Object Oriented Programming [Concept]

Inggriani Liem

Pustaka OOP

- [Meyer97] Bertrand Meyer, “Object Oriented Software Construction”, 2nd edition, Prentice Hall, 1997.
- Stroustrup : C++ Programming Language
- Java : http://java.sun.com
- C# : http://

Definisi OOP

- **[Meyer98]:** Sebuah sistem yang dibangun berdasarkan metoda berorientasi objek adalah sebuah sistem yang komponennya di-enkapsulasi menjadi kelompok data dan fungsi, yang dapat mewarisi atribut dan sifat dari komponen lainnya, dan komponen-komponen tersebut saling berinteraksi satu sama lain.

Karakteristik OOP

- Abstraksi
- Enkapsulasi
- Generik/templates
- Pewarisan (inheritance)
- Polymorphisme
- Spesialisasi - generalisasi
- Komunikasi antar objek
- Reuseability
- Component
- Pattern

“Tingkatan” OOProgramming

- OOP ‘hanya’ untuk enkapsulasi, simple : ADT (Abstract Data Type) pada pemrograman prosedural
- OOP dengan genericity
- OOP dengan inheritance, multiple inheritance, repeated inheritance
- OOP dengan polymorphism
- OOP secara konkuren

OOP dengan design pattern, component, framework.

Bahasa OO (OOL)

- Murni : Smalltalk, Eiffel, Java
- Procedural - OO : C++
- Functional - OO : Object LISP
- Deklarative – OO : **beberapa versi OO**

Memprogram secara OO tidak harus menggunakan bahasa OO.

Contoh yang sudah diajarkan : ADT dalam bahasa C

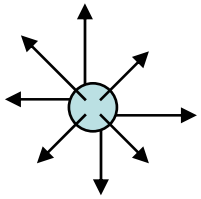
Pertanyaan praktis:

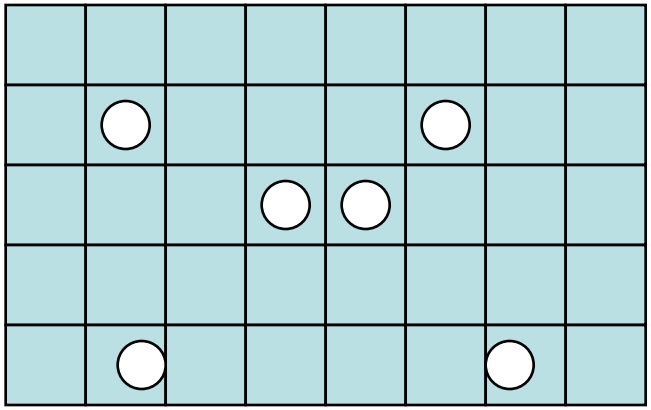
Apakah sebuah program diprogram “secara” OO?

- OO adalah sebuah “paradigma”, cara berpikir, sudut pandang. Cara berpikir bukan “produk” (source code). Source code adalah “hasil”
- Walaupun ditulis dalam sebuah bahasa OO, sebuah program mungkin saja :
  - Merupakan sebuah program Prosedural dan sekuensial. Salah satu ciri: tidak ada definisi CLASS dibuat oleh programmer
  - Programmer mendefinisikan kelas, namun hanya sebatas pembungkus ADT. Ada definisi kelas, namun programnya prosedural
  - Program menggunakan library Class yang bukan/maupun merupakan GUI object, sehingga ada pemakaian kelas yang disediakan.
  - Program memang dirancang berparadigma OO. Untuk mampu mengetahui hal ini, diperlukan “persepsi” khusus ☺ ☺ ☺

Ilustrasi: Bola dalam bidang

- Sebuah bidang mengandung sekumpulan bola. Setiap bola mempunyai arah
- Bola bergerak sesuai dengan arahnya
- Jika “bersitabrak”, maka bola akan mati
- Jika bola habis sama sekali, sistem mati
- Jika bola tersisa “sedikit”,
- akan lahir bola-bola baru
- sehingga jumlah tertentu





Ilustrasi: bola dalam bidang

- Solusi prosedural sekuensial dengan ADT list of bola, dan sebuah main program:
  - Main program mengendalikan pergerakan setiap bola secara sekuensial.
  - Main program mengirim pesan kepada setiap bola, supaya “menggerakan dirinya saat diberi pesan untuk bergerak
- Solusi dengan proses konkuren dan ADT sederhana, idem di atas namun misalnya menggunakan satu “thread” untuk setiap pergerakan bola
- Solusi dengan proses konkuren dan ADT generik (list of list) : list of cell(position), list of bola, . . .
- Solusi di mana penggerak bola dan viewer yang diimplementasi dengan mesin gambar, ada satu atau empat mesin gambar
- Solusi lainnya : ……..

Paradigma OO

- **Pra kehidupan**: ada dunia “sebelum” jagat raya & objek dihidupkan, definisi “statis” [kelas, hubungan antara kelas, kontrak]. Programmer berfungsi sbg perancang kelas.
- **Saat run time**: dimulailah kehidupan (biasanya dengan mulai hidupnya Sang Objek Utama, yaitu aplikasi yang akan menghidupkan objek lain. Setiap objek akan hidup dan berinteraksi dengan objek lain sesuai dengan definisinya. Setelah tugas menghidupkan selsai, Sang Objek Utama “tidur”. Objek saling berinteraksi dan mengirimkan pesan, sesuai dengan definisinya. Objek yang sudah tidak dibutuhkan dihancurkan atau menghancurkan diri.
- **Akhir kehidupan**: Sang Objek Utama terbangun, dan lenyap bersama jagat raya beserta seluruh objek tersisa.

OOP: Class vs Object

- **Class**: entitas statik, didefinisikan dalam teks program.
- **Object**: entitas dinamik, instansiasi dari class, ada pada saat run time.
- **Kehidupan objek**: creation, manipulation, destruction.
- **Memprogram OO**:
  - Controller: melahirkan objek, mengendalikan kehidupan objek tsb.
  - Objek lain: saling mengirimkan message dan melaksanakan operasi

Catatan Penting

- Class mempunyai “feature”:
  - Method
  - Atribut
- Implementasi prosedur dan fungsi dalam sebuah kelas, pelajari perbedaan dengan prosedural (akan diberikan contoh):
  - Perhatikan penulisan prototype/signature
  - Perhatikan parameter input, output dan input/output
  - Perhatikan passing parameter, by value by ref
  - Perhatikan invokasi (bukan CALL)

Invokasi/Message Passing

```
Class Point {
 int x; int y;
//getter
//setter
//fungsi
 Point MirrorOf( );
//prosedur
  void Mirror ();
}
```

```
void main {
   Point P1, P2;
   P2 = P1.Mirrorof ();
   P1.Mirror ();
}
```

Prosedural

```
typedef struct {
 int x; int y;} Point;
//getter
//setter
//fungsi
 Point MirrorOf(Point P );
//prosedur
 void Mirror (Point *P);
```

```
void main {
   Point P1, P2;
   P2 = Mirrorof (P1);
   Mirror (P1);
}
```

Persoalan ☺ ☺ ☺

Contoh kasus :

- Function Plus : Point x Point → Point
- Prosedure SWAP untuk menukar nilai dua buah POINT

- Buatlah program OO untuk kedua primitive tersebut.

Main program

- Main program: titik awal eksekusi, pemicu “kehidupan” objek utama
- Dalam sebuah program OO, main program hanya bertugas menghidupkan objek, kemudian “tidur”, sampai semua objek sudah dimusnahkan
- Realisasi:
  - Dalam beberapa bahasa pemrograman, main program harus merupakan instansiasi dari kelas, atau diaktivasi dengan teknik tertentu tanpa instansiasi kelas. Contoh: JAVA [`public static void main()`]
  - Dalam bahasa lain tidak harus. Contoh: C++

Contoh
Main dihidupkan dari dunia luar

```
class X {
//
}
public class DriverX {
// mengandung main program
// berupa driver untuk test X
Public void main () { }
}
```

Contoh
Main hanya menghidupkan objek

```
class X {
//
}
class DriverX {
 // constructor
 public void DriverX() {// isinya kode driver
 }
}
class MainX {
 // program utama hanya menghidupkan "objek" DriverX
   DriverX d= new DriverX();
}
```

Contoh Main Program SWING

```
public class SwingApplication implements
   ActionListener {public static void main(String[]
   args) {
  //Schedule a job for the event-dispatching thread:
  /creating and showing this application's GUI.
  javax.swing.SwingUtilities.invokeLater(new
    Runnable() {
              public void run() {
                  createAndShowGUI();
              }
         });
     }
}
```

Object

- Setiap object mempunyai reference (pointer dalam konteks prosedural)
- Reference: punya salah satu dari dua keadaan
  - unattached (void, NULL), belum punya "container"
  - attached (sudah dialokasi container/memori untuk menyimpan data)

Klasifikasi Objek [Booch]

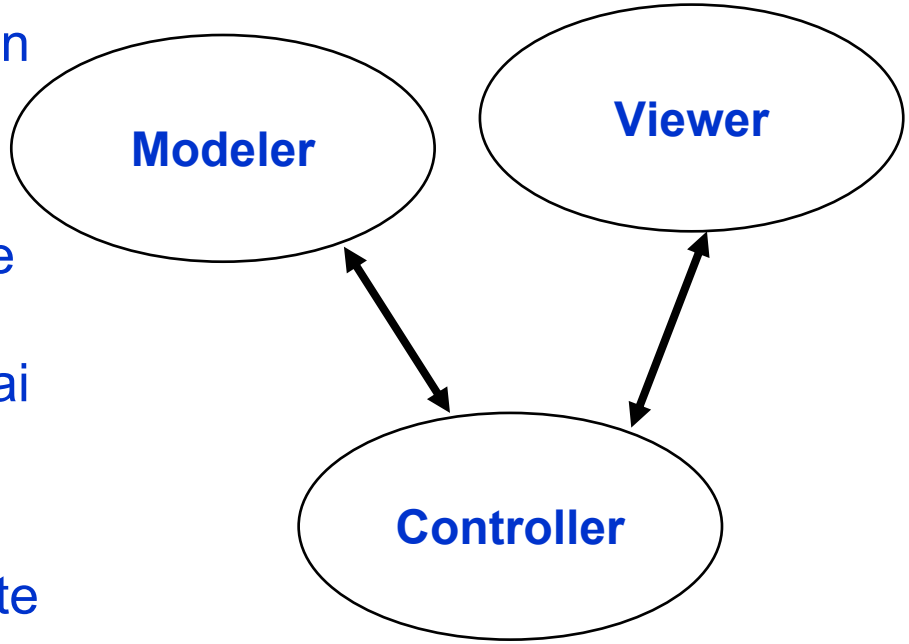
- ADT:
  - Objek pasif, definisi “type”, operasi terhadap type
- Mesin:
  - objek pasif yang mempunyai state dan primitif
- Proses:
  - objek aktif
- Catatan:
  - Pada source code, ada definisi kelas dan penghidupan objek
  - Dalam program yang sedang running, hanya ada objek
  - Sebuah objek adalah hasil instansiasi kelas
  - Jadi, mesin dan bahkan “proses” adalah instans!

Klasifikasi Objek [UML]

- Boundary Object:
  - interface dengan dunia luar (system boundary)
- Entity Object:
  - domain model
- Controller:
  - mengontrol pemfungsian software

Klasifikasi Objek [MVC]

- Modeler
  - Representasi domain persoalan yang diprogram
- Viewer
  - Menampilkan state modeller ke dunia luar
  - Satu modeler boleh mempunyai banyak Viewer
- Controller
  - Mengendalikan perubahan state modeler dan tatacara mengkomunikasikan/menerima perubahan state modeler ke dunia luar

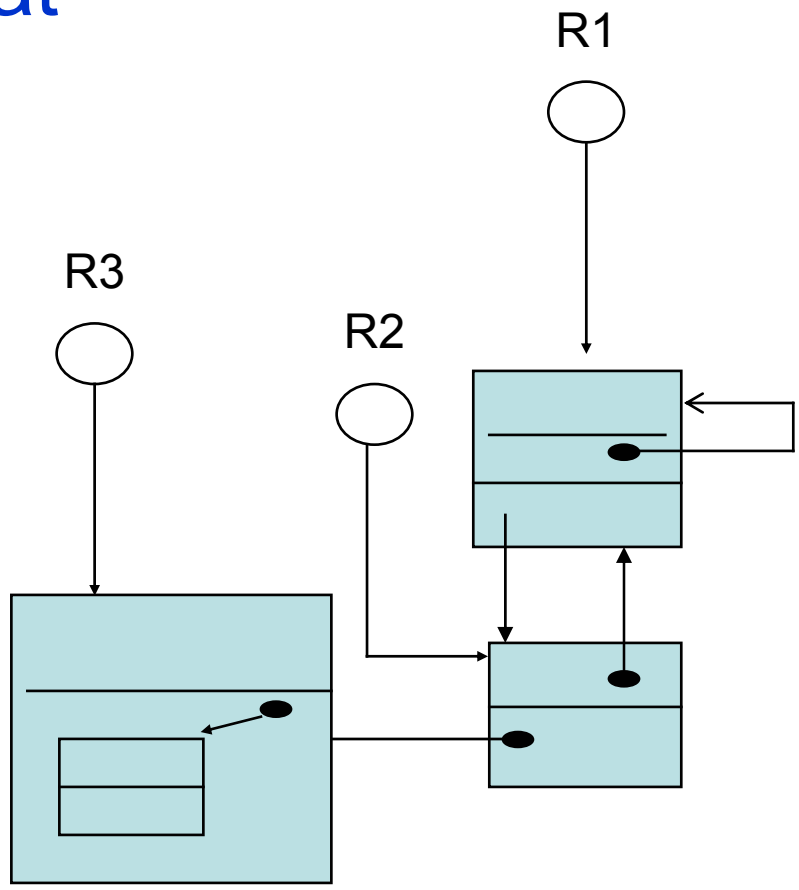


Reference

- Reference : entitas yang hanya mempunyai dua kemungkinan state :
  - Void (NULL)
  - Attached
- Jika statusnya “attached”:
  - harus yakin bahwa memori suah dialokasi dan mengacu ke container objek yang dimaksudkan.
  - Berhati-hati dengan dua buah reference yang “attached” ke kontainer objek yang “identik”
- Reference bukan “pointer” ☺
- “pointer” [C, C++] juga berbeda dengan “reference” [Java, C++]

OOP: Object

- Reference ke Objek (lihat contoh)
- Reference versus container (memori)
- Composite objek
- Operasi:
  - Attachment, Clone,Deep Clone
  - Compare

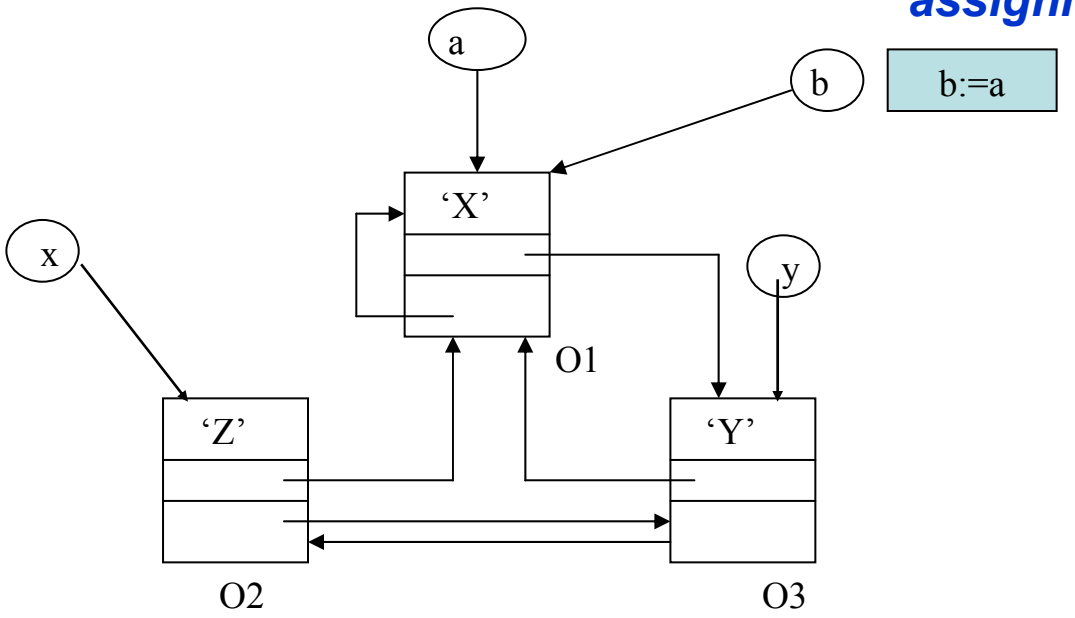


Contoh Operasi [1]
Reference Attachment, assignment

Class: Orang

Nama : String
CintaKe : Orang
berGuruKe :Orang

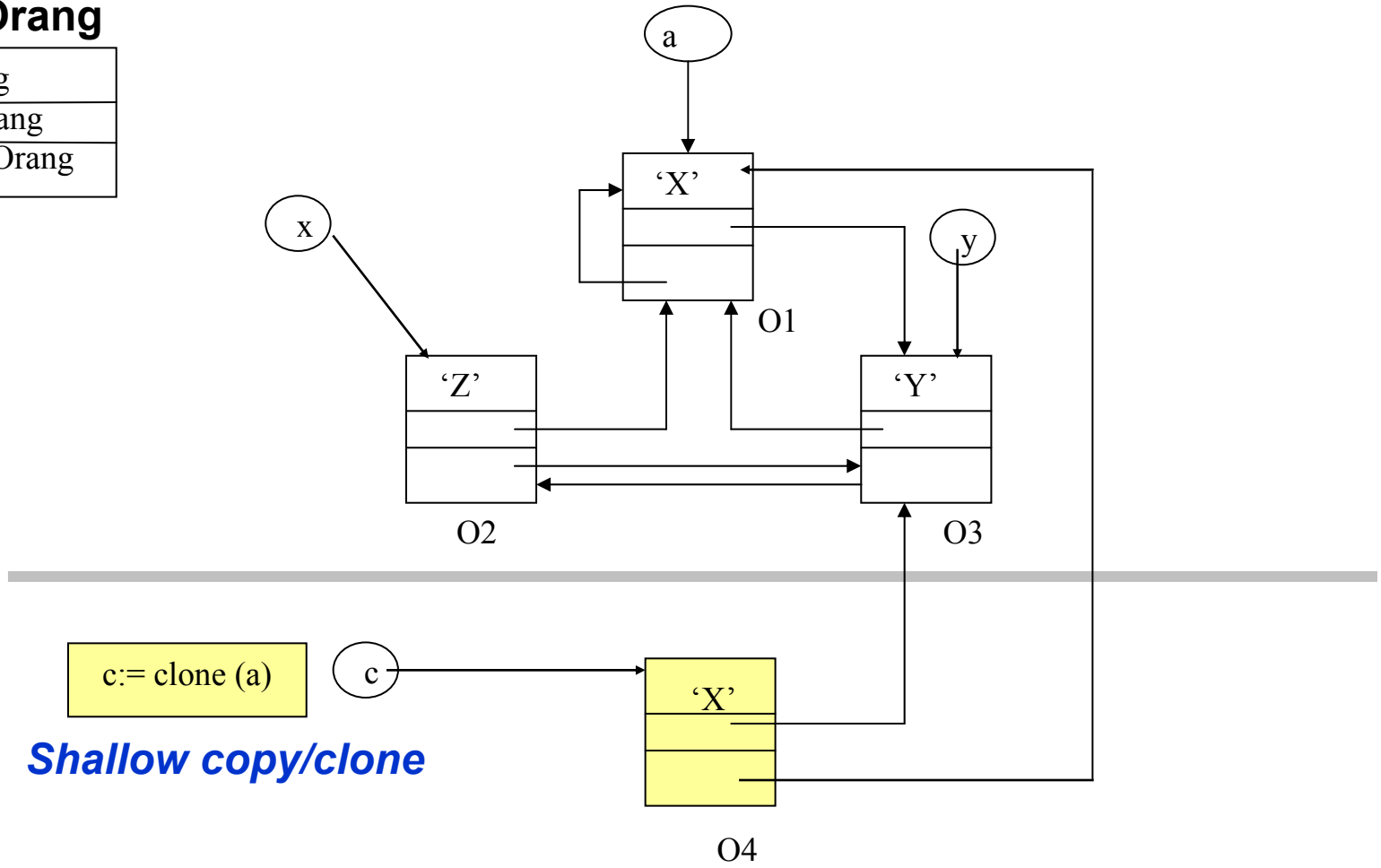
Reference attachment, "assignment"



Contoh Operasi [2]
Clone, Shalow copy

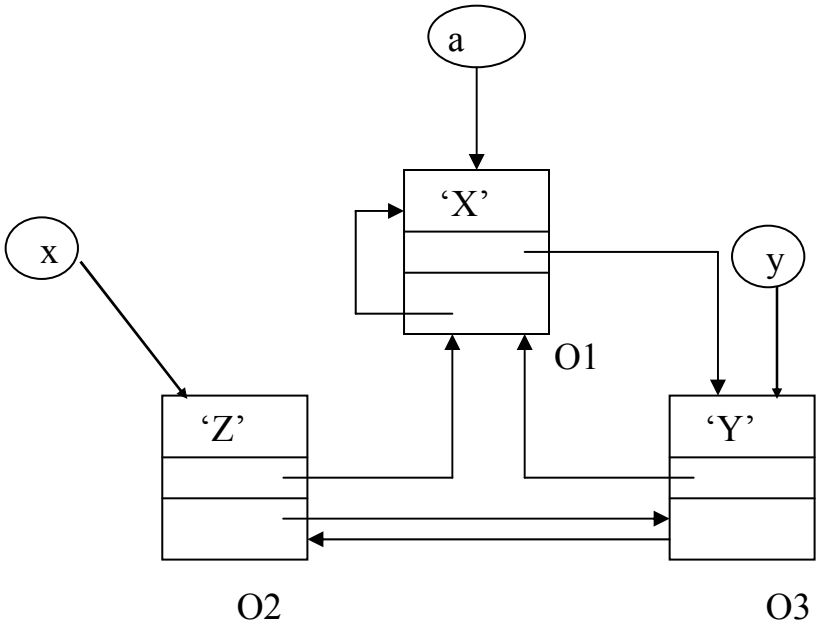
Class: Orang

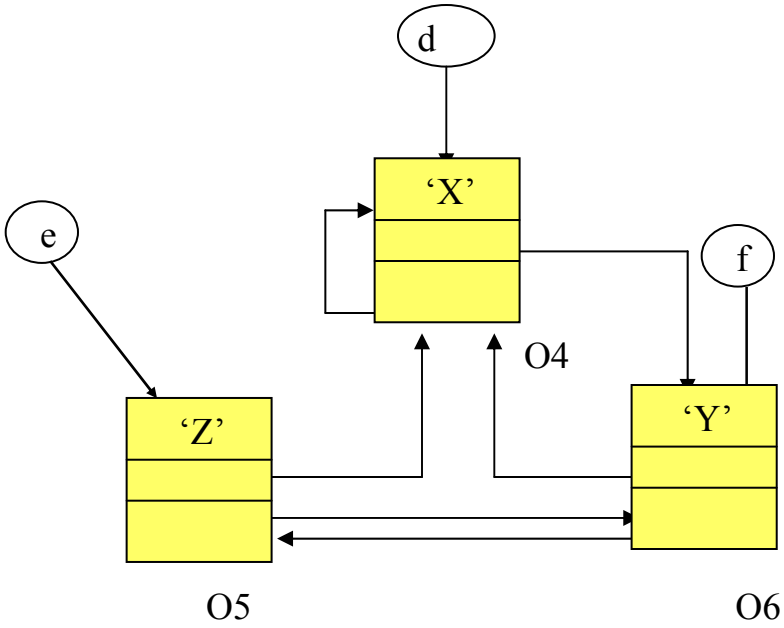
Nama : String
CintaKe : Orang
berGuruKe :Orang



Shallow copy/clone

Contoh Operasi [3]
Deep clone/deep copy





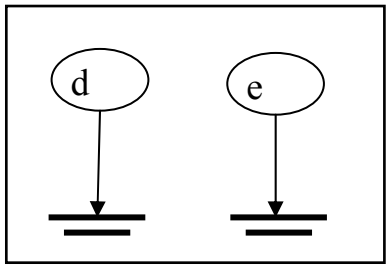
d := deepclone(a)

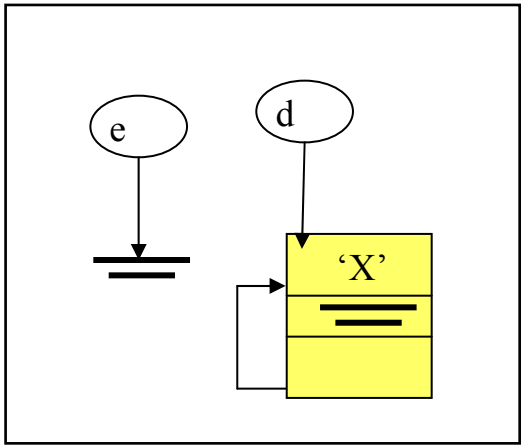
Deep clone / copy

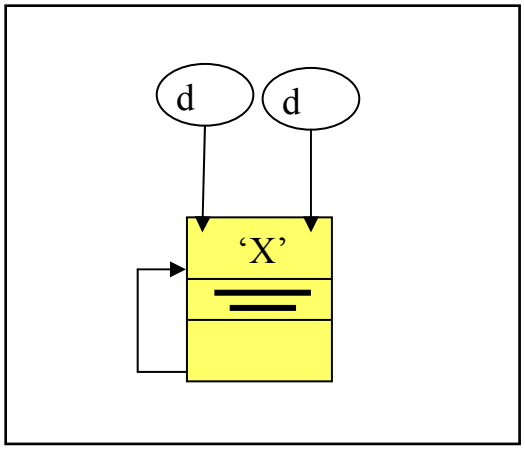
Contoh Operasi Perbandingan Reference, Value

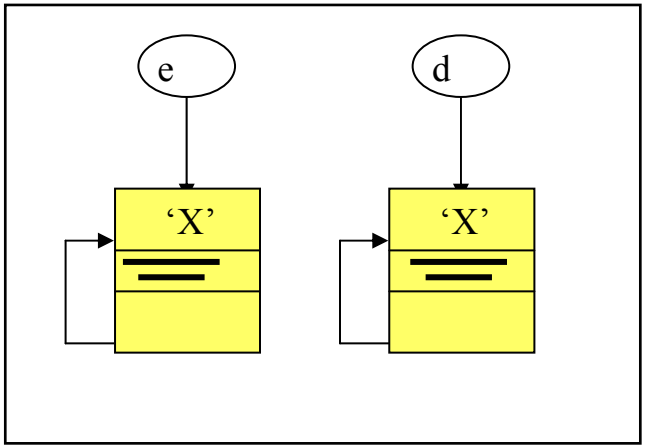
???

d = e









Abstract/Meta/Deferred/Virtual Class

- Tidak bisa dan tidak boleh diinstansiasi objeknya, biasanya berisi spesifikasi.
- Dua ciri:
  - Deklarasi kelas dengan keyword tertentu: “deferred”, “virtual”, “abstract”, meta
  - Mengandung feature yang belum lengkap, hanya spesifikasi, harus diimplementasi
- Fitur yang belum lengkap harus dilengkapi oleh Kelas turunan. Setelah lengkap baru boleh diinstansiasi menjadi objek

Root Class & Kelas Pengunci

- Root Kelas: kelas induk, semua kelas yang didefinisikan dalam program merupakan turunan dari kelas ini. Contoh: dalam bahasa Java, semua Class yang didefinisikan oleh programmer akan merupakan instans dari kelas **`Object`**
- Kelas pengunci, merupakan turunan dari semua kelas

Konsep Interface (spesifik JAVA)

- Interface adalah semacam kelas, bukan kelas
- Interface tidak boleh mengandung state, dan belum mempunyai body
- Body dari Interface akan diimplementasi oleh kelas yang memakai interface

Generic Class

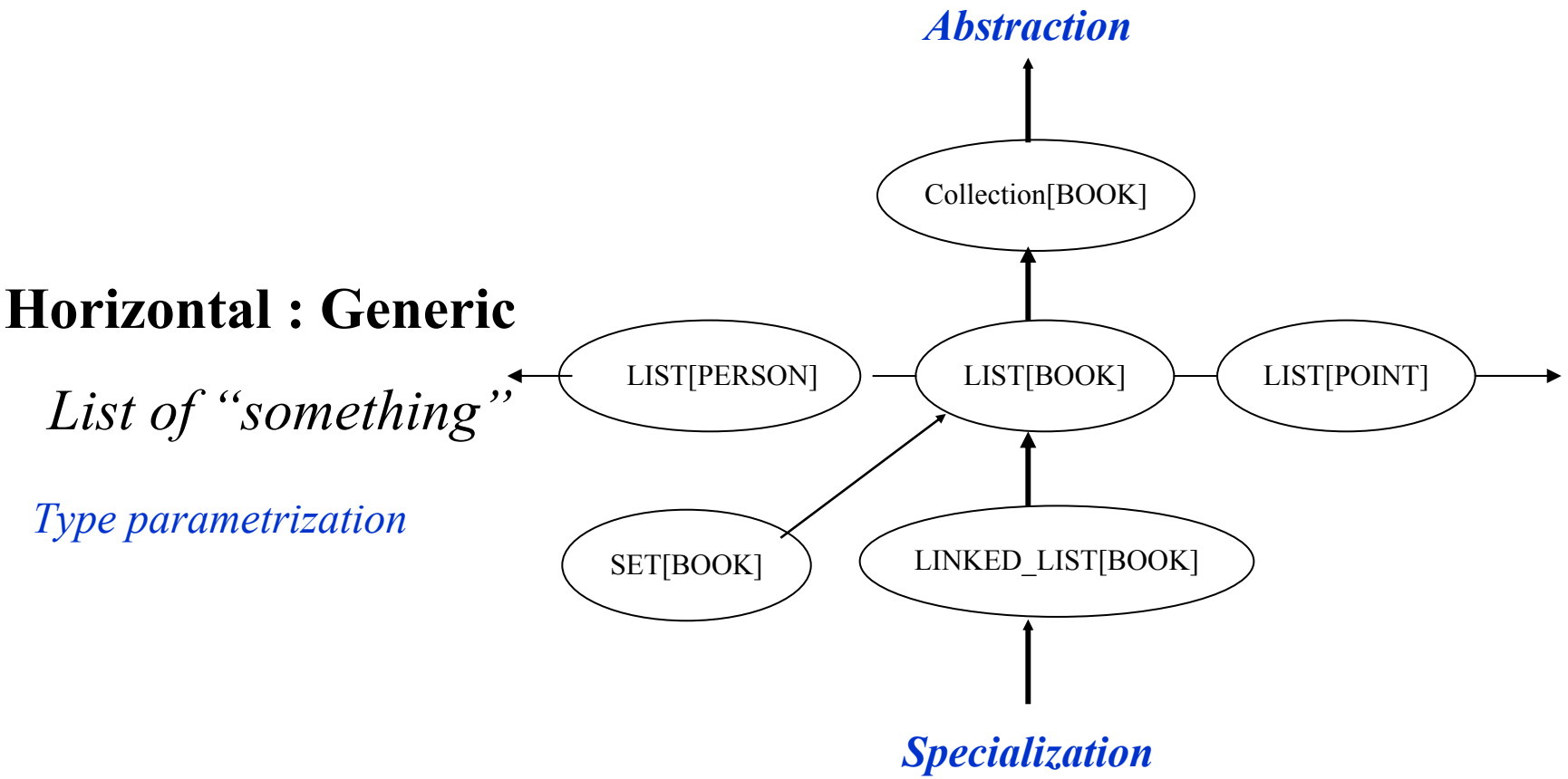
- Kelas Generik: kelas yang masih “umum”, belum spesifik ketika didefinisikan.
- Pada saat deklarasi objek, hal yang masih umum harus dibuat spesifik, sehingga didapat type spesifik.
- Setelah menjadi spesifik, baru boleh dipakai
- Biasanya yang generik adalah “type” nya, dipakai untuk membungkus “operasi” yang sama
- Dalam bahasa C++ menjadi <<*templates*>>.

Contoh Generic

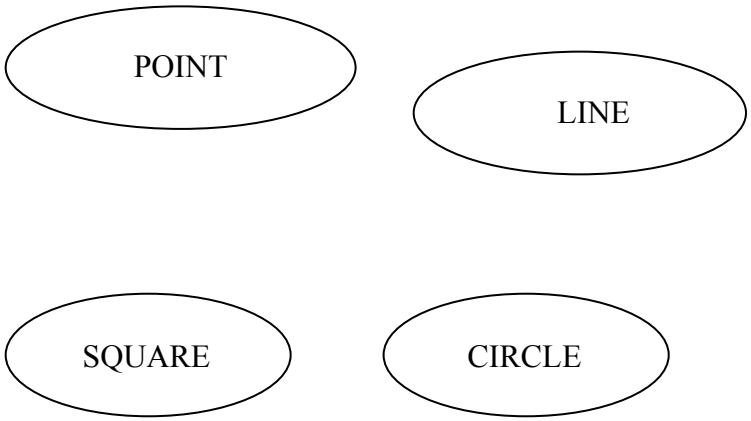
- Kelas Point yang masih generic:
  - Gpoint [Numeric]: kelas point dengan absis dan ordinat bertype numerik
  - Saat dideklarasi,
    - Gpoint[integer] P; maka absis dan ordinat akan bertype integer
    - Gpoint[real] P; maka absis dan ordinat akan bertype real
- Kelas G_array: array of element [a_type]
  - pada saat dideklarasikan dapat menjadi : array of [integer], array of [float] , array of [Point], array of [array of [Point]], . . .

Generic vs Inheritance

Vertical : Inheritance



Hubungan Antar kelas
Client Supplier, *"has"*



Class POINT { absis, ordinat :integer;}

Class LINE { P1, P2: integer;}

Class SQUARE { TopLeft,
BottomRight:Point; }

Class CIRCLE { Center :Point,
Radius:integer }

A POINT has absis and ordinat (integer);

A Line has Two POINT (P1, P2);

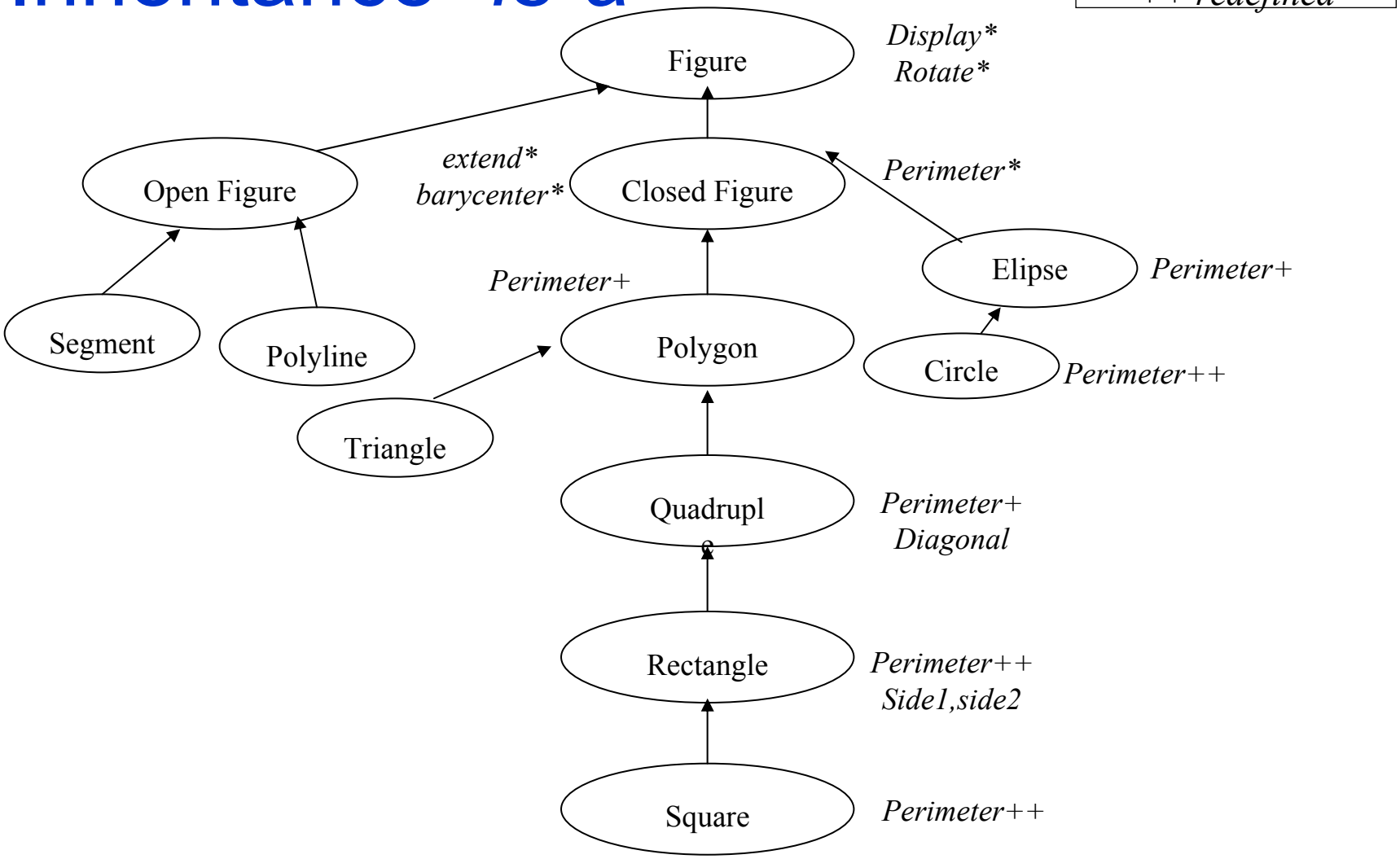
A SQUARE has Two POINT as identifier (TopLeft, BottomReight)

A CIRCLE has A CenterPoint and Radius

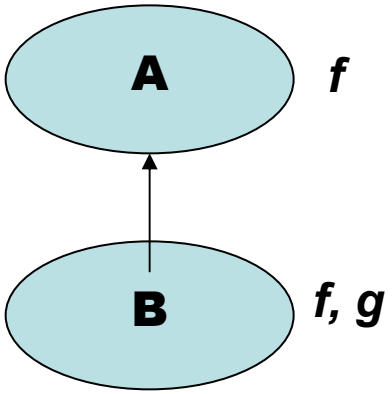
Client - Supplier

- Hubungan: “memakai”, has, terdiri dari, whole-part
- Coupling rendah, independency, delegasi
- Sebagian besar hubungan kelas adalah hubungan ini
- Peran programmer:
  - penyedia kelas, membuat spesifikasi, menjamin post condition
  - pemakai kelas, mentaati spesifikasi, prekondisi

Hubungan Antar Class Inheritance *"is-a"*



Inheritance



- Istilah:
  - parent, ascendant, base class, leluhur.
  - child, descendant, derived class turunan.
- Memungkinkan:
  - feature overriding
  - kompatibilitas kelas anak terhadap parent (tapi tidak sebaliknya) pada saat run time : polimorphism (poly= banyak; morph=bentuk)

```
A a; B b;
//valid:
a.f; b.f;
//tidak valid:
//a.g
a ← b;
a.f; //a.g
```

Inheritance

- Hubungan: is - a, gen-spec.
- Harus dipakai dengan hati-hati, ada pedoman untuk memakai hubungan ini. Bertrand Meyer menyebutkan 7 kasus di mana boleh digunakan.
- Pada saat membuat kelas dengan hubungan inheritance, harus didefinisikan seluruh pohon keturunannya dengan lengkap. Test harus dilakukan terhadap semua kelas yang berhubungan
- Jika hubungan sudah kusut, maka harus dirancang ulang secara keseluruhan

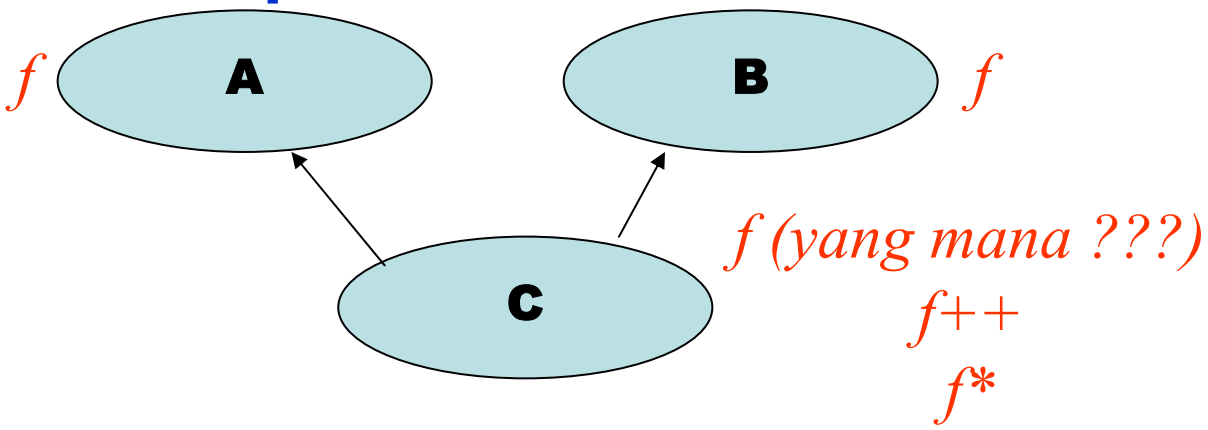
Valid use Of Inheritance

Jenis inheritance	**Sub Jenis inheritance**	**Keterangan**
Model	Subtype	Suatu sistem dapat dibagi menjadi disjoint subtype
	View	Memberikan hak akses saja terhadap beberapa feature
	Restriction	Menambahkan batasan, seperti rectangle dengan square
Variation		B adalah turunan dari A B redefine beberapa feature dari A
	Functional	Redefinition affect body, tidak hanya signature
	Type variation	Signature saja yang berubah
	Uneffecting	Redefines some of effective feature of A into deferred
SW	Reification	A adalah struktur data umum, B partial atau komplit Contoh : Tabel, bisa sebagai hash table atau sequential table. Sequential table bisa diimplementasi sebagai array atau linked
	Structure	A deferred class, represent general structural B :certain type of object Contoh : A comparable dan B adalah string atau integer
	Implementation	Implementasi fisik dari sebuah struktur lojik (misalnya stack dengan representasi array)
	Facility	Constraint : untuk mendefiniikan semua konstanta sistem, misalnya konstanta ASCII
		Machine : misalnya **iterator** (skema sekuensial)

Multiple Inheritance

- Kasus di mana ada lebih dari satu kelas Parent
- Harus dipakai dengan sangat berhati-hati
- Dalam banyak kasus, salah satu Parent harus “abstrak”
- Menimbulkan konflik (lihat slide berikut), setiap bahasa akan menyelesaikan dengan caranya, atau di tingkat pemrograman. Akan diberikan contoh

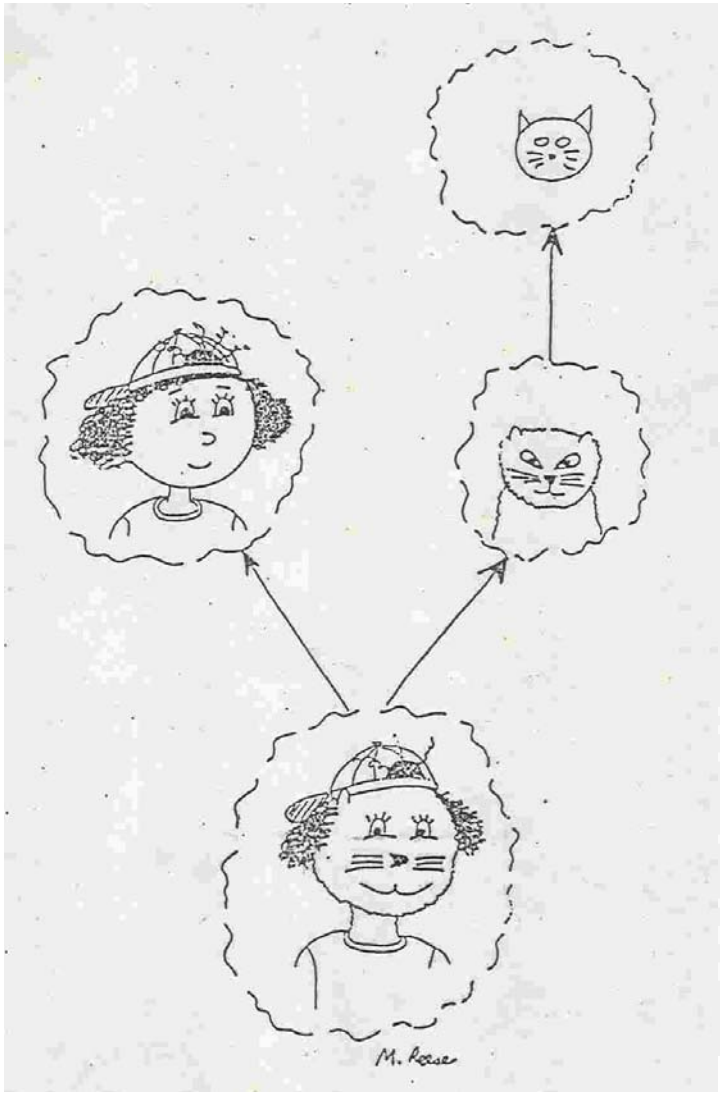
Multiple Inheritance



- Resolusi Konflik
  - Duplikasi (undefined)
  - Nama (rename)
  - Definisi (redefinition)
  - Runtime : select

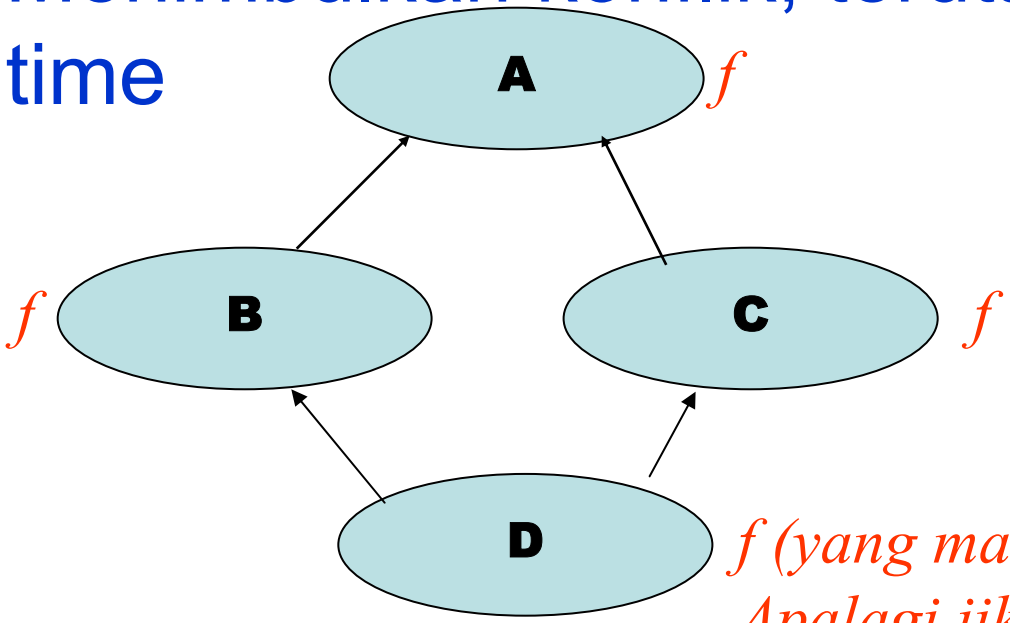
```
A a; B b; C c;
//alokasi beres
a.f; b.f; c.f
a ← c;
 b ← c;
```

Inheritance



Repeated Inheritance

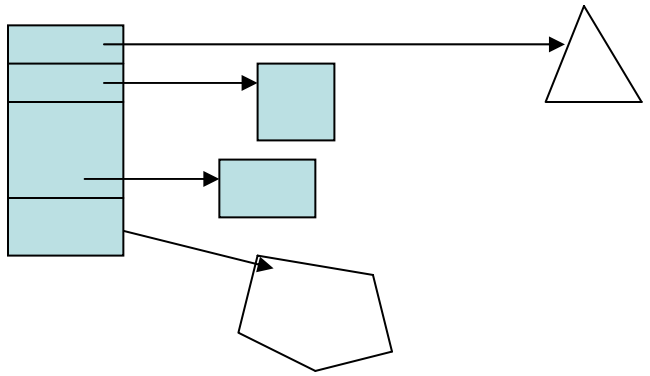
- Kasus inheritance, dimana dua buah parent mempunyai descendant yang sama
- Menimbulkan konflik, terutama saat run time




```
A a;
B b;
C c;
D d;
//beres
a.f; d.f;
a ← d;
```

Polymorphism

- Polymorphism:
  - terjadi saat run time, oleh karena itu hanya bisa terjadi pada entitas dinamik
  - Kompatibilitas “attachment” Parent dengan Child
- Polimorphic attachment
- Polimorphic data structure



Passing Parameter

- Polymorphic juga dapat terjadi saat passing parameter. Perhatikan kompatibilitas parameter aktual dengan parameter formal
- Contoh:
  - sebuah method mempunyai parameter sebuah kelas Foo.Kelas Fog adalah turunan Foo
  - Maka , misalnya `Display (F:Foo)` akan dapat dipakai untuk melakukan Display terhadap objek Kelas `Fog`

Exception

- Penanganan eksepsi: mekanisme yang berhubungan dengan kondisi yang tidak diharap pada saat eksekusi
- Beberapa kata kunci dalam hubungan dengan eksepsi: `catch, retry, throw`
- Failure: sebuah routine gagal memenuhi kontrak. Rutin mendapat eksepsi sebagai hasil dari failure, salahnya asersi, kondisi abnormal oleh HW dan OS
- Rutin yang mengurus eksepsi: retry atau organized panic. Retry mengekesekusi ulang body, Organized panic menyebabkan kegagalan meneruskan exception ke pemanggil.
- Aturan formal dari exception handler: me-restore invariant sehingga dapat di-reeksekusi kembali untuk memenuhi kontrak.

Asersi [1]

- Asersi: ekspresi boolean, ekspresi dari sifat (property) semantik kelas, menyatakan aksioma dan prekondisi dari class ybs
- Asersi dipakai pada prekondisi, postkondisi dan invarian kelas. Di bahasa Eiffel ada instruksi invariant dan CHECK yang berhubungan dengan invarian
- Prekondisi dan postkondisi adalah pernyataan kontrak antara Client dan Suplier. Pernyataan kontrak merupakan dasar yang kuat untuk kebenaran program
- Invariant dari kelas: ekspresi dari batasan semantik terhadap instans dari kelas. Invariant secara implisit ditambahkan pada prekond dan post cond dari semua rutin yang dieksport dari kelas tsb.

Asersi [2]

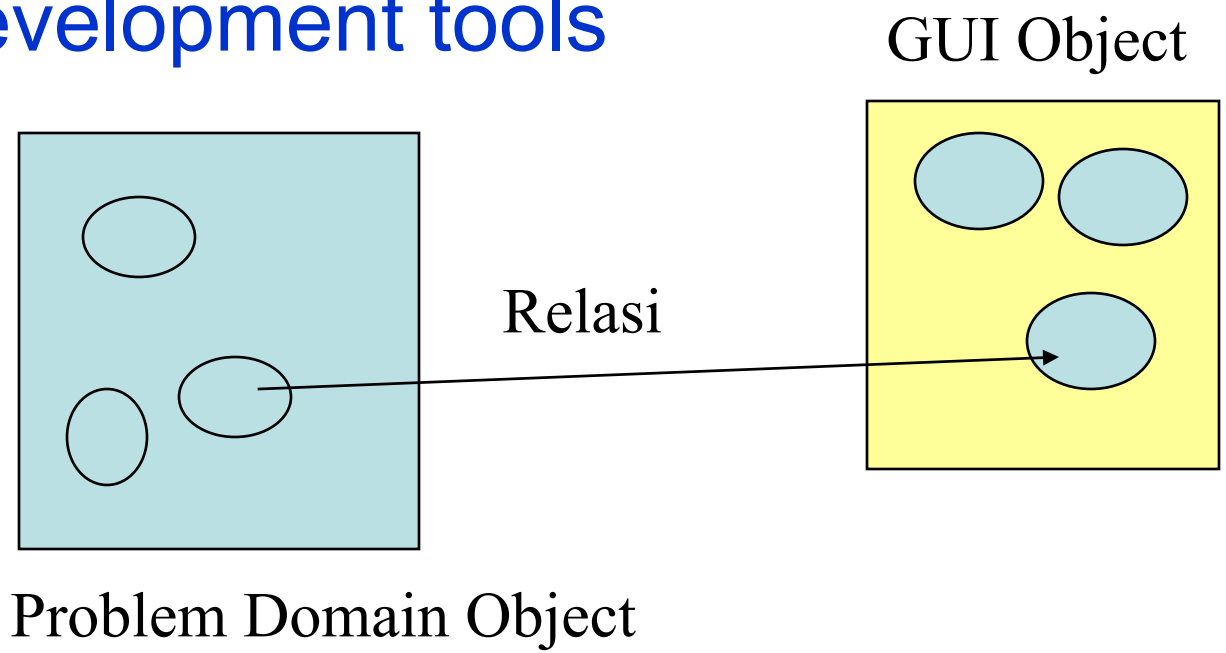
- Implementasi dari invarian, bagian dari Class invariant, merupakan ekspresi dari representasi terhadap ADT
- Loop dapat mengandung loop invariant dan variant. Invariant dipakai untuk deduksi property hasil, variant dipakai untuk menjamin berhentinya loop
- Jika kelas dilengkapi dengan asersi, maka timbul kemungkinan mendefinisikan secara formal, definisi kebenaran dari kelas
- Asersi mempunyai 4 kegunaan: bantuan menghasilkan program yang benar, dokumentasi, debugging aid, basis dari exception mechanism

OOP dan lingkungan Windows

- OOP vs event drivent programming
- OOP vs Visual programming
- OOP dengan Lingkungan pemrograman yang menyediakan tools

Pemrograman di lingkungan GUI Object Based Programming

- GUI: interaksi standard (seperti di lingkungan Windows)
- Development tools



GUI Object

- **Window**
- **Dialog (modal, modeless dialog),.**
- **Menu (toolbar, menu item, …)**
- **Button (check box, combo box, radio button)**
- **Scroll bar**
- **Text box (editable, non editable)**
- **Timer**
- **Gambar sederhana (line, box, circle,…)**
- **Image**
- **Grafik (line, bar chart, pie chart,.)**

Tahap Perancangan Program

- Buat snapshot layar, dengan objek yang akan diletakkan di layar tsb
- Untuk setiap objek di layar, definisikan AKSI (prosedur) yang diasosiasikan jika terjadi ***event*** tertentu
- Buat script (rangkaian layar)

Tahap Pemrograman

- Ciptakan objek dari root class (misalnya Form)
- Ciptakan objek lain yang diletakkan di layar dan set porpertinya
- Tulis kode program yang mewakili AKSI yang terdefinisi untuk setiap event

Menciptakan objek, baru mengkode