# Temu05

## Constructor, Encapsulation & Information Hiding

**Ali Tarmuji, S.T., M.Cs.**
alitarmuji@tif.uad.ac.id

# Bahan Diskusi

- Constructor
- Package
- Enum
- Encapsulation
  - Information Hiding
  - Java Beans
  - Inner Class

# Konstruktor

- Konstruktor merupakan prosedur yang diakses secara langsung ketika membuat sebuah objek.
- Konstruktor adalah sebuat method semu,
- Nama konstruktor harus sama dengan nama kelas.
- Dapat difungsikan untuk menginisiasi (nilai awal) data member
- Konstruktor dapat memiliki parameter dan juga dapat di *overloading*.
- Tidak mempunyai modifier (seperti void, int, double, dll)

# Membuat Konstruktor

```
class NamaKelas {

    NamaKelas(){
    }

    // overloading konstruktor
    NamaKelas(TipeData parameter){
    }

}
```

# Menggunakan Konstruktor

- *NamaKelas objek1 = new NamaKelas();*

- *NamaKelas objek2 = new NamaKelas( “parameter ” );*

# Overloading constructor

Suatu class dapat mempunyai lebih dari 1 konstruktor dengan syarat daftar parameternya tidak boleh ada yang sama.

```
public class Siswa {
        private int nrp;

        public Siswa() {
                nrp=0;
        }

        public Siswa(int n) {
                nrp=n;
        }
}
```

# Package/ paket

- Paket merupakan fitur yang dapat digunakan untuk memanajemen file Kelas.
- Biasanya kelas-kelas yang sejenis dikumpulkan dalam sebuah paket, misal kelas-kelas data disimpan dalam paket data, kelas-kelas form disimpan dalam paket GUI, dan sebagainya.
- Package akan sangat bermanfaat jika class-class yang kita buat sangat banyak sehingga perlu berdasarkan kategori tertentu.

# Membuat Paket

- Membuat paket dilakukan pada kelas, dan dapat menggunakan perintah **package** diikuti dengan nama paket nya.

- ***package namapaket;***

- *class NamaKelas {*
- 
- *}*

# Paket Bertingkat

- Sebuah paket dapat dibuat didalam paket lagi dengan menggunakan tanda . (titik).
- ***package paket.subpaket;***
- *class NamaKelas {*
- 
- *}*

# Mengakses Kelas dari Luar Paket

- Secara default, sebuah kelas tidak dapat menggunakan kelas yang berada dalam luar paket.
- Untuk menggunakan kelas di luar paket, perlu menggunakan perintah import diikuti dengan nama paket dan kelasnya.

- ***import namapaket.subpaket.NamaKelas;***

- *NamaKelas objek = new NamaKelas();*

# Visibility

- Visibility merupakan kemampuan untuk mengontrol sebuah akses ke **kelas**, **atribut**, **prosedur** dan **fungsi**.
- Java mendukung beberapa visibility, yaitu :
  - Default
  - Public
  - Private, dan
  - Protected

# Hak Akses Visibility

| Visibility | Kelas | Paket | Sub Kelas | Umum |
|---|---|---|---|---|
| Private | Y | | | |
| Default | Y | Y | | |
| Protected | Y | Y | Y | |
| Public | Y | Y | Y | Y |

# Membuat Public

- *public class NamaKelas {*
- *public TipeData namaAtribut;*
- *public void namaProsedur(){*
- *}*
- *public TipeData namaFungsi(){*
- *return hasil;*
- *}*
- *}*

# Membuat Private

- ***public*** *class NamaKelas {*
- 	*private TipeData namaAtribut;*
- 	*private void namaProsedur(){*
- 	*}*
- 	*private TipeData namaFungsi(){*
- 		*return hasil;*
- 	*}*
- *}*

# Kata Kunci static

- Kata kunci static dapat digunakan untuk membuat sebuah atribut, fungsi dan prosedur dapat diakses secara langsung tanpa membuat objek.
- Atribut, fungsi dan prosedur yang telah diberi kata kunci static, tidak akan dapat mengakses data pada kelas tersebut secara langsung.

# Membuat static

- *public class NamaKelas {*
- *public **static** TipeData namaAtribut;*
- *public **static** void namaProsedurStatic(){*
- *}*
- *public **static** TipeData namaFungsiStatic(){*
- *return hasil;*
- *}*
- *}*

# Enum

- Enum merupakan jenis kelas yang hanya memiliki atribut dia sendiri.
- Enum digunakan sebagai pengganti konstanta.

# Membuat Enum

- *public* ***enum*** *NamaEnum{*
- 
- *ENUM_SATU,*
- *ENUM_DUA,*
- *ENUM_TIGA,*
- *ENUM_EMPAT*
- 
- *}*

# Menggunakan Enum (1)

- public class NamaKelas {
- public NamaEnum atribut;
- }

# Menggunakan Enum (2)

- *NamaKelas objek = new NamaKelas();*

- *// mengubah atribut enum*
- *objek.atribut = NamaEnum.ENUM_SATU;*

# Menggunakan static

- *// atribut static*
- *NamaKelas.namaAtributStatic;*

- *// prosedur static*
- *NamaKelas.namaProsedurStatic();*

- *// fungsi static*
- *TipeData hasil = NamaKelas.namaFungsiStatic();*

# Enkapsulasi

- Merupakan suatu cara untuk menyembunyikan implementasi detail dari suatu class.
- Dua hal mendasar yang ada di enkapsulasi:
  - Information hiding

    → Memastikan atribut objek tidak dapat diubah dan diakses secara langung.

  - Interface to acces data

    →Hanya prosedur dan fungsi tertentu yang diberi izin untuk mengakses dan mengubah atribut objek sebagai perantara

# Enkapsulasi

- Misal : NRP dari siswa-siswa IT2 : range 1-10.
- Jika NRP tidak dienkapsulasi :
  - Siswa dapat memasukkan sembarang nilai, sehingga perlu melakukan penyembunyian informasi (nformation hiding) thd atribut nrp, sehingga nrp tidak bisa diakses secara langsung.

- Lalu, kalau atribut nrp tersebut disembunyikan, bagaimana cara mengakses atribut nrp itu untuk memberikan atau mengubah nilai?.
  - Perlu suatu interface untuk mengakses data, yang berupa method dimana di dalamnya terdapat implementasi untuk mengakses data nrp.

# Contoh Enkapsulasi

```
public class Orang {

    private String nama;

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

    public String tanyaNama(){
        return this.nama;
    }

}
```

# Java Beans

- Java Beans merupakan implementasi dari enkapsulasi.
- Dalam Java Beans semua atribut dibuat menjadi **private.**
- Untuk mengakses atribut dibuat fungsi **getNamaAtribut**.
- Untuk mengubah atribut dibuat prosedur **setNamaAtribut**.
- Untuk nilai **boolean**, fungsi yang dibuat adalah **isNamaBoolean**.

# Contoh Java Beans (1)

```
public class Orang {
    private String nama;

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

# Contoh Java Beans (2)

```
public class Orang {
    private boolean menikah;

    public boolean isMenikah(){
        return this.menikah;
    }

    public void setMenikah(boolean menikah){
        this.menikah = menikah;
    }
}
```

# Inner Class

- Inner Class merupakan kelas yang terdapat dalam sebuah kelas.
- Cara membuat inner class sama seperti membuat kelas biasa, yang membedakan adalah kelas tersebut berada dalam blok kelas yang sudah ada.

# Membuat Inner Class

- *public class NamaKelas {*
- *// isi kelas*
- *public class NamaInnerClass{*
- *// isi inner class*
- *}*
- *}*

# Menggunakan Inner Class

- NamaKelas o1 = new NamaKelas();

- NamaInnerClass o2 = o1.new NamaInnerClass()

# Static Inner Class

- Kata kunci static dapat juga digunakan untuk sebuah inner class.
- Dengan menggunakan kata kunci static, maka inner class tidak akan bergantung lagi pada *outer class*.

# Membuat Static Inner Class

- *public class NamaKelas {*
- *// isi kelas*
- *public **static** class NamaInnerClass{*
- *// isi inner class*
- *}*
- *}*

# Menggunakan Static Inner Class

- *NamaKelas.NamaInnerClass objek =*
- *new NamaKelas.NamaInnerClass();*