

Pemrograman Berbasis Komponen

Antonius Rachmat C, S.Kom, M.Cs

Yuan Lukito, S.Kom

Info Matakuliah

- SKS: 3
- Tidak berpraktikum
- Web Lecturer:
 - <http://lecturer.ukdw.ac.id/anton>
 - <http://lecturer.ukdw.ac.id/yuan>
- Email:
 - anton@ti.ukdw.ac.id
 - yuanlukito@gmail.com

Waktu Kuliah

- Jumat 13.30 (A)
- Di ruang : B.33

Deskripsi

- Matakuliah ini berisi tentang:
 - pemrograman menggunakan pendekatan berorientasi obyek dan berbasis komponen
 - cara menggunakan design patterns (pola desain) sehingga bersifat reusable dan dapat diterapkan pada dan menghasilkan komponen-komponen siap pakai seperti pada:
 - J2EE (JavaBeans dan Enterprise JavaBeans)
 - Microsoft (.NET Component)
 - Delphi (VCL)
 - Web dan browser component

Tujuan Matakuliah

- Mahasiswa mampu:
 - Menjelaskan konsep tentang Component Oriented Programming (COP)
 - Menjelaskan dan menggunakan design patterns seperti observer, decorator, singleton, command, adapter, state dan proxy
 - Menjelaskan dan menggunakan desktop component:
 - Delphi VCL, .NET Component, JavaBeans & EJB
 - Menjelaskan dan menggunakan web component & API:
 - Facebook API, Twitter, Plurk, dan WS lainnya

Silabus

- silabus, review OOP dan pengantar COP - 20/1
- pengantar DP, strategy & observer pattern - 27/1
- factory & singleton pattern - 3/2
- command pattern - 10/2
- PRESENTASI IDE PROJECT + LAPORAN - 17/2
- adapter pattern & MVC pattern - 24/2
- state & decorator pattern - 2/3
- TTS

Silabus

- Web component Introduction + PHP OOP – VCL 30/3
- PRESENTASI PROGRESS PROJECT - 31/3
- desktop component: delphi - 13/4
- desktop component: .net - dll dan activeX - 20/4
- Java component: Java Beans dan EJB - 27/4
- Mobile Component - 4/5
- PRESENTASI FINAL - 11/5
- Masa TAS

Komponen Penilaian

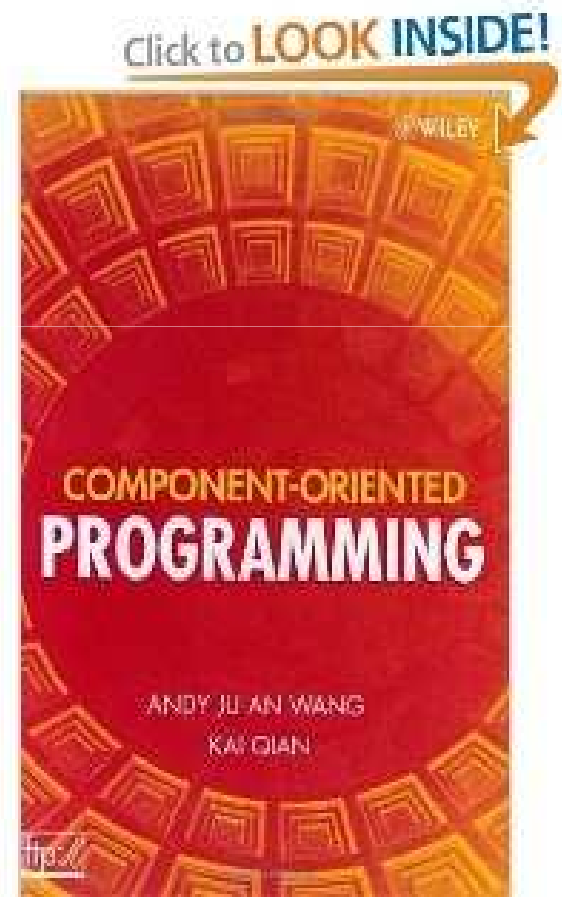
- TTS 20%
- TAS 20%
- Project 60%
 - Ide 20%
 - Progress 20%
 - Final 20%

Penilaian

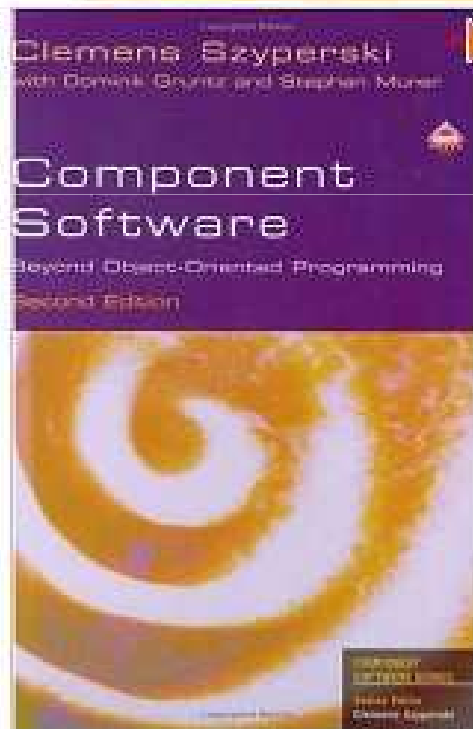
- 85.0 - 100 A 4.0
- 80.0 - 84.9 A- 3.7
- 75.0 - 79.9 B+ 3.3
- 70.0 – 74.9 B 3.0
- 65.0 – 69.9 B- 2.7
- 60.0 – 64.9 C+ 2.3
- 55.0 – 59.9 C 2.0
- 0 – 54.9 E 0.0
- -- F 0.0

Acuan

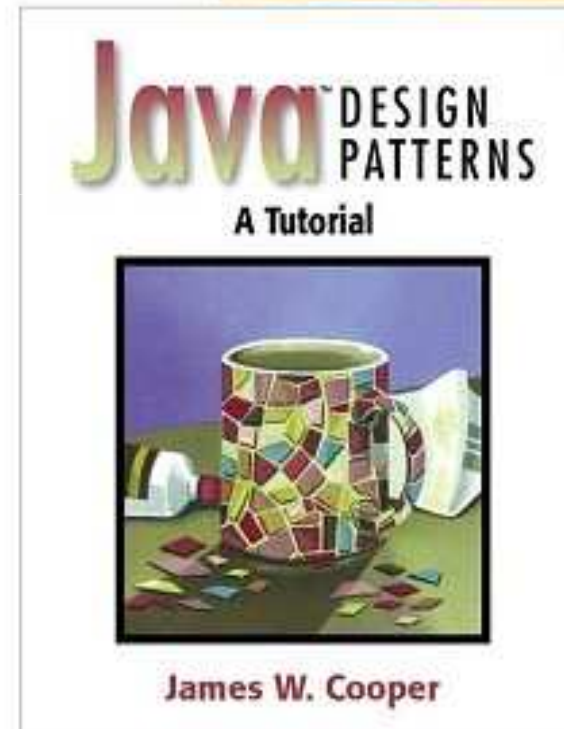
- **Head First Design Pattern** by Eric Freeman dan Elisabeth Freeman, O'Reilly, 2004
- **Component Oriented Programming**, by Andy Ju An Wang dan Kai Qian, Willey, 2005
- **Head First Enterprise Java Beans**, O'Reilly
- **Component Software, Beyond Object Oriented Programming**, Clemens Szyperski, Addison-Wesley Professional; 2 edition, 2002
- **Design Patterns Java Companion** by James W. Cooper, Addison-Wesley, 1998



Click to **LOOK INSIDE!**



Click to **LOOK INSIDE!**



Project + Presentasi

- 1 kelompok = 4 orang
- Nilai Total : 60%
- Terdiri dari 3 tahap:
 - Presentasi Ide : 20%
 - Presentasi Progress : 20%
 - Presentasi Final : 20%

Laporan Project

- Laporan Ide:
 - Komponen yang akan dibuat
 - Tools yang digunakan
 - Arsitektur komponen
 - Fitur komponen
 - Database (opsional)
 - Desain Interface
- Laporan Progress:
 - Prototype komponen
 - Prototype aplikasi
 - Fitur yang sudah jadi

Laporan Project

- Laporan Final:
 - Seluruh laporan yang sudah final
 - Komponen dan aplikasi final

Project yang sudah...

- ImageResizer (PHP Component)
- FileManipulator (PHP Component)
- CustomProgressbar (.NET Component)
- MobileRSSReader (J2ME)
- CustomButton (Delphi VCL)
- TwitURL (Firefox Component)
- Reminder (.Net Component)
- DotMatrix (.Net Component)
- DatePicker (Java Component)
- Clock (.Net Component)
- Mathematical & Image Captch (PHP Component)

Ide Project tahun ini

- Menggunakan data yang ada di Internet:
 - Web 21, Alkitab / Renungan Harian, Jadwal penerbangan, Data siaran TV berlangganan
- Dibungkus melalui web services
- Dibuat komponennya (versi desktop / mobile)
- Dibuat juga aplikasi yang menggunakannya (versi desktop / mobile)

Contoh Ide Project lain

- CommandExecuter Component (Desktop)
- Artist, Lyrics& Cover Finder (Dekstop)
- Anda punya ide sendiri? Silahkan dikonsultasikan 😊

Aturan

- Aturan absensi mengikuti biro 1
- Saling menghormati, menghargai
- Sopan dan rapi
- Tidak ada tes kecil, TTS susulan
- Toleransi keterlambatan 15 menit
 - Ada tugas membuat paper bagi yang terlambat lebih dari 15 menit

Remidi

- Hanya untuk mhs dengan nilai dibawah C
- Perbaikan maks nilai = C
- Ingat, perbaikan tidak selalu berhasil!
- Syarat remidi:
 - Hanya untuk mhs yang memenuhi 75% kehadiran dari biro 1
 - Ikut TTS
 - Nilai ≤ 30 (D/E)
- Perbaikan bisa berupa soal teori/tugas/program

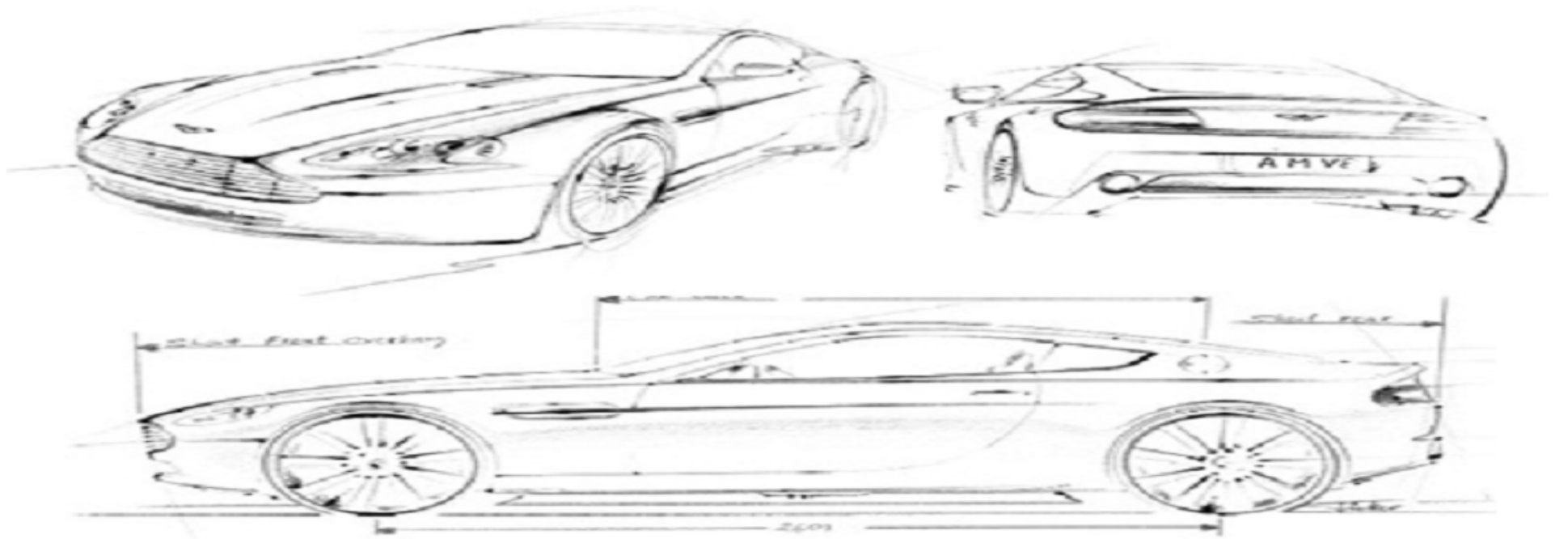
MATERI

BAGIAN 1

OBJECT ORIENTED PROGRAMMING

Pemrograman Berorientasi Obyek

- Suatu metode pemrograman yang dipakai untuk mengorganisasikan program kedalam suatu komponen logis (=class).
- Class harus **diinstansiasi** – dibuat obyeknya terlebih dahulu sebelum bisa digunakan.
 - Misalnya: keyword pada java dan .NET: **new**
 - Inisialisasi awal suatu obyek = **null**
- Class merupakan suatu template yang digunakan sebagai pola desain suatu obyek
 - Analogikan kelas : rancangan mobil
 - Analogikan obyek: mobil nyata



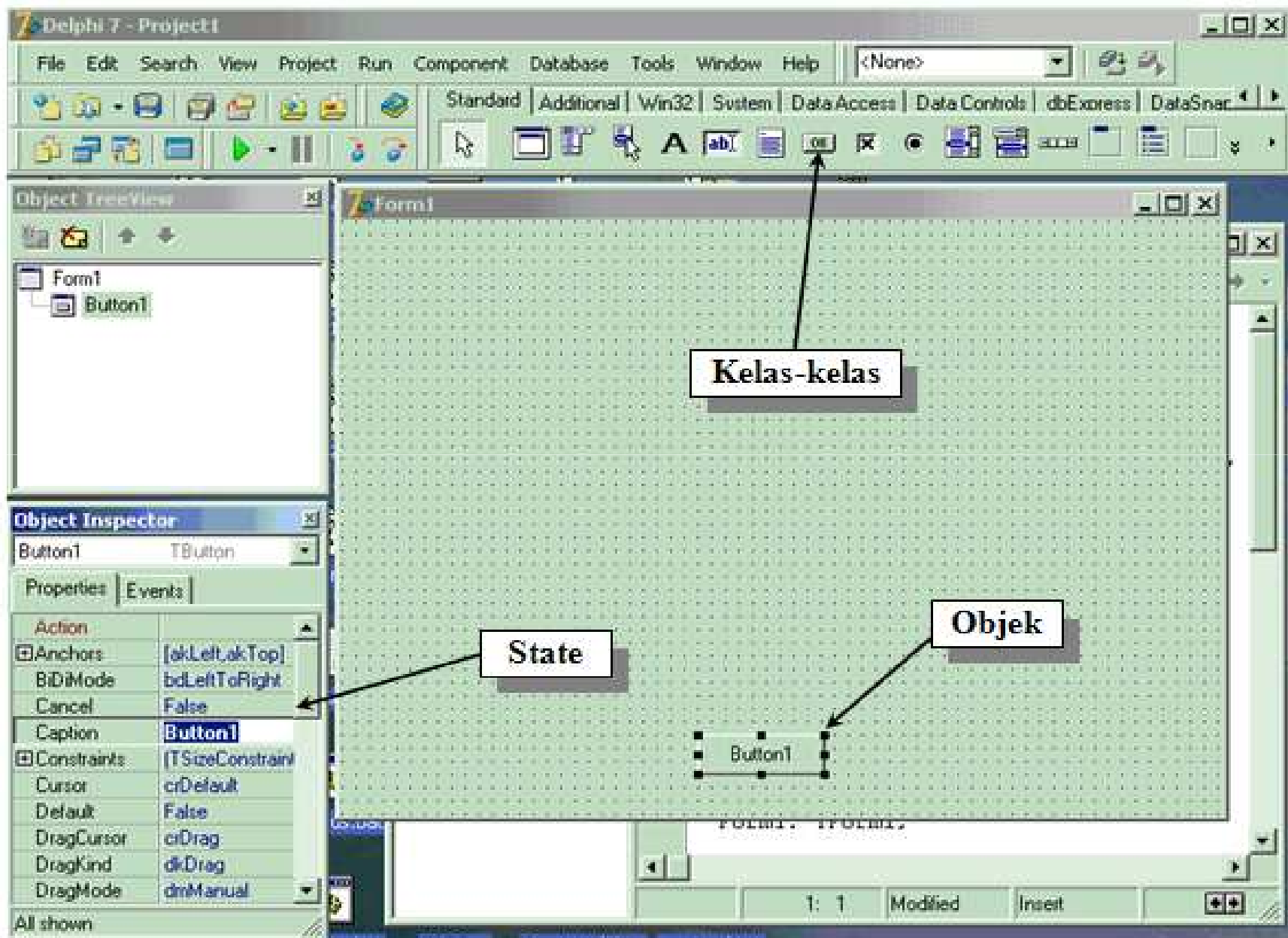
Class dan Obyek

Class mobil		Obyek mobil A	Obyek Mobil B
Variabel Intsance	Nomor Plat	ABC 111	XYZ 123
	Warna	Biru	Merah
	Manufaktur	Mitsubishi	Toyota
	Kecepatan	50 km/h	100 km/h
Method <i>Instance</i>	Method Akselerasi		
	Method Belok		
	Method Rem		

Attribute dan Method

- **Atribut:** Ciri pembeda antar obyek
- **Method:** Tingkah laku sebuah obyek

<i>Obyek</i>	<i>Atribut</i>	<i>Tingkah Laku</i>
Mobil	Tipe dari transmisi manufaktur Warna	Berbelok Mengerem Mempercepat
Singa	Berat Warna Lapar atau tidak lapar Jinak atau liar	roaring Tidur Berburu



Class Template

```
class namaKelas {  
    Fields;  
    Constructors;  
    Methods;  
}
```

Method = Fungsi

- **Pass By Value:** untuk tipe data primitif
 - Mengkopi **nilai** datanya, bukan referensi obyek
 - Hanya bisa memodifikasi data di dalam fungsi, tidak bisa diluar fungsi
- **Pass By References:** untuk tipe data bentukan (ADT) dan obyek
 - Yang dikirimkan adalah **referensi** obyeknya
 - Bisa memodifikasi nilai di dalam dan luar fungsi karena referensi yg diacu memiliki alamat referensiyg sama

```
public class TestPassByValue
{
    public static void main(String[] args)
    {
        int i = 10;
        System.out.println(i);

        test(i);

        System.out.println(i);
    }

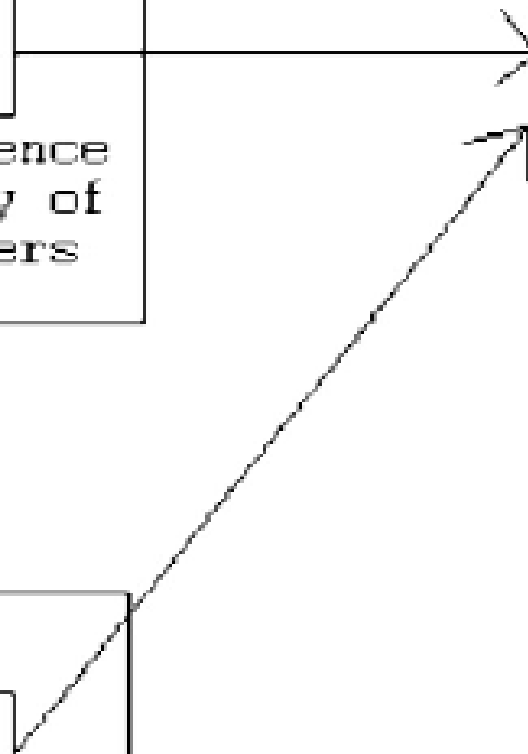
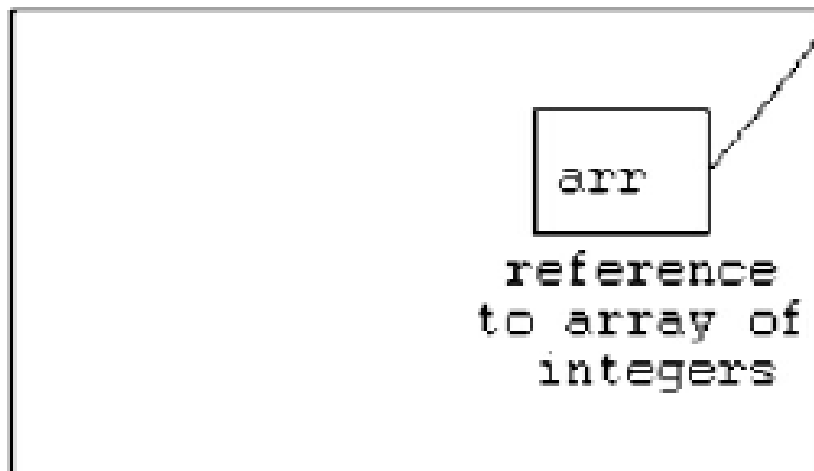
    public static void test(int j)
    {
        j = 33;
    }
}
```

```
public class TestPassByReference {
    public static void main(String[] args) {
        int[] ages = {10, 11, 12};
        for (int i = 0; i < ages.length; i++) {
            System.out.println(ages[i]);
        }
        test(ages);
        for (int i = 0; i < ages.length; i++) {
            System.out.println(ages[i]);
        }
    }
    public static void test(int[] arr) {
        for(int i = 0; i < arr.length; i++) {
            arr[i] = i + 50;
        }
    }
}
```

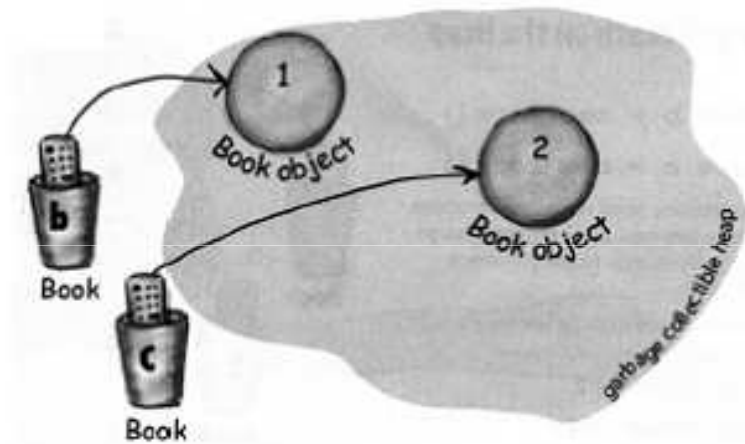
main method



test method



Ilustrasi class dan instansiasi



```
Book b = new Book();
```

```
Book c = new Book();
```

The 2 Book objects are now living on the heap.

References=2

Objects=2

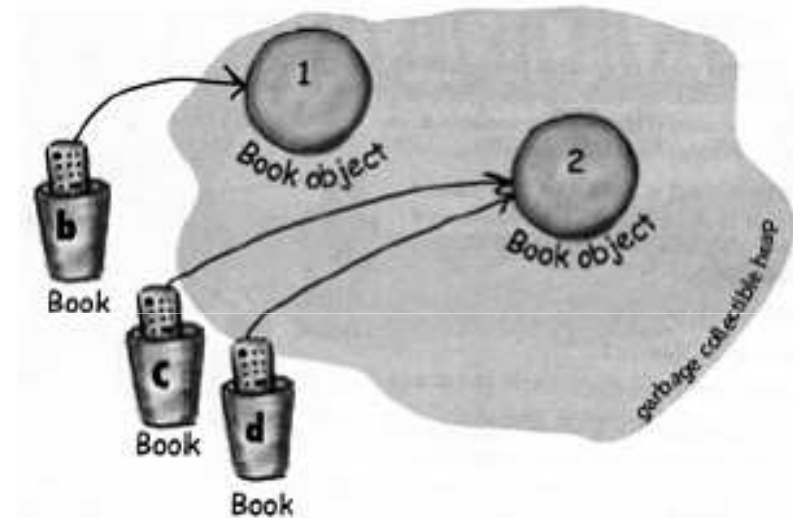
Object/Class assignment

`Book d = c;`

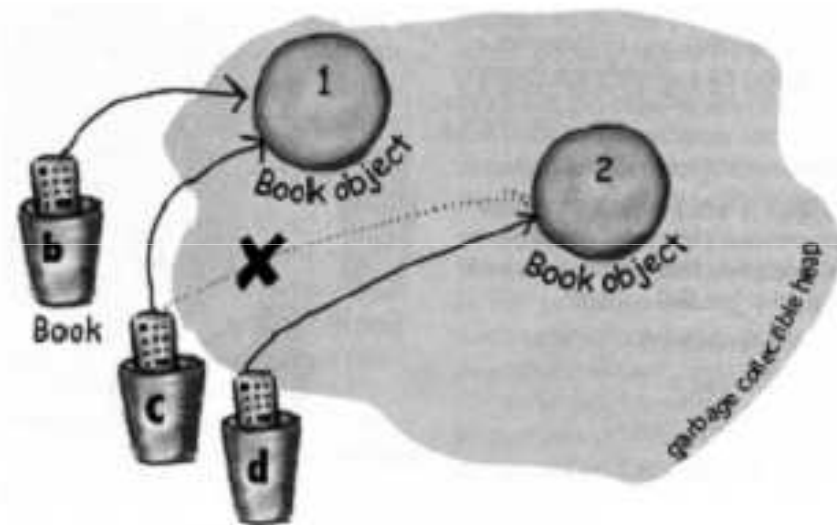
Both c & d refer to the same object.

References=3

Objects=2



Object/Class assignment



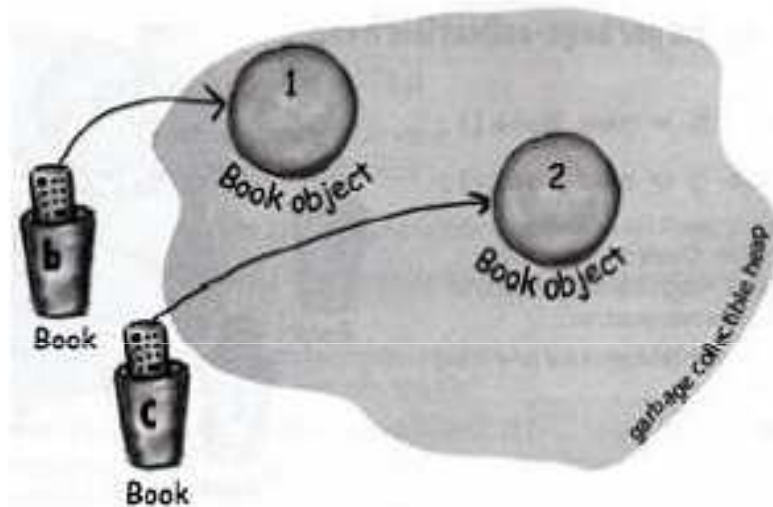
`c = b;`

Both b & c refer to the same object.

References=3

Objects=2

Ilustrasi kerja Garbage Collector



```
Book b = new Book();
```

```
Book c = new Book();
```

The 2 Book objects are now living on the heap.

Active References=2

Reachable Objects=2

Ilustrasi kerja Garbage Collector

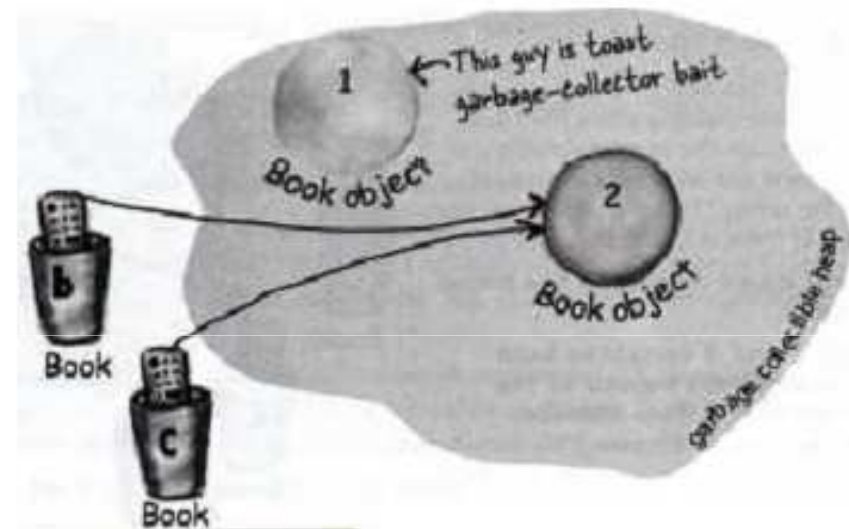
`b = c;`

Both *b* & *c* refer to the same object.
Object 1 is abandoned & eligible for
Garbage Collection (GC).

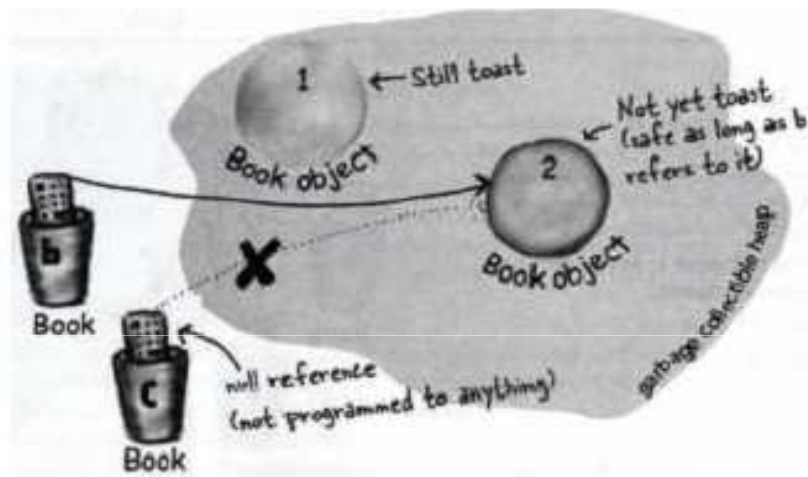
Active References=2

Reachable Objects=1

Abandoned Objects=1



Ilustrasi kerja Garbage Collector



```
c = null;
```

Object 2 still has an active reference (b), and as long as it does, the object is not eligible for GC.

Active References=1

Null References=1

Reachable Objects=1

Abandoned Objects=1

Ciri khas OOP

- **Abstraksi** : Mendefinisikan class yang mampu mengabstraksikan/mendefinisikan suatu obyek dan mampu melakukan kegiatan, mengubah state, dan berkomunikasi dengan obyek lain pada sistem
 - Membuat **class** yg terdiri dari atribut dan method
- **Enkapsulasi** : Menyembunyikan informasi dan detail implementasi sebuah atribut/method, serta mengatur akses terhadap atribut/method
 - Hak akses pada method

Ciri Khas OOP

- **Polimorfisme** : Membuat obyek dari kelas dasar dapat berperilaku seperti obyek lain yang merupakan turunannya
 - Polimorfisme juga berarti banyak bentuk yg diimplementasikan pada multiple constructor class
 - Class anak bisa diassign ke kelas induk
- **Inheritance**: pewarisan atribut dan method dari class induk ke kelas anak

Abstract dan Interface

- **Abstract class**: superclass khusus yang bersifat generik untuk mendefinisikan class yang berisi atribut dan method yang umum
 - Menggunakan keyword: **abstract**
- **Interface** : template yang seluruh methodnya abstract sehingga seluruh methodnya harus diimplementasikan oleh class yang mengimplementasikan interface tersebut.
- Interface boleh memiliki atribut yg berupa **konstanta**
 - **Untuk standarisasi**
 - Konstanta = static final
 - Keyword: interface & implements

Contoh Abstract

```
abstract class Bangun {
    abstract double getLuas();
}

class Lingkaran extends Bangun {
    private int r;

    public Lingkaran(int r) {
        this.r = r;
    }

    double getLuas() {
        return Math.PI*r*r;
    }
}
```

Inheritance

- Merupakan pewarisan atribut-atribut dan method-method dari dari sebuah sebuah class ke class lainnya.
- Class yang memberi warisan => **superclass**
- Class yang diberi warisan => **subclass**
- Contoh:
 - Superclass => sepeda
 - Subclass => sepeda gunung, sepeda balap, sepeda motor
 - Keyword pada Java = **extends**

Inheritance

- Keuntungan:
 - Memberikan **ciri khas** pada masing-masing subclass
 - Superclass mewariskan atribut dan methodnya ke subclass sehingga menerapkan **reuse**
- Pada inheritance juga dikenal adanya **overriding**
 - Method yang sama nama dan tipenya tapi di kelas **berbeda** namun masih dalam satu hubungan keturunan
 - Jika ada method di kelas parent yang sudah didefinisikan, dan didefinisikan ulang, maka method pada kelas anak akan *menutup* method parent, kecuali method pada parent dibuat **final**

Interface

```
interface Draw {  
    void draw();  
    void draw3D();  
}
```

- Suatu class boleh mengimplementasikan **lebih dari satu** interface
- Suatu class hanya boleh mengextends **satu** class

```
interface Color {  
    void setColor(int color);  
    int getColor();  
}
```

```
class Lingkaran implements Draw, Color {  
    void draw() { /*implementasi draw*/ }  
    void draw3D() { /*implementasi draw3D*/ }  
    void setColor(int color);  
    int getColor();  
}
```

Polimorfisme

- **Konstruktor** = method untuk inisialisasi obyek suatu class
- Overloading konstruktor:
 - Dalam 1 kelas lebih dari satu konstruktor
- Overloading method:
 - `public void println(char c);`
 - `public void println(String s);`
 - `public void println(int i);`

Polimorfisme

- Sebuah object mampu melihat **konteks** berdasarkan **reference** yang ditunjuknya
 - Lingkaran dan Kubus diturunkan dari Bangun
 - Bangun memiliki method luas yang mengembalikan double yang masih **abstrak**, tergantung apakah luas Kubus atau Lingkaran
 - Jika sebuah variabel (reference) bertipe Bangun menunjuk ke Lingkaran, maka “sifatnya” akan seperti Lingkaran, jika menunjuk ke Kubus, sifatnya seperti Kubus

```

abstract class Bangun {
    abstract double getLuas();
}

class Lingkaran extends Bangun {
    private int r;

    public Lingkaran(int r) {
        this.r = r;
    }

    double getLuas() {
        return Math.PI*r*r;
    }
}

class Kubus extends Bangun {
    private int s;

    public Kubus(int s) {
        this.s = s;
    }

    double getLuas() {
        return (int)s*s;
    }
}

public class BangunRuang {
    public static void main(String[] args) {
        Kubus k = new Kubus(5);
        System.out.println("Luas Kubus = " + k.getLuas());

        Lingkaran r = new Lingkaran(10);
        System.out.println("Luas Lingkaran = " + r.getLuas());

        Bangun b = r;
        System.out.println("Luas " + b.getClass().getName() + " = " + b.getLuas());

        b = k;
        System.out.println("Luas " + b.getClass().getName() + " = " + b.getLuas());
    }
}

```

Overloading

- Suatu kelas bisa memiliki behaviour lebih dari satu dengan nama yang **sama**, tetapi dengan parameter yang **berbeda** (urutan maupun tipe)

```
public class Mahasiswa {  
    /* Boleh */  
    public boolean find (String nama) {return true;}  
    public boolean find (String nama, int nim) {return true;}  
    public boolean find (int nim) {return true;}  
  
    /* Tidak Boleh */  
    public boolean find (String nama, int nilai) {return true;}  
}
```

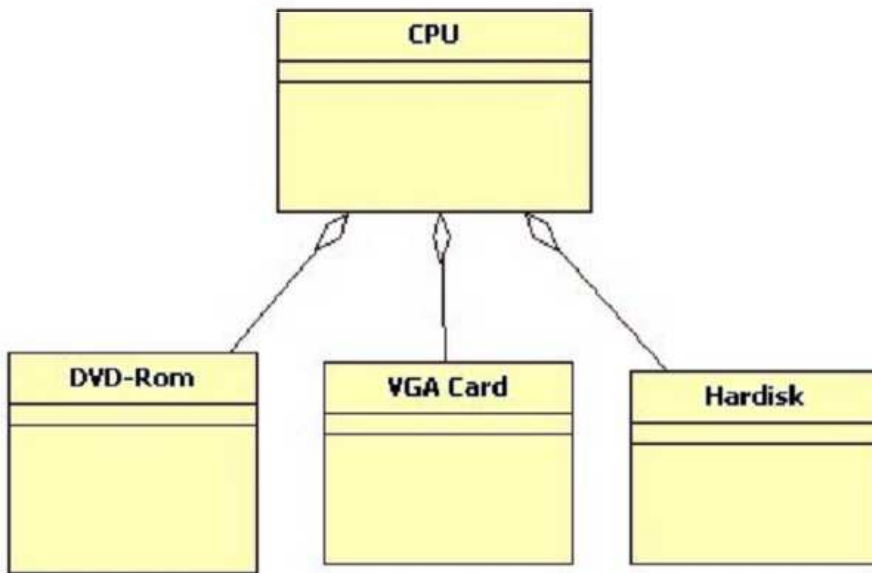
```
Mahasiswa.java:5: find(java.lang.String,int) is already defined in Mahasiswa  
    public boolean find (String nama, int nilai) {return true;}  
                ^
```

```
1 error
```

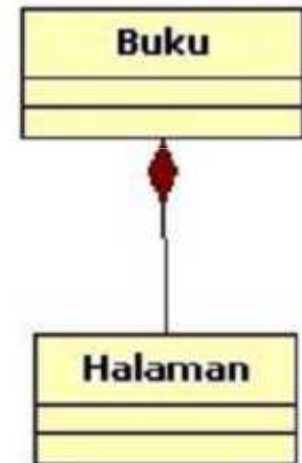
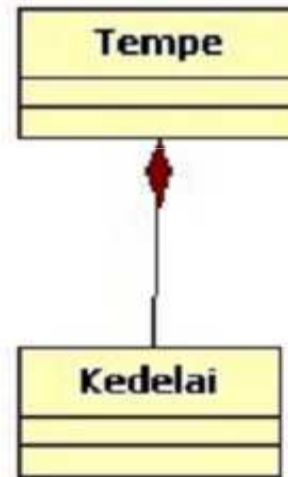


Containment

Gambar Hubungan Asosiasi



Gambar Hubungan Agregasi



Gambar Hubungan Komposisi

Ada Pertanyaan ?

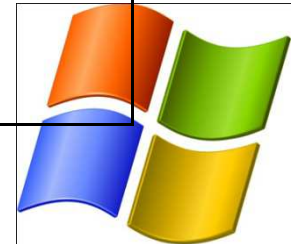
BAGIAN 2
COMPONENT ORIENTED
PROGRAMMING

Component Oriented Programming (COP)

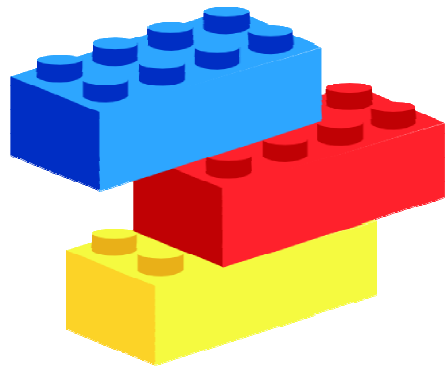
- Pemrograman Berorientasi Komponen
- Komponen sebagai sudut pandang utama
- Program tersusun dari komponen-komponen yang saling terhubung dalam hubungan yang terdefinisi dengan baik, dapat digunakan ulang dan saling independen satu sama lain

Kenapa COP Penting ?

Sistem Operasi	Lines of Code
Windows NT 4.0	11-12 Juta
Windows 2000	29+ Juta
Windows XP	40 Juta
Windows Server 2003	50 Juta
Windows 7	> 50 Juta



Kenapa COP Penting ?



Kenapa COP Penting



Black Box



Grey Box

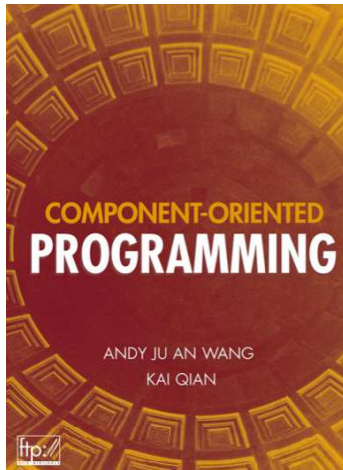


White Box

Kenapa COP Penting ?

- Perkembangan software yang semakin kompleks, butuh teknik untuk mengatasi kompleksitas tersebut
- Menangani perubahan sesuai kebutuhan
- Penggunaan ulang untuk mengurangi waktu pengembangan

Komponen Itu Apa ?



- A software component is a piece of **self-contained, self-deployable** computer code with **well-defined functionality** and **can be assembled** with other components through its **interface**.

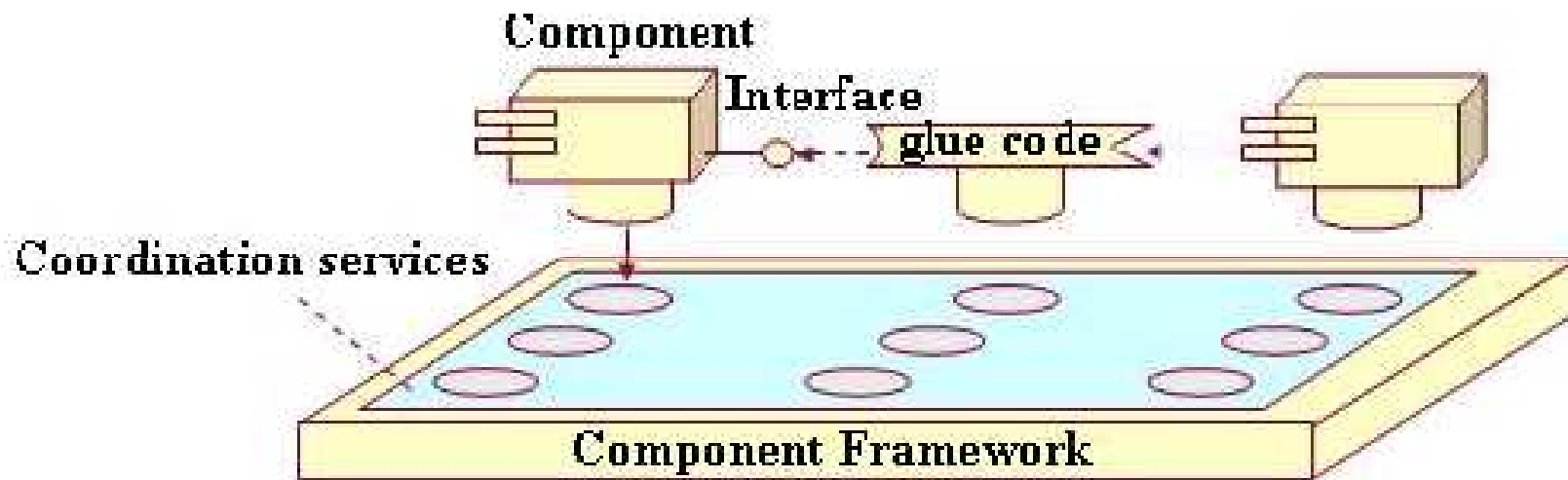
Komponen Itu Apa ?

- Self contained
- Self deployable
- Well-defined functionality
- Can be assembled through its interface

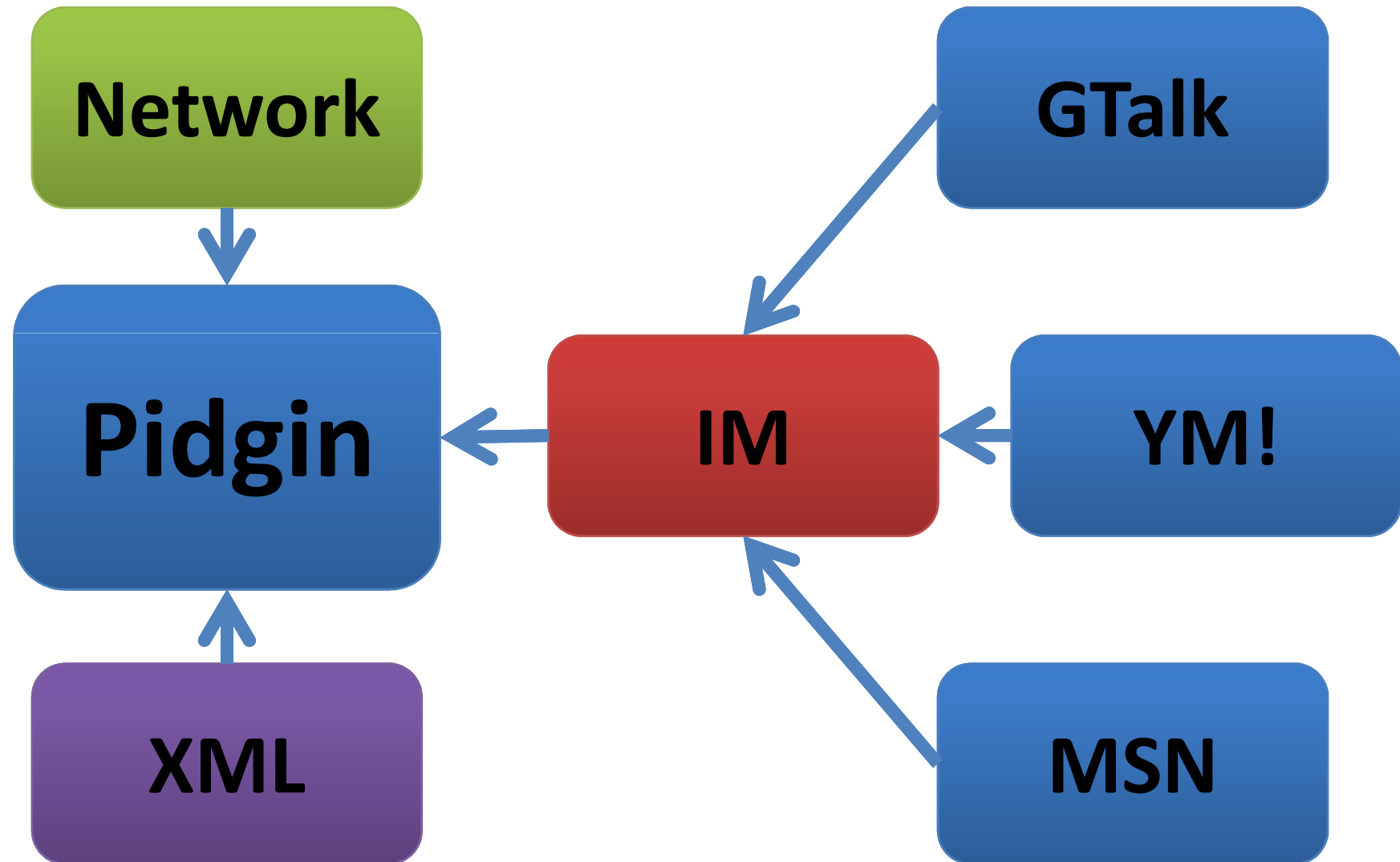
Komponen Itu Apa ?



Komponen Itu Apa ?



Komponen Itu Apa ?



Contoh Komponen

- JavaBeans dan EJB (Sun Microsystem)
- COM, DCOM, ActiveX dan .Net Component (Microsoft)
- Corba (OMG)
- XPCOM (Mozilla)
- VCL, CLX dan LCL (Borland)
- UNO (OpenOffice.org)
- Dan lainnya ...

Perkembangan Paradigma Pemrograman

- Structured Programming (SP)
 - Pemecahan program menjadi fungsi-fungsi
- Object Oriented Programming (OOP)
 - Object sebagai penyusun program
- Component Oriented Programming (COP)
 - Program tersusun dari komponen

SP vs OOP vs COP

Divide and Conquer

- Memecah masalah besar menjadi masalah-masalah yang lebih kecil
- Untuk menangani kompleksitas

SP	OOP	COP
YES	YES	YES

SP vs OOP vs COP

Data + Function		
- Menggabungkan data dan fungsi		
SP	OOP	COP
NO	YES	YES

SP vs OOP vs COP

Encapsulation		
- Pembungkusan, isolasi - Mengurangi coupling (ketergantungan)		
SP	OOP	COP
NO	YES	YES

SP vs OOP vs COP

Identity

- Setiap software entitas punya identity yang unik

SP	OOP	COP
NO	YES	YES

SP vs OOP vs COP

Interface

- Ketergantungan spesifikasi
- Membagi spesifikasi komponen menjadi interface
- Mencegah ketergantungan antar interface

SP

OOP

COP

NO

NO

YES

SP vs OOP vs COP

Deployment		
- Unit abstrak dapat dideploy secara independen		
SP	OOP	COP
NO	NO	YES

COP vs OOP

COP

- Interface-based
- Packaging and distribution technology
- Supports high-level reuse
- Can be written in any language
- Loosely coupled components
- Supports multiple interfaces and interface-oriented design

OOP

- Object-based
- An implementation technology
- Supports low-level reuse
- Bound to OO languages
- Tightly coupled objects dependent on each other through inheritance implementation
- Does not provide clear relationship of interfaces among superclasses and subclasses.

COP vs OOP

- Supports more forms of dynamic binding and dynamic discovery
- Has better mechanisms for third-party composition
- Provides more support for higher-order services (security, transactions, etc.)
- COP components are designed to obey rules of the underlying component framework
- OOP provides limited support for object retrieval and runtime composition mechanisms
- Has limited forms of connectors (method invocation)
- Has limited sets of supported services such as security, transactions, and so on
- OOP objects are designed to obey OO principles

Ada Pertanyaan ?

Materi Berikutnya
Pengantar Design Pattern
(Strategy dan Observer Pattern)