for(int i=0;i<args.length();i++)
  - Langsung akses ke indexnya

    args[1], dst...

Demo

Object Initialization

- Obyek diciptakan berdasarkan definisi **class**
- Setiap obyek membutuhkan **memory**
- Proses pembuatan obyek :
  - JVM memesan dan menyiapkan memory untuk obyek tersebut
  - Obyek tersebut diinisialisasi
  - Obyek tersebut siap digunakan
  - Jika sudah tidak digunakan, obyek akan dihapus dari memory

Object Initialization

- Kenapa obyek harus **diinisialisasi** ?
  - Untuk mempersiapkan nilai default / awal dari atribut-atribut suatu obyek
- Pada Java, ada 2 cara untuk melakukan inisialiasi :
  - Initializer:
    - Static initializer
    - Instance initializer
  - Constructor

Konstruktor

- **Konstruktor** dipanggil pada saat **instansiasi** sebuah object
  - Digunakan untuk melakukan **inisialisasi**
- Melakukan **instansiasi** adalah mengalokasikan sejumlah **memory** dari komputer untuk sesuatu kebutuhan **struktur data** yang digunakan, jika ditambah konstruktor maka ditambah **inisialisasi**
- Nama konstruktor pada Java namanya **harus sama** dengan nama classnya
- Konstruktor pada Java bersifat **opsional**
- Kata kunci **super()**, bisa digunakan untuk memanggil konstruktor yang ada pada superclass

Konstruktor

- Mempunyai karakteristik seperti **method**, dapat mempunyai **parameter** dan dapat mengakses **variabel** class
- Namun ada perbedaan prinsip antara konstruktor dan method yaitu :
  - Nama konsturktor **PASTI** sama dengan nama **Kelas**.
  - Konstruktor tidak pernah mempunyai tipe data, ataupun **void**
  - Konstruktor **tidak bisa** mengembalikan nilai, tidak boleh ada statement **return**.
  - Hanya dieksekusi (dipanggil) saat **instansiasi** saja.

Konstruktor

- Dalam suatu class **pasti** mempunyai konstruktor, jika konstruktor tidak dibuat dalam suatu class, maka kompiler Java secara otomatis akan membuat konstuktor kosong (***null constructor***)
- Jika dalam suatu class sudah ada **minimal satu** konstruktor, maka konstruktor otomatis tidak dibuat oleh kompiler **Java**
- Suatu class dapat mempunyai konstruktor lebih dari satu, dengan syarat tiap-tiap konstruktor harus mempunyai ***signature*** yang berbeda satu dengan yang lain

Konstruktor

- Yang dimaksud dengan ***signature*** adalah bentuk parameter **formal** konstruktor tersebut, yaitu :
  - banyaknya parameter formal
  - tipe-tipe data dari tiap parameter formal
  - serta urutannya letak parameter formalnya.
- Adanya lebih dari satu konstruktor ini disebut dengan **overloading konstruktor**
- Programmer dapat memilih konstruktor yang ingin digunakannya, demikian juga Java akan mengeksekusi konstruktor yang sesuai pilihan

Konstruktor

- Contoh Constructor:

```
class Mobil{
	int jumRoda;

	public Mobil(int jumRoda){
		this.jumRoda = jumRoda;
	}
}
```

Konstruktor

- Contoh Overloading Constructor:

```
class Mobil{
      int jumRoda;

      public Mobil(int jumRoda){
          this.jumRoda = jumRoda;
      }
      public Mobil(){
          System.out.println(“mobil baru”);
      }
}
```

Static initializer

- Dilakukan menggunakan static method dan static variabel

```
public class MyClass {

   public static int number = 10;
   public static int getNumber() {
      return number;
   }

   public static void main(String[] args){
      System.out.println (MyClass.getNumber());
   }
}
```

```
10

Process completed.
```

Static initializer

- Karena variabel number bersifat static, maka tidak bisa diinisialisasi menggunakan konstruktor, jadi:

```
public class MyClass {

  public static int number;

  static {
    number = 10;
  }

  public static int getNumber() {
   return number;
  }

  public static void main (String[] args) {
     System.out.println (MyClass.getNumber());
  }
}
```

```
10

Process completed.
```

Instance Initialization

- Ditulis dibaris awal sebelum konstruktor yang sebenarnya

```
public class MyClass {

 {
  System.out.println("I'm the instance initalizer.");
 }

 public MyClass() {
  System.out.println("I'm the constructor!");
 }

 public static void main (String[] args) {
    MyClass m = new MyClass();
 }
}
```

```
I'm the instance initalizer.
I'm the constructor!
```

Kata kunci “this”

- Dipergunakan pada sebuah class dan digunakan untuk menyatakan **class sekarang / kelas itu**

```
public class Mahasiswa{
    String nim;
    String nama;
    float ipk;

    void cetakNIMdanNama(){
        System.out.println ("NIM : " + this.nim);
        System.out.println ("Nama : " + this.nama);
    }

    void setNama(int nm){
        this.nama = nm;
    }
}
```

Lingkup variabel

Dipergunakan juga untuk membedakan ruang lingkup variabel class dan variabel pada method

```
public class Lingkup{
    String warna = "Merah";
    void infoLingkup(){
        String warna = "Biru";
        System.out.println ("Warna pada metode : " + warna);
        System.out.println ("Warna milik kelas : " + this.warna);
    }
    public static void main (String[] args) {
        Lingkup l = new Lingkup();
        l.infoLingkup();
    }
}
```

Instance vs Local Variable

- Instance Variable / Class Variabel

```
class Kuda {
   private double tinggi = 15.2;
   private String jenis;
}
```

- Local Variable / Method Variabel

```
class Contoh2 {
   int a;
   int b = 12;
   public int tambah() {
      int total = a+b;
      return total;
   }
}
```

Review: Enkapsulasi

- Information Hiding
- Pengaturan hak akses atas bagian-bagian dari suatu object
- Hak akses terhadap bagian-bagian dari object
  - Class
  - Atribut / Variabel
  - Method / Behavior
- Haks akses disebut dengan **modifier**

Modifier

- Modifier diletakkan pada anatomi kelas, sifatnya **optional**, digunakan berdasarkan kebutuhan.
- Ada beberapa keyword yang digunakan sebagai modifier dan dikelompokkan menjadi :
  - Modifier akses (*public*, *protected*, *default*, *private*)
  - Modifier *final*
  - Modifier *static*
  - Modifier *abstract*

Modifier akses

- Modifier akses digunakan untuk **membatasi** akses class lain terhadap suatu bagian dari class. (attributes, methods, ataupun class itu sendiri)

Modifier akses

Wilayah Akses	***public***	***protected***	***default***	***private***
Di kelas yg sama	**√**	**√**	**√**	**√**
Beda kelas, di package/folder yg sama	**√**	**√**	**√**	**x**
Beda kelas, beda package/folder, di kelas turunan	**√**	**√**	x	x
Beda kelas, beda package/folder, tidak di kelas turunan	**√**	**x**	x	x

Ingat!

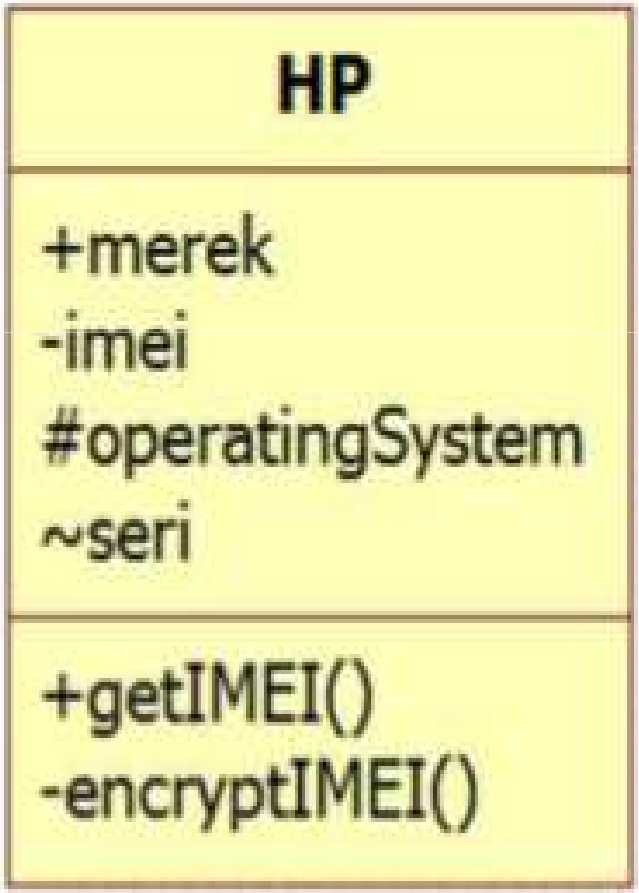
- Jika tidak disebutkan akses modifiernya berarti **default**, sifatnya:
  - Dapat diakses pada kelas itu
  - Dapat diakses pada kelas yang sama pada paket (package/directory) yang sama

Contoh pada Class

```
public class Tabungan {
        public String nama;
        private int saldo;
        protected double bunga;
        int nomorRekening;
        ...
}
```

- Jika class dibuat public, maka nama file harus sama dengan nama classnya, karena file dianggap class
- Jika tidak dibuat public tidak harus sama

Modifier akses pada Class diagram



+ public

- private

 # protected

~ default

Modifier final

- Untuk membuat sebuah variabel class/method menjadi **konstanta**.
- Kalau modifier final diberikan pada suatu **class**, maka class tersebut tidak bisa diturunkan/diwariskan.
- Jika diberikan pada suatu **method**, maka method tersebut tidak bisa dioverride di class anaknya

Contoh variabel final

```
class MyClass {
 final int x = 10;
 public void setX(int x){
    this.x = x;
 }
 public int getX(){
    return x;
 }
 public static void main (String[] args) {
    MyClass myc = new MyClass();
    myc.setX(15);
    System.out.println (myc.getX());
 }
}
```

```
J:\arc\dosen\pbo\pertemuan3\MyClass.java:13: cannot assign a value to final variable x
    this.x = x;
        ^
1 error
```

Contoh class final

```
final class MyClass {

 public static void main (String[] args) {
    System.out.println ("hallo");
 }
}

class Coba extends MyClass{

}
```

```
J:\arc\dosen\pbo\pertemuan3\MyClass.java:17: cannot inherit from final MyCla
class Coba extends MyClass{
                   ^
1 error
```

Contoh method final

```
class MyClass {
 int x = 10;
 public final void setX(int x){
     this.x = x;
 }
 public int getX(){
     return x;
 }
}

class Coba extends MyClass{
 public void setX(int x){

 }
}
```

```
J:\arc\dosen\pbo\pertemuan3\MyClass.java:21: setX(int) in Coba cannot override setX(int) in MyClass; overridden method is final
 public void setX(int x){
             ^
1 error
```

Modifier static

- Modifier **static** digunakan untuk membuat sebuah method/attribute bisa diakses *tanpa melakukan instansiasi terlebih dulu*.
- Contoh *System.out.println()* bersifat static artinya untuk memanggil method **println()** tidak harus dilakukan instansiasi dari kelas **System**.
- Nilai attribute yang **static** akan **sama** untuk semua objek.

Modifier static

```
class MyClass
{
    static int varStatic = 2;
    int VarNonStatic;

    public static void doIt(){
        System.out.println("Ini metode statis");
    }

    public static void main(String[] args)
    {
        MyClass.varStatic++;
        System.out.println("Nilai :" + MyClass.varStatic);

        MyClass ObjStatic = new MyClass();
        ObjStatic.varStatic++;

        System.out.println("Nilai :" + MyClass.varStatic);
        MyClass.doIt();
    }
}
```

```
Nilai :3
Nilai :4
Ini metode statis
```

Static

- Kelebihan:
  - Mudah digunakan
  - Tidak perlu instansiasi obyek
  - Dapat digunakan bersama-sama (dishare) dengan obyek lain
- Kekurangan:
  - Boros memory
  - Tidak bersifat private untuk masing-masing obyek

Tipe data Primitif pada Java

byte	8-bit signed, -128 sampai 127
short	16-bit signed, -32768 sampai 32767
int	32-bit signed, -2147483648 sampai 2147483647
long	64-bit signed, -9,223,372,036,854,775,808 sampai 9,223,372,036,854,775,807
float	32-bit IEEE 754 floating point
double	64-bit IEEE 754 floating point
boolean	true / false
char	16-bit Unicode character, '\u0000' (or 0) sampai '\uffff' (65,535 karakter)

Keywords

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used
** added in 1.2
*** added in 1.4
**** added in 5.0

Membaca data dari Keyboard

- Sejak versi 1.6x keatas:

  Gunakan:

  System.console().readLine();
- Fungsi diatas menerima inputan dari pengguna bertipe data String sehingga harus ditampung terlebih dahulu ke variabel bertipe String
- Contoh:
- String nama = System.console().readLine(“Masukkan nama:”);

Konversi Tipe Data

- Secara default semua inputan dari pengguna adalah bertipe **String**
- Untuk perhitungan harus dilakukan **konversi** tipe data
- X to String
  - int to String:

    int aInt = 1;
    String aString = Integer.toString(aInt);
- String to X
  - String to int:

    String aString = "78";
    int aInt = Integer.parseInt(aString);
  - Integer.valueOf(<*nilaiString*>).intValue();

Konversi Tipe Data

- Secara default semua inputan dari pengguna adalah bertipe **String**
- Untuk perhitungan harus dilakukan **konversi** tipe data
- X to String
  - int to String:

    int aInt = 1;
    String aString = Integer.toString(aInt);
- String to X
  - String to int:

    String aString = "78";
    int aInt = Integer.parseInt(aString);
  - Integer.valueOf(<*nilaiString*>).intValue();

Konversi

- Java akan melakukan konversi otomatis dengan ketentuan sbb.:
  - Apapun ditambahkan dengan tipe String menjadi String
  - Tipe bulat dioperasikan dengan tipe bulat menghasilkan tipe bulat dengan membuang nilai desimalnya
  - Jika dalam suatu ekspresi, terdapat tipe dengan kapasitas lebih tinggi, maka ekspresi akan menghasilkan nilai dengan tipe data dengan kapasitas tertinggi

Demo

NEXT

- Atribut dan Method