DIKTAT KULIAH
Pemrograman Berorientasi Objek

Oleh :

Inggriani Liem

Departemen Teknik Informatika

Institut Teknologi Bandung

2003

Kata Pengantar

Diktat kecil ini merupakan diktat pendahuluan sebelum mahasiswa mendapatkan kuliah dan praktek Pemrograman Berorientasi Objek dalam bahasa yang akan dipakai sebagai bahasa pengeksekusi selama kuliah (misalnya C++, Java). Sebenarnya, diktat ini hanya merupakan catatan ringkas yang diambil dari [Meyer-97], yang menyajikan pemrograman berorientasi objek dalam bahasa Eiffel. Bahasa Eiffel dipakai sebagai bahasa "pengantar" dan "konseptual" seperti halnya bahasa algoritmik pada perkuliahan Pemrograman Prosedural. Sayang bahasa ini tidak dapat dipraktekkan karena keterbatasan waktu dan ketersediaannya. Karena bahasa Eiffel dipakai sebagai bahasa konseptual, maka notasi yang dipakai adalah notasi dalam bahasa Eiffel. Di samping bahasa konseptual tersebut, disadari akan perlunya bahasa-bahasa berorientasi objek yang banyak dipakai di lingkungan industri, seperti Java dan C++. Karena itu, bahasa yang dipakai dalam Kuliah Pemrograman Berorientasi Objek di Jurusan Teknik Informatika adalah C++ dan Java.
Diktat Pemrograman dalam bahasa C++ [Dulimarta-99] dan Java [Kistijantoro-2000] telah dikembangkan oleh rekan sejawat di Jurusan Teknik Informatika.
Diktat ini juga dilengkapi dengan tiga buah lampiran:

- *Lampiran A, tabulasi perbandingan antara ketiga bahasa yang diajarkan dikuliah sebagai rangkuman. Tabulasi tersebut dikembangkan bersama oleh penulis dengan rekan-rekan pengajar tahun 1999/2000 yaitu Achmad Imam Kistijantoro dan Avan Suenesiaputra.*
- *Lampiran B, kumpulan peristilahan dalam dunia "OO", yang dicuplik dari sebuah Kamus mengenai OO. Cuplikan ini disalin dalam bahasa Inggris, supaya mahasiswa diperkaya penguasaan akan kata yang sangat beragam dan seringkali sama artinya dalam dunia OOP..*
- *Lampiran C. contoh program kecil dalam bahasa Eiffel, yang dibuat sebagai bagian Tugas Akhir [Tegawinata-98]*

Pustaka Rujukan

1. [Coad91a] Coad, Peter & Yourdon, “Object Oriented Analysis, Second edition, Prentice Hall, 1991
2. [Coad91b] Coad, Peter & Yourdon, “Object Oriented Design, Second edition, Prentice Hall, 1991
3. [Booch95] Grady Booch, “Object oriented Analysis and Design with Application”, The Benjamin/Cumming Publishing Company, 1995.
4. [Dulimarta-99] Dulimarta Hans: “Diktat Bahasa C++”, Jurusan Teknik Informatika ITB, 1999
5. [Kistijantoro00] A.I. Kistijantoro, “Diktat Pemrograman Bahasa Java”, Jurusan Teknik Informatika ITB, 2000
6. [Meyer97] Bertrand Meyer, “Object Oriented Software Construction”, 2nd edition, Prentice Hall, 1997.
7. [Tegawinata98] Agustinus Tegawinata, “Studi dan Pemrograman Berorientasi Objek Smalltalk dan Eiffel”, Jurusan Teknik Informatika ITB, 1998
8. [UML] standard OMG, 1997

DAFTAR ISI

PEMROGRAMAN BERORIENTASI OBJEK

Definisi [Meyer-97]

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.

Beberapa definisi lain

Object orientation:

1.a. the paradigm that use objects with identity that encapsulate properties and oeprations, message passing, class, inheritance, polymorphism, and dynamic binding to develop solution that model problem domains [Firesmith, Lorenz]

1.b. any technique based on the concept of object, class, instances and inheritance [Jacobson]

2. the use of objects as the atom of modeling [Coleman]

Object adalah abstraksi dari sesuatu yang mewakili sesuatu pada dunia nyata. Pada pemrograman berorientasi objek, Objek adalah entitas pada saat **run time**. Objek mempunyai siklus hidup : diciptakan, dimanipulasi, dihancurkan. Sebuah objek dapat diacu lewat namanya atau lewat referensinya (addressnya)

Class adalah kumpulan objek yang mempunyai atribut yang sama. Class adalah definisi statik dari entitas.

Entitas :

Entitas adalah salah satu dari berikut ini:
- atribut kelas
- variabel lokal
- parameter formal
- hasil fungsi

Tujuh langkah untuk mendapatkan hasil (SW) yang memuaskan [Meyer-97]:

1. *Object based modular structure*, sistem dimodularisasi berdasarkan struktur objek
2. *Data abstraction*, objek harus dideskripsikan sebagai implementasi dari ADT
3. *Automatic memory management*, objek yang sudah tidak dibutuhkan lagi harus di-dealokasi oleh sistem pemroses bahasa tanpa perlu intervensi pemrogram
4. *Classes*, setiap type yang tidak sederhana adalah sebuah modul, setiap modul adalah type tingkat tinggi
5. *Inheritance*, sebuah Class dapat didefinisikan berdasarkan ekstensi atau restriksi dari kelas lain
6. *Polymorphism and dynamic binding*, entitas program harus dimungkinkan untuk mengacu kepada lebih dari satu kelas dan operasi harus dimungkinkan untuk lebih dari satu kelas
7. *Multiple and repeated inheritance*, harus dimungkinkan untuk membuat deklarasi kelas sebagai pewaris dari banyak kelas, dan lebih dari satu jika pewarisnya sebuah kelas

Karakteristik utama sistem beorientasi objek [Meyer-97]

- abstraksi
- enkapsulasi
- pewarisan (inheritance)
- reuseability
- spesialisasi
- generalisasi
- komunikasi antar objek
- polymorphisme

Pengembangan sistem dengan metoda OO dapat meningkatkan :

- produktifitas
- kecepatan pengembangan
- kualitas perangkat lunak
- kemudahan pemeliharaan

Catatan : pada kuliah ini sangat dibedakan aspek statik dan aspek dinamik (program pada saat run time)

Dinamika kehidupan objek : lahir (diciptakan), dimanipulasi/memanipulasi, mati (dihancurkan)

Objek merupakan model eksekusi

Objek mengalamai dynamic creation

- program mmenciptakan sejumlah objek menurut pola yang tidak mungkin diprediksi pada saat kompilasi
- pada saat operasi, objek baru mungkin saja diciptakan, reference sebuah objek diubah ke objek yang lain atau tidak lagi mengacu ke objek apapun
- nilai yang disimpan dalam sebuah field objek diubah

Perbedaan Kelas dan Objek

Adalah sangat penting untuk membedakan antara Kelas dan Objek . Objek adalah elemen pada saat runtime yang akan diciptakan, dimanipulasi dan dihancurkan saat eksekusi

Kelas adalah deskripsi statik dari himpunan objek yang mungkin lahir/diciptakan yang merupakan instansiasi dari Kelas

Pada saat runtime, yang kita punyai adalah objek. Di dalam teks program, yang kita lihat hanyalah kelas.

KELAS , definisi statik :

Pada lingkungan program berorientasi objek, pemrogram mendefinisikan kelas secara statik. Pada saat run time, kelas akan diinstansiasi menjadi objek (lihat dinamika). Objek yang merupakan instansiasi dari suatu kelas selalu dapat diacu lewat **Current Object**, apapun nama instantnya.

Untuk menjamin bahwa setiap instans yang lahir dari kelas sesuai dengan definisi kelas, bebrapa bahasa menyediakan fasilitas untuk menentukan class invariant, yaitu berupa asersi yang menjamin kebenaran objek.

Untuk menciptakan (menghidupkan) objek, diperlukan konstruktor. Untuk mematikan (menghancurkan) objek, diperlukan destruktor
Kelas mempunyai:

- **atribut** (data, konstanta, properti). Nilai atribut pada saat run time menyatakan "keadaan" (state) dari objek yang merupakan instans dari kelas ybs. Beberapa bahasa pemrograman mendefinisikan atribut harus sebuah Kelas, atau beberapa bahasa memperbolehkan atribut dideklarasi sebagai kelas atau type dasar (numerik: integer/float, character, boolean)
- **method** (service, prosedur, fungsi). Pada saar run time, method akan dieksekusi sesuai dengan kode programnya, atas permintaan (lewat pesan, message) objek lain. Method mempunyai spesifikasi, signature (nama dan parameter), dan mempunyai body (kode program yang akan dieksekusi). Signature adalah informasi bagi kelas yang akan menggunakan kelas ini, sedangkan body merupakan kode program yang akan dieksekusi. Sebenarnya body tidak perlu diketahui oleh kelas pemakai asalkan spesifikasinya jelas. Spesifikasi : Prekondisi (initial state), post kondisi (Final State) dan "proses" apa yang akan dikerjakan ketika mehod dieksekusi. Prekondisi dan Post kondisi dapat dituliskan sebagai asersi (pada beberapa bahasa), sedangkan proses dinyatakan dalam komentar. Parameter prosedur/fungsi dalam OOP selalu parameter input, tidak pernah ada parameter output atau parameter input/output:
  - **Fungsi dirancang untuk melahirkan sebuah objek baru (range, hasil) dari objek input. Parameter fungsi** selalu merupakan parameter input, karena fungsi akan memetakan semua objek input (domain) menjadi sebuah objek lain (range), tanpa mengganggu state dari objek input
  - **Prosedur dirancang untuk mengubah state dari Current objek, tanpa melahirkan objek baru. Parameter prosedur** selalu parameter input yang mewakili informasi perubahan dari Current Objek. Current objek merupakan parameter input/output secara implisit, sehingga tidak pernah dimunculkan dalam signature

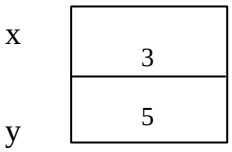
[Meyer-97] menyebut atribut dan method sebagai feature. Selanjutnya istilah feature akan dipakai dalam kuliah ini untuk mewakili atribut dan/atau method

Deklarasi Kelas

```
// Hubungan dengan kelas lain
Class X   // Nama Kelas
// feature: Atribut

// feature: Method

// invarian kelas
End Class X;
```

Simple Class : hanya mempunyai field

```
Class Point
// atribut
  x,y : integer
End class Point
```



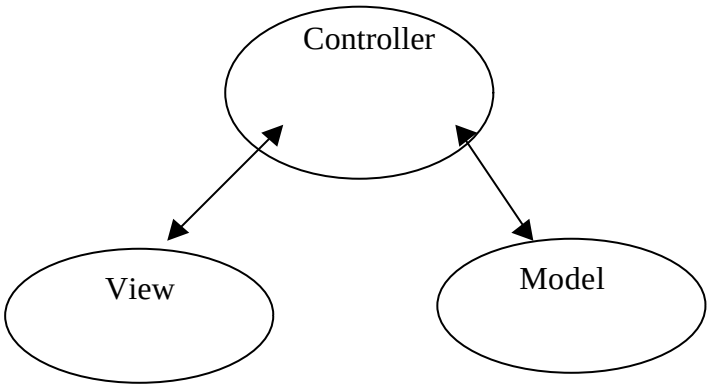
Lingkup akses terhadap Feature didefinisikan mulai dari yang umum sampai dengan yang sangat restriktif:

- **public** : dapat diakses/dipakai kelas apapun
- **friend**, hanya kelas tertentu yang boleh mengakses
- **private** : hanya kelas yang bersangkutan yang boleh memakai

Penentuan lingkup akses terhadap feature merupakan bagian yang harus dirancang. Sebenarnya, dalam perancangan yang murni OO, sebaiknya "friend" tidak digunakan

Klasifikasi kelas

Model program berorientasi objek yang termasuk paling "lama" dan mendasar adalah model MVC dari Smalltalk, dimana setiap aplikasi dipandang terdiri dari tiga jenis kelas : ***M (Modeler), V (Viewer), C(Controller)*** dengan hubungan sebagai berikut:



Modeler adalah representasi dari domain persoalan yang akan diprogram. ***Viewer*** bertugas untuk menampilkan objek "domain" sesuai dengan jenis objek ke antarmuka pengguna (misalnya di lingkungan GUI : layar/windows), sedangkan ***Controller*** bertugas untuk mengatur interaksi dan aliran data/pesan antara Modeler dan Viewer. Objek dilahirkan berdasarkan definisi kelas, dan ketika eksekusi program akan "hlang". Jika objek harus dapat disimpan secara permanen, maka modeler juga harus merepresentasi **persistent objek** (pada pemrograman prosedural menjadi arsip eksternal, ***external file***) .

Klasifikasi kelas dari sudut pandang instansiasinya:

- kelas “biasa”, instansnya adalah objek, siap dipakai semua featurenya
- kelas abstrak : instance bukan objek.
- deferred class : kelas yang belum seluruhnya diimplementasi

Klasifikasi kelas dari sudut pandang Booch:
- ADT : definisi type dan method
- Mesin : punya state dan behavior, pasif
- Proses: objek aktif

Klasifikasi kelas dari sudut pandang UML:
- Boundary entity
- Domain entity
- Controller

Pada metodologi ini, , boundary entity merupakan objek-objek yang menjadi antarmuka interaksi, domain entity merepresentasi persoalan yang dimodelkan (pada saat runtime menjadi objek) sedangkan controller adalah objek pengendali yang mengendalikan semua objek yang ada.

Klasifikasi kelas dari sudut pandang Coadd:
- Domain problem
- Interface

Klasifikasi "kelas" pada bahasa Java
- CLASS
- Interface

Definisi-definisi dan jenis CLASS dari berbagai sudut pandang dapat dilihat pada lampiran B.

Hubungan antar kelas

Antara sebuah kelas dengan kelas yang lain ada hubungan : Client-Supplier atau inheritance.

Hubungan Client-Supplier

Pada hubungan **Client-Suplier**, sebuah kelas Client memakai kelas Supplier. Hubungannya adalah hubungan “kontrak”. Supplier menyediakan sejumlah services yang dapat dipakai oleh Client, dan menjanjikan akan memenuhi “kontrak”, yaitu memenuhi prekondisi yang ditentukan. Client wajib mentaati aturan (prekondisi) yang tertulis sebelum memakai services yang disediakan oleh Supplier.
Hubungan yang lebih simple, sederhana ini lebih disarankan untuk dipakai!

Definisi :
Kelas A adalah **Client** dari kelas B dan B adalah **Supplier** dari Kelas A jika A mengandung definisi entitas e: B
Entitas adalah :
- atribut
- argumen formal dari rutin
- hasil evaluasi fungsi

Hubungan inheritance

Pada hubungan **Inheritance**, sebuah kelas turunan (descendant, heir, child,…) mewarisi kelas leluhur (parent, …). Karena mewarisi, maka “semua” atribut dan method kelas bapak akan “dibawa”, secara intrinsik menjadi bagian dari kelas anak. Dalam beberapa keadaan, membawa secara intrinsik semua atribut dan method tidak dikehendaki. Maka pemroses bahasa menyediakan sarana untuk:

- menambah feature baru,
- mengubah atau menggantikan feature yang diwarisi ,
- menghapus feature yang diwarisi,
- menentukan feauture yang masih deferred (belum terdefinisi)

Ini menimbulkan persoalan yang tidak sederhana. Karena penghapusan menimbulkan beberapa konsekuensi berbahaya, maka sedikit sekali metodologi/bahasa yang membolehkan penghapusan feature yang diwarisi.

***Feature visibility* dalam sebuah kelas ("scope" dalam pemrograman prosedural):**

- **private**, feature yang "visible" hanya untuk kelas dimana feature tersebut didefinisikan
- **protected**, feature yang "disembunyikan" terhadap kelas Client, tetapi "visible" untuk kelas turunan.
- **public**, feature yang "visible", dapat dipakai oleh semua kelas lain (Client, turunan).
- **friend**: dalam bahasa C++, memberikan hak akses ke kelas-kelas tertentu untuk dapat mengakses semua feature yang sebenarnya private. Ini merupakan pelanggaran terhadap prinsip information hiding. Jadi beberapa feature yang sebenarnya "private" diijinkan untuk menjasi "visible" hanya untuk kelas tertentu. Sebenarnya pemakaian friend tidak terlalu dianjurkan karena menyalahi kaidah inkapsulasi.

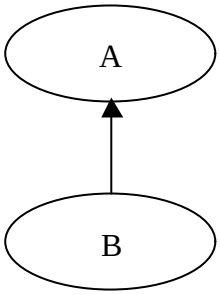
Visibility dari feature yang diwarisi pada kelas turunan sehubungan dengan inheritance:

Private inheritance : semua feature yang diwarisi menjadi private dalam kelas anak, tanpa peduli visibility pada Parent. Pemakaiannya mendukung information hiding dan maintainability dari feauture yang diwarisi karena dengan cara ini, perubahan pada ancestor tidak mempengaruhi anak. Tetapi, penggunaannya harus hati-hati sebab menimbulkan nonconformity terhadap parent, sebab apa yang visible di parent : menyalahi subtyping, spesialisasi dan substitusi polimorfik.

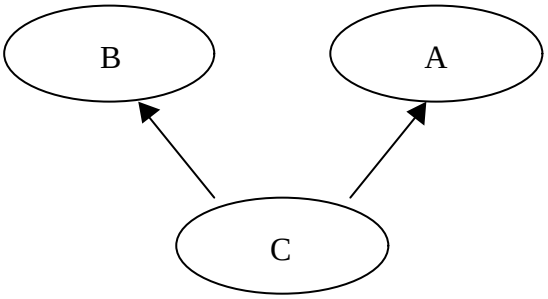
Protected inheritance : semua feature yang mempunyai visibility protected pada parent, tetap protected pada Child, sedangkan yang public pada parents, menjadi protected pada Child.

Public inheritance : semua feature yang diwarisi mempunyai visibility yang sama dengan visibility pada parent (jika public maka tetap public; jika private tetap private; dst)

Jenis inheritance :
Single inheritance : sebuah kelas turunan merupakan turunan dari sebuah kelas bapak. Jika simbol lingkaran/elips merupakan simbol sebuah kelas, maka hubungan inheritance digambarkan sebagai berikut [Meyer-97]. Arti dari gambar tersebut: B mewarisi A. B adalah turunan dari A.



Multiple inheritance :sebuah kelas turunan mewarisi lebih dari satu kelas bapak (Join)

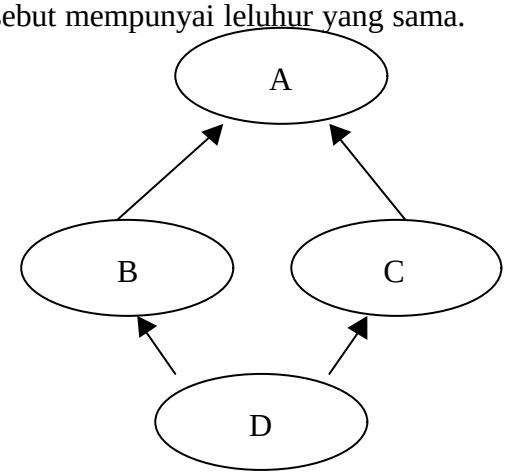


Multiple inheriance menimbulkan beberapa persoalan : jika ternyata ada feature di kelas-kelas leluhur yang ternyata "konflik". Konflik yang terjadi mungkin adalah :

- konflik nama
- konflik "body" (untuk method)

Tidak semua bahasa mensupport repeated inheritance, tetapi menyediakan mekanisme lain untuk merealisasi konsep ini.

Repeated inheritance : sebuah kelas turunan mewarisi lebih dari satu kelas bapak, dan kelas bapak tersebut mempunyai leluhur yang sama.



Repeated inheritance menimbulkan beberapa persoalan :

- Konflik pada D, seperti halnya multiple inheriance.
- konflik pada D, jika ternyata beberapa feature di B dan C sudah dimodifikasi

Sangat sedikit bahasa yang mensupport repeated inheritance.

Replicated repeated inheritance : Sebuah feature yang diwarisi dari common anscestor dengan nama yang sama menjadi lebih dari satu buah feature dalam Current Class. Jadi dalam hal ini, feature yang diwarisi muncul lebih dari satu kali. Dalam beberapa bahasa, permogram harus mengubah nama sehingga feature menjadi unik.

Shared repeated inheritance (virtual inheritance) : Sebuah feature yang diwarisi dari common anscestor dengan nama yang sama menjadi hanya satu buah feature dalam Current Class. Jadi dalam hal ini, feature yang diwarisi muncul hanya satu kali, dan dipakai bersama.

Pemakaian inheritance perlu dikaji secara baik, dan dirancang dari awal. Perancangan inheritance yang tambal sulam dan tidak tepat akan menimbulkan banyak kesulitan pada saat implementasi. Pada “top level” inheritance, biasanya dibuat kelas abstrak, yang merupakan spesifikasi dari kelas-kelas turunannya. Makin ke “bawah”, definisi kelas menjadi makin spesifik, dan dapat diinstansiasi menjadi objek yang “jelas”. [Meyer-97] bahkan mendefinisikan hubungan inheritance dalam beberapa tipologi.

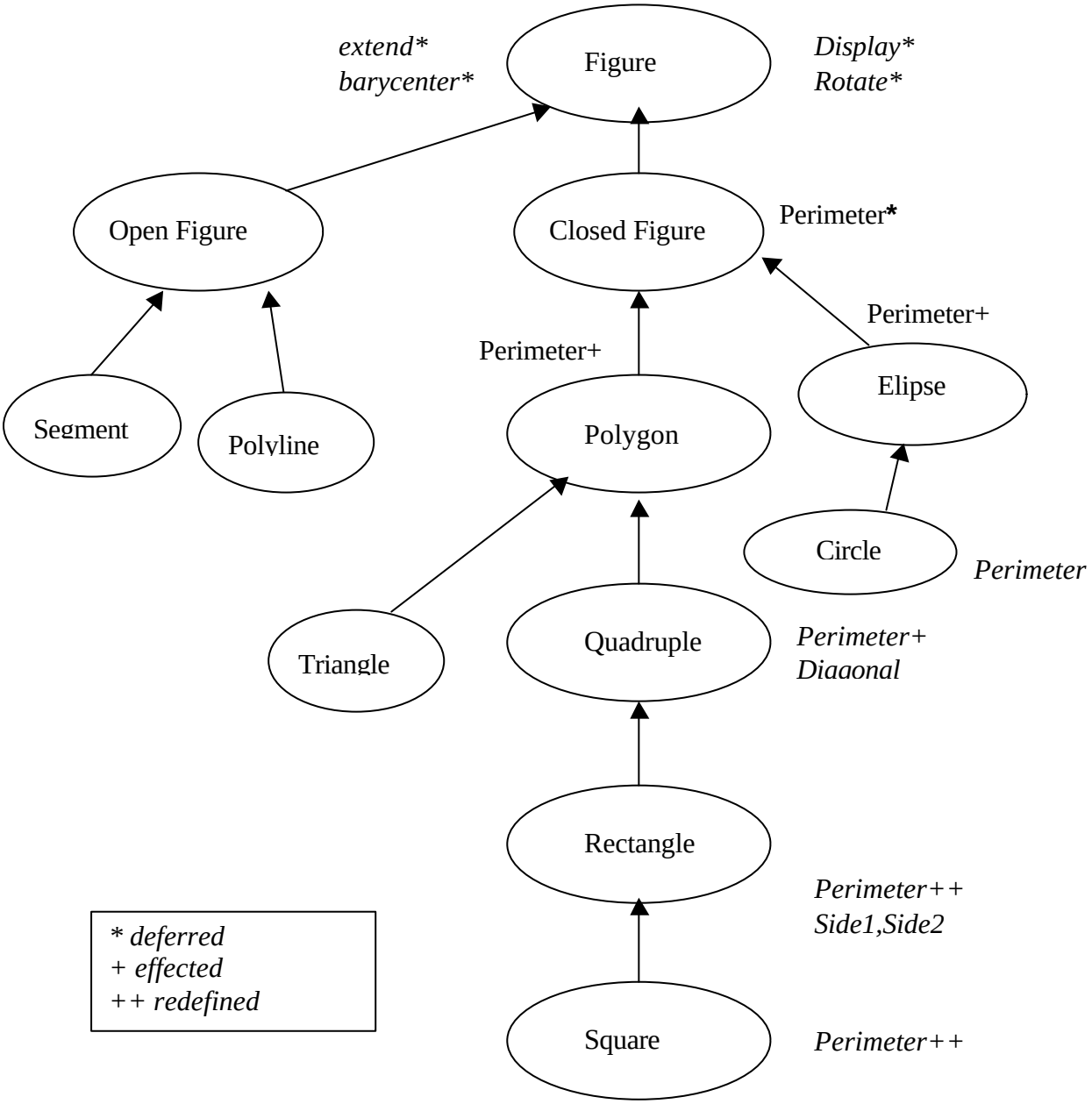
Buku [Webster-95] menyebutkan tiga macam hubungan antar kelas, yaitu : **has, is-a, is-implemented-using**. Hubungan **has** dan **is-implemented-using** sering dikacaukan menjadi hubungan inheritance.

Hubungan **is-a** adalah hubungan antara kelas general (umum) dengan subkelas yang lebih spesifik. Ingat bahwa objek pada subkelas harus tetap merupakan objek superkelasnya. Contoh : reptil adalah binatang. Perhatikanlah instans dari kelas tsb: seekor cicak adalah reptile, tetapi belum tentu semua reptile adalah cicak. Demikian pula dengan kelas : reptile termasuk binatang, tidak semua binatang adalah reptile. Hubungan is-a adalah konsep **inheritance**.

Hubungan **has-a (mempunyai)** merefleksikan keseluruhan dengan komponennya. Contoh : Sebuah mobil mempunyai mesin, mempunyai roda, .. dst. Pemula dalam bidang OO sering mengimplementasikan hubungan ini dengan inheritance. Yang lebih benar adalah : Kelas Mesin, Roda merupakan data member dari kelas Mobil.

Hubungan is-**implemented-using (diimplementasi menggunakan)** termasuk di antara kedua hubungan itu. Contoh : sebuah Daftar Nomor Telpon (TelponList) diimplementasi menggunakan Notebook. Hubungan itu bukan inheritance, dan juga bukan bagian dari. Ada banyak cara untuk melakukan impelementasi dari Daftar nomor telpon. Bahkan perancang OO yang berpengalaman banyak yang mengimplementasi TelponList sebagai turunan dari Notebook. Sebaiknya TelponList "memiliki" notebook, berarti Notebook adalah data member dari telpon List.

Contoh “pohon” inheritance [Meyer-97]



Pengertian Root Class Dan Main Program

Root Class adalah kelas yang merupakan awal dari semua kelas (awal penciptaan)
Main program adalah titik awal eksekusi. Eksekusi dari sistem OO terdiri dari dua tahap:

- Penciptaan sejumlah objek, yang disebut root objek
- Aplikasi prosedur tertentu, yang disebut creation dari objek tsb

Main program menggabungkan kedua konsep di atas

- titik awal eksekusi
- top, fundamental architecture dari sistem

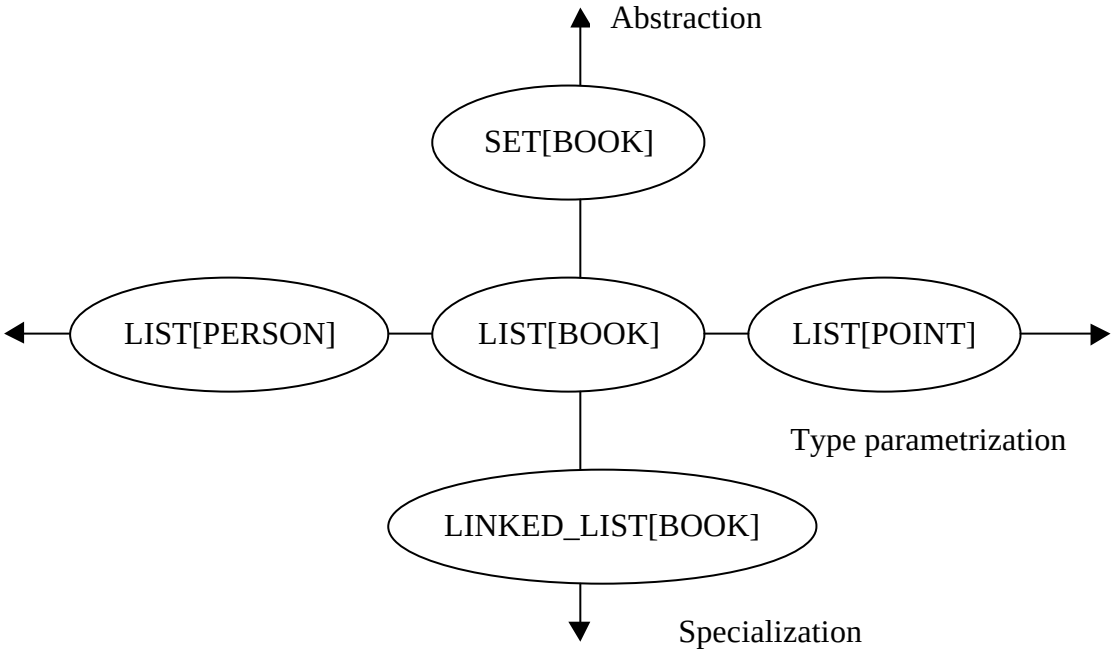
Pada implementasi menjadi sebuah program, kelas atau beberapa kelas dikelompokkan menjadi satu atau beberapa file. Definisi dari file dimana kelas disimpan harus dimengerti oleh pemroses bahasa. Setiap bahasa mempunyai cara tersendiri. Conoth : dalam bahasa JAVA, dapat dibuat PACKAGE. Dalam bahasa C++, didefinisikan file header dan body seperti dalam bahasa C. Dalam bahasa Eiffel, peta dari peletakan kelas dalam file didefinisikan dalam SYSTEM
Assembling dari sistem adalah menyebutkan nama root class, sebuah univers atau sekumpulan file yang mungkin mengandung definisi kelas.
Dalam beberapa sistem, “dunia” (universe) dari sistem dibangun sedikit demi sedikit. Dalam bahasa lain, harus disebutkan secara ekspilist.
Pada setiap bahasa berorientasi objek, biasanya tersedia cara untuk mendefinisikan “dunia” tersebut.

Overloading

Overloading : kemampuan suatu nama untuk diasosiasikan ke lebih dari satu arti terhadap nama yang muncul dalam program. Fasilitas ini biasanya disediakan dalam suatu lingkungan berorientasi objek.
Contoh : prosedur dengan nama yang sama, namun parameternya berbeda dimungkinkan untuk dieksekusi dalam program berbahasa Ada dan C++. Contoh yang sangat berguna dalam program, adalah tersedianya lebih dari satu macam konstruktor untuk menciptakan objek sesuai keperluan.

Genericity

Genericity bukan hanya dikenal dalam OOP, tetapi memegang peranan penting dalam OOP. Contoh: Pada bahasa Ada, dikenal generic type dan generic procedure/function. Dalam konsep generik, suatu type atau method baru dikenal ketika runtime, maka pada kode program secara "statik", baru dikenal namanya secara generik.
[Meyer] melukiskan hubungan generik sebagai hubungan "horizontal", dimana instans dari sebuah kelas generik bisa menjadi ber-macam-macam sesuai denganinstansiasi yang dilakukan. Sedangkan hubungan inheritance dilukiskan sebagai hubungan "vertikal", dimana sebuah kelas anak mewarisi feature leluhur.
Dalam bahasa C++, pengertian Generic dituangkan dalam TEMPLATE.



OBJEK : definisi run time, Dinamika

Penciptaan dan manipulasi objek

- Ketika objek diciptakan : dibentuk container, address dan inisialisasi nilai (lewat konstruktor)
- Mengubah reference dari sebuah objek
- State dari reference (mengacu ke objek yang mana)
- Forget dan storage management
- Inisialisasi nilai objek ketika diciptakan
- Mengapa harus ada create, clone, deep clone ? karena hanya dengan deklarasi referens tidak tahu mengacu ke siapa
- Accessing field sebuah objek : dikenal konsep Current_Obj

Prinsip : program (pada saat run time) dimulai dari sekumpulan objek yang "dihidupkan". Setiap Objek mempunyai *state* dan kelakuan *(behaviour)* yang telah didefinisikan. Objek saling berinteraksi dengan saling mengirimkan message. Yang memicu dinamika : objek mengirimkan message ke objek lain, ada "event" yang menyebabkan suatu objek bereaksi dengan objek lain. Akibat inreraksi : ada objek baru menjadi hidup, atau sejumlah objek mati. Program selesai sesuai dengan skenario yang diprogram. Berbeda dengan berpikir secara prosedural, paradigma pemrograman berorientasi objek mengharuskan pemrogram tidak lagi berpikir sekuensial. Pemikiran pemrogram tidak lagi "sekuensial", karena setiap objek mungkin aktif, dan sekaligus ada banyak objek yang "hidup". Hubungan antar objek aktif dapat ditentukan sesuai dengan modus yang tertentu, misalnya Single executor, Master-slave, Client-Supplier, Watch dog, …???

Objek pasif versus aktif :

Objek dapat merupakan sebuah objek pasif, yaitu objek yang dinamika eksekusinya ditentukan oleh objek lain. Objek ini baru dieksekusi jika diperintahkan oleh (menerima pesan dari) objek lain.
Objek aktif adalah objek yang mempunyai *"thread of control"* sendiri[Booch-95] . Dalam istilah UML, objek pengendali adalah salah satu contoh dari Objek aktif ini.

Komunikasi antar objek :

- Walaupun ada sejumlah objek, hanya ada **satu objek aktif** (*main program*), maka program akan dieksekusi secara **sekuensial**. Kehidupan dari semua objek ditentukan oleh sebuah objek aktif tersebut. Jika objek aktif mengirimkan pesan ke sebuah objek lain, maka pesan dieksekusi, dan setelah eksekusi berakhir, kendali berada dalam kuasa objek aktif tersebut.
- Ada **lebih dari satu objek aktif**, tetapi **independent**. Kehidupan suatu objek tidak "mempengaruhi" objek lain. Karena masing-maing objek mempunyai atribut dan method, maka tidak saling mengganggu. Program adalah sebuah program **konkuren sederhana** tanpa komunikasi/interaksi. Program adalah sebuah program konkuren sederhana. Jika bahasa mensupport konkurensi, maka tidak ada hal yang dilakukan oleh pemrogram
- Ada **lebih dari satu objek aktif**, dan objek saling "**dependent"**. Objek saling berkomunikasi lewat objek komunikasi (misalnya *mail box, shared memori*, …).

Eksekusi dari sebuah services milik sebuah objek yang dikendalikan objek lain harus mengalami sinkronisasi lewat protokol tertentu. Program adalah **program konkuren dengan mekanisme komunikasi/sinkronisasi**.

Constructor & Desctructor

Objek dilahirkan (diciptakan, dihidupkan) berdasarkan definisi kelasnya. Maka setiap kelas harus mempunyai prosedur yang memungkinkan penciptaan objek (creation procedure, creator, konstruktor). Ada kreator yang secara otomatis "dihidupkan" oleh pemroses bahasa, ada kreator yang dikendalikan oleh pemrogram.
Pada saat dihidupkan, maka yang terjadi adalah:

- dibuat reference terhadap objek
- dilakukan alokasi untuk penyimpan informasi objek (hati-hati sebab beberapa bahasa tidak melakukan)
- dilakukan inisialisasi field-field sesuai dengan aturan inisialisasi bahasa.

Creation: Penciptaan objek (konstruktor) pada bahasa Eiffel :

Instruksi creation: `!!x,` dengan type x adalah reference ke C, efeknya adalah

- Create sebuah instansiasi dari C (terdiri dari field yang didefinisikan dalam C). Instansianya disebut OC
- Inisialisasi setiap field OC tergantung kepada nilai default yang ditentukan
- Melakukan Attachment x ke OC

Selain default creation tsb, dalam sebuah kelas dapat didefinisikan creator yang lain

```
Class C
   creation
       P1,P2 ….
// realisasi dari feature P1 dan P2
End Class C
```

Dengan definisi di atas, maka dapat dilakukan `!!x.p()` yang berefek:

- Create sebuah instansiasi dari C (terdiri dari field yang didefinisikan dalam C). Instansianya disebut OC
- Inisialisasi setiap field OC tergantung kepada nilai default yang ditentukan
- Melakukan Attachment x ke OC
- Memanggil prosedur P dengan argumennya terhadap OC

Aturan creator dalam bahasa Eiffel : `!!x` dan `!!x.p` adalah mutual exclusive
Dalam bahasa Java dan C++, ada ***default constructor*** dan konstruktor lain yang didefinisikan oleh pemrogram.

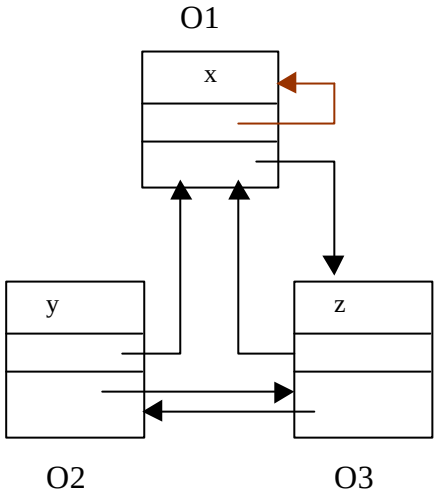
Reference : Pengenal Objek

Pengertian **reference:** pengenal objek, *runtime value* yang nilainya ***void*** atau ***attached***. ***Void*** artinya tidak diasosiasikan ke objek apapun (yang ada alokasi memorinya). ***Attached*** artinya sudah dialokasi memorinya.
Jika attached, maka sebuah reference mengidentifikasi sebuah objek (dikatakan bahwa reference tsb di-attach ke sebuah objek tertentu)

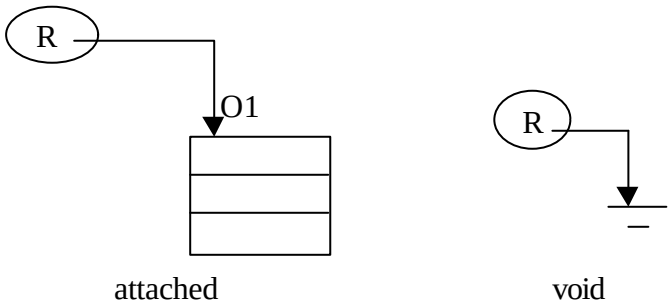
Deklarasi reference:
Jika terdefinisi sebuah kelas C, maka semua deklarasi `x:C`
akan menunjukan akan ada reference pada saat *run time* ke objek bertype C

Reference dalam konteks prosedural dapat dianalogikan dengan “pointer”, alokasi dinamik. Dalam bahasa Java dan Eiffel tidak ada deklarasi pointer. Setiap objek pasti mempunyai reference. Sedangkan dalam bahasa C++, objek dengan deklarasi “Pointer ke” yang akan mempunyai reference.

Self reference :
Sebuah objek O1 yang mengandung reference ke dirinya sendiri



Jika notasi segi empat digunakan sebagai sebuah objek, maka pada saat run time kondisi reference adalah salah satu di natra dua kemungkinan sebagai berikut mengacu ke sebuah objek (**attached**), atau berstatus **void (Null)**



R adalah sebuah reference (pointer), yang dipakai untuk mengenali objek O1. Objek O1 untuk menyimpan informasi harus mempunyai container (dialokasi, diciptakan).

Manipulasi yang penting terhadap Objek (secara keseluruhan)
Operasi yang penting pada saat runtime adalah :
- operasi pembandingan/kesamaan ,dan
- operasi assignment.

Operasi pembandingan/kesamaan : Apakah Objek O1 sama dengan O2 ?

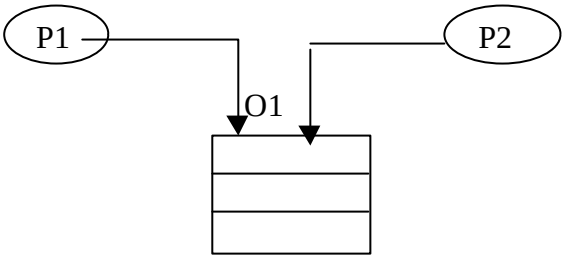
Pembandingan

Jika x dan y adalah entitas bertype reference, maka kesamaan x dengan y : x=y akan bernilai true jika :

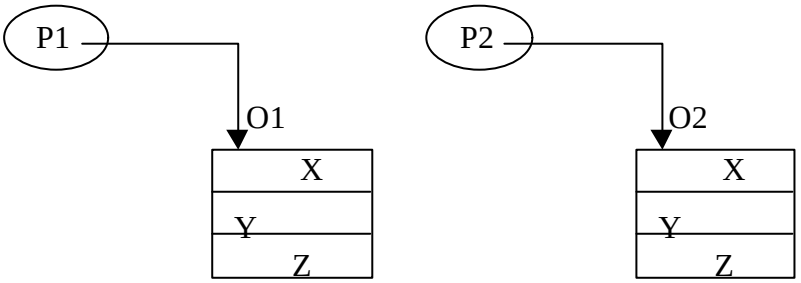
- keduanya void, atau
- attached ke objek yang sama

Arti pembandingan:

- dapat berarti membandingkan apakah O1 dan O2 adalah objek yang sama, artinya dua buah reference mengacu ke objek yang sama ?



- dapat berarti membandingkan apakah O1 dan O2 adalah objek yang sama, artinya dua buah objek adalah identik kandungan informasinya ? Kasus menjadi rekursif, jika X,Y atau Z adalah sebuah Objek dan bukan type biasa.



Bahasa Eiffel mendefinisikan operasi perbandingan antarobjek dengan tabulasi sebagai berikut :

Comparison : `x = y`

Type of y / **Type of x**	**Reference**	**Expanded**

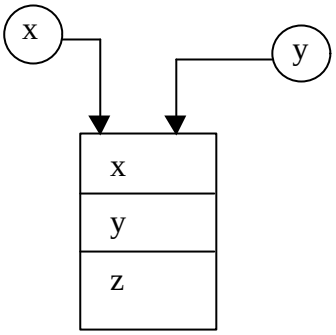
Reference	Reference comparison	If x is non-void then
		  Equal(x,y)
		Else
		  False

expanded	If y is non-void then
	  Equal(x,y)
	Else
	  False
		Equal(x,y), i.e. object comparison

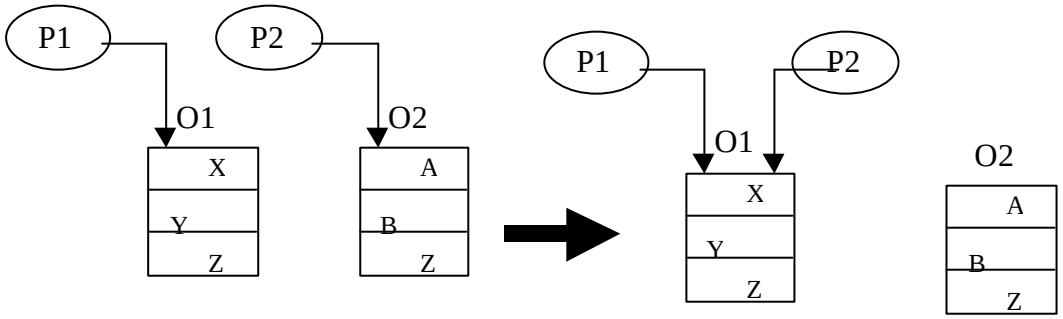
Operasi Assignment

Dynamic aliasing

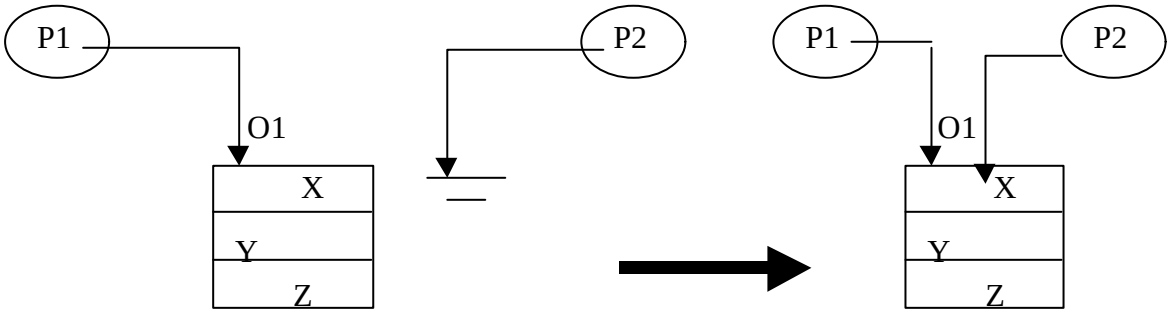
Jika x dan y adalah reference, maka x:=y menyebabkan x dan y di-attached ke objek yang sama. Dikatakan terjadi dynamic aliasing. Hati-hati, sebab x.f akan sama efeknya dengan y.f



Operasi assignment : P2 := P1, jika P1 dan P2 adalah reference yang sudah mengacu ke objek:



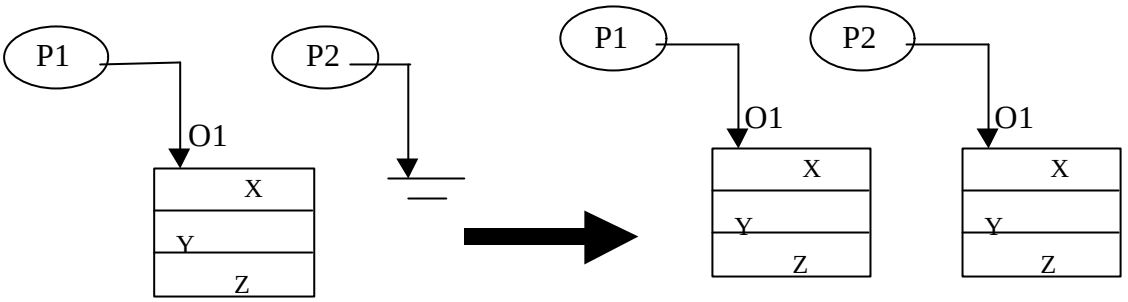
atau dalam kasus kedua, jika P2 adalah reference yang masih belum di-attache ke objek manapun :



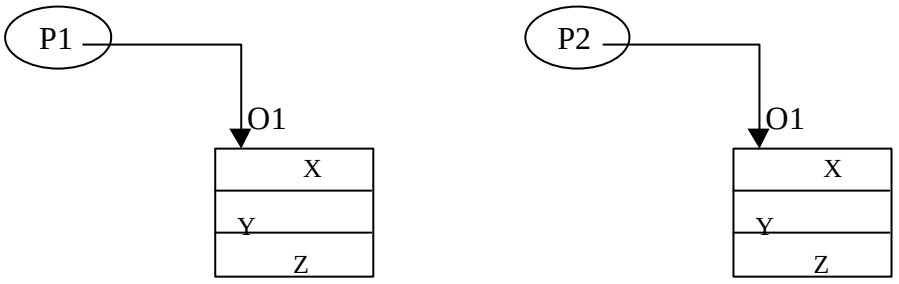
Assignment sebagai proses COPY (menyalin Objek)

Perhatikanlah “assignment” sebagai berikut: `P2:=Copy(P1)`
Masalah : apakah P2 sudah dialokasi ???

(a)

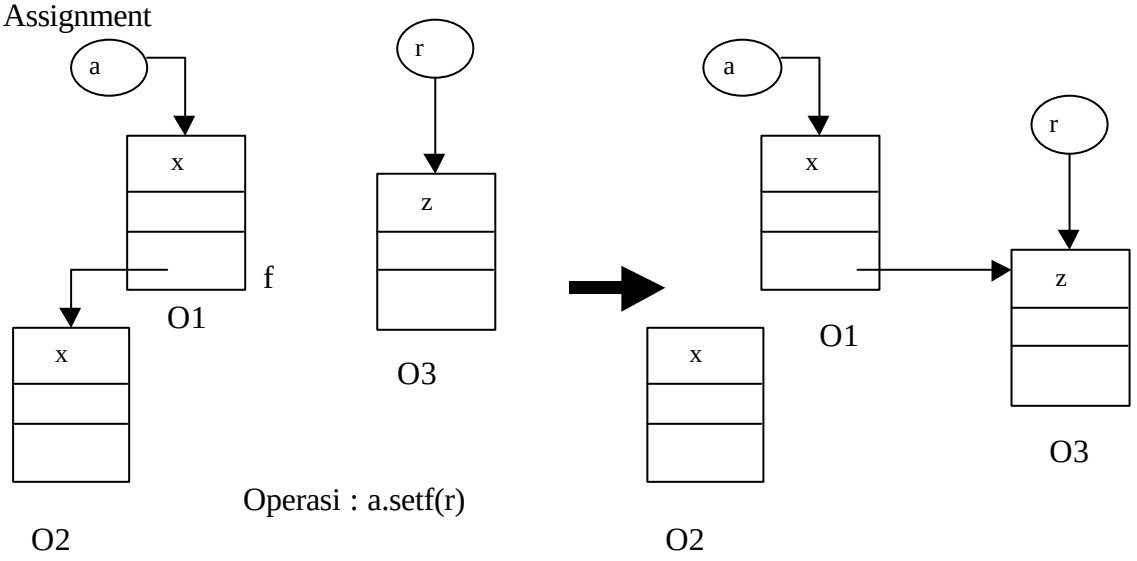


(b)

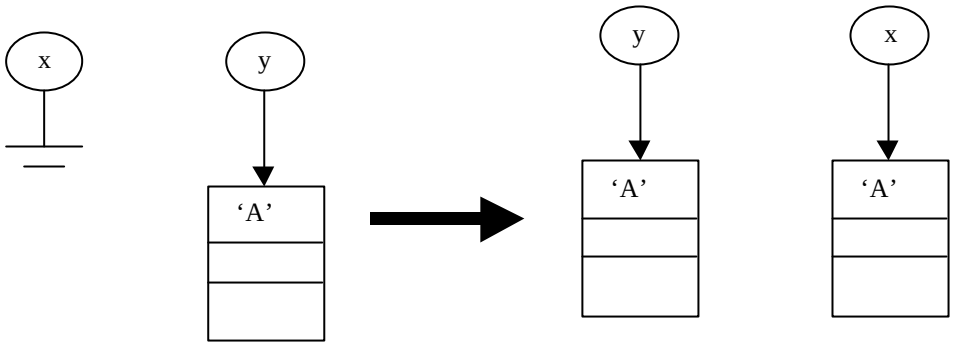


Operasi terhadap reference

Assignment



Cloning : `x:= clone(y)`
Clone menciptakan carbon copy (salinan) objek.

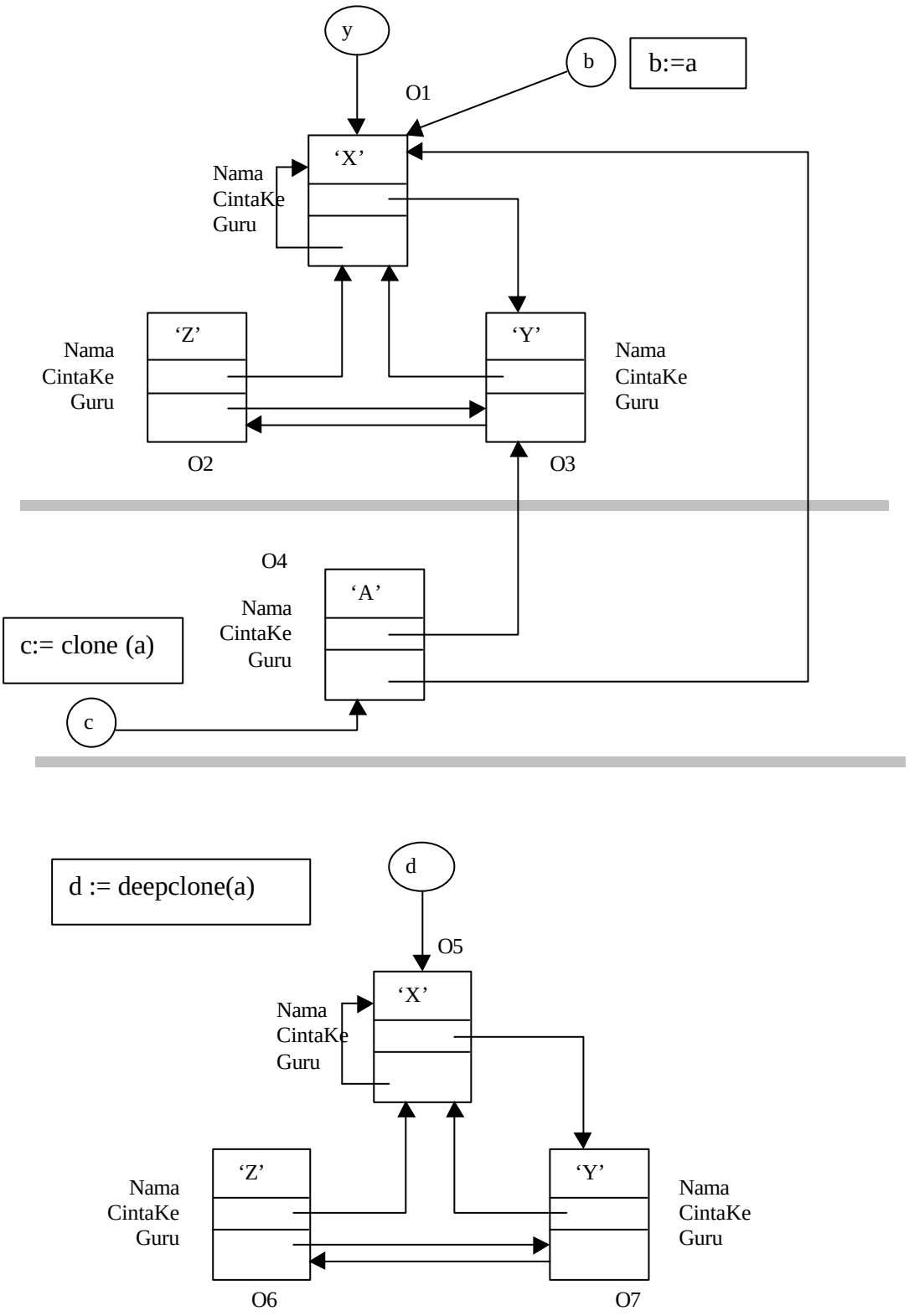


Before

After

Copy : `x.copy(y)`
Akan menyalin y ke dalam x. Copy tidak menciptakan carbon copy dari objek. Hanya menyalin fields (tidak melakukan alokasi container baru)
Pada Copy, target (untuk kasus di atas adalah x) tidak boleh void

Deep Clone : `x:= deepclone(y)`



Beberapa bahasa menyediakan semua operasi tersebut, bahasa lain tidak. Beberapa bahasa memakai istilah : shallow copy, deep copy. Pelajarilah mekanisme "assignment" pada setiap bahasa tersebut.

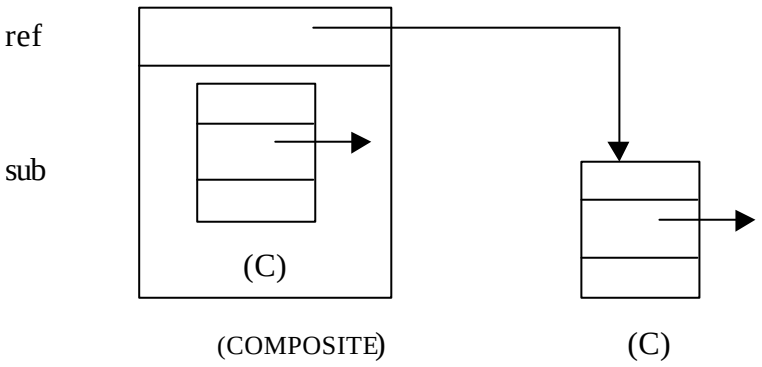
Bahasa Eiffel bahkan membolehkan operasi assignment sebagai berikut :

Attachment : `x := y`

Type of source y / **Type of target x**	**Reference**	**Expanded**
Reference	Reference attachment	X:=clone(y)
expanded	x.copy(y), fail if y is void	x.copy(y)

Composite object: mengandung expanded type
Expanded type :
`X : expanded C`

```
Class COMPOSITE
  Ref: C
  Sub : expanded C

//

End Class COMPOSITE
```



Peran expanded type :
- Menambah efisiensi
- Pemodelan yang lebih baik
- Basic type pada OO type system

Aturan *creation* pada reference yang menyatakan Creation (C1), kemudian Inisialisasi (C2) dan Attachment(C3).
Untuk expanded type, aturan C1 tidak sesuai lagi, dan aturan C3 tidak perlu, maka efeknya hanya inisialisasi ke nilai default.

Pemusnahan Objek Dan Garbage Collection

Pada saat run time, objek dihidupkan dan menempati memori. Objek yang sudah tidak dibutuhkan lagi seharusnya dimusnahkan supaya tidak menempati ruang memori, dan memori dapat dialokasi untuk objek lain . Untuk itu harus dilakukan pengancuran objek

Pada beberapa bahasa pemrograman (misalnya C++), adalah tugas dari pemrogram untuk melakukan pemusnahan objek yang sudah tidak dibutuhkan.
Pada beberapa bahasa yang lain (misalnya Eiffel, Java), mekanisme pemusnahan dilakukan oleh eksekutor bahasa. Objek yang sudah tidak dibutuhkan lagi dapat dikenali karena referencenya sudah tidak ada lagi. Pemusnahan oleh eksekutor bahasa dapat dilakukan dengan dua cara: reference counting atau garbage collection.
Ketersediaan mekanisme pemusnahan objek oleh eksekutor bahsa membebas pemrogram dari kewajiban memikirkan ruang memori, tetapi di lain pihak biasanya akan memberikan penalti terhadap waktu eksekusi.
Jika pemusnahan objek dilakukan oleh pemrogram, pemrogram sudah tahu pasti objek mana yang masih dibutuhkan atau tidak dibutuhkan lagi. Maka pemusnahan dapat dilakukan sesegera mungkin
Jika pemusnahan objek dilakukan oleh eksekutor bahasa, maka eksekutor harus memeriksa setiap saat keadaan objek di ruang memori. Ini akan memakan waktu/CPU sehingga dapat mempenalti waktu eksekusi jika algoritma eksekutor tidak efisien.

Exception handling

Idealnya, semua kesalahan program dapat dideteksi pada saat statik (sebelum eksekusi). Namun, kesalahan pada saat run time tidak mungkin dihindari. Kesalahan pada saat Eksekusi menimbulkan kegagalan program, yang jika tidak ditangani akan menyebabkan program abort. Beberapa bahasa pemrograman menyediakan mekanisme untuk menangani kesalahan yang terjadi pada saat runtime [Meyer-97] melakukan klasifikasi terhadap error pada saat run time yang mungkin ditangani dalam bahasa Eiffel. Setiap kasus yang ditangani oleh suatu bahasa biasanya diberi identitas yang memungkinkan pemrogram untuk "menangkap" error yang terjadi dan mendefinisikan tindakan yang sesuai.

Ringkasan tentang eksepsi

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

ASERSI

- Asersi : ekspresi boolean, ekspresi dari sifat (property) semantik kelas, menyatakan aksioma dan prekondisi dari class ybs
- Asersi dipakai pada prekondisi. postkondisi dan ivarian 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
- 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 kebernaran dari kelas
- Asersi mempunyai 4 kegunaan : bantuan menghasilkan program yang benar, dokumentasi, debugging aid, basis dari exception mechanism

INHERITANCE

Inheritance adalah konsep yang fundamental dalam OO
Inheritance mempunyai konsekuensi lanjut :
- multiple inheritance
- renaming
- sub contracting
- pengaruh dalam typing system

Inheritance harus digunakan dengan baik, dan menghindari penggunaannya yang salah.
Lihat klasifikasi penggunaan inheritance di akhir bab

Beberapa istilah sehubungan dengan inheritance

Parent	heir
Ancestor	descendant
Base class	derived class
Leluhur	turunan

Aturan pewarisan :
Sebuah turunan dari kelas C adalah semua kelas yang mewarisi (secara langsung atau tidak langsung) kelas C, termasuk kelas C sendiri. Jadi secara formal: adalah kelas C sendiri atau secara rekursif : turunan dari kelas C.
Turunan sesungguhnya dari C adalah semua turunan C kecuali C sendiri
Orangtua dari C adalah kelas A sehingga C merupakan turunan dari A. Orangtua sesungguhnya dari C adalah kelas A sehingga C adalah turunan dari A

DEFERRED FEATURE & CLASS

Deferred feature dan class adalah sarana yang dibutuhkan untuk abstraksi
Deferring versus effecting : Deferred pada saat desain, efektif pada saat imple:mentasi.
Maka sebuah deferred class mengandung deferred feature, yaitu feature yang baru "diketahui" pada saat deklarasi, dan diimplementasi oleh turunannya. Pada saat didefinisikan itulah menjadi efektif. Proses untuk membuat sesuatu yang deferred menjadi efektif disebut efecting.

Redeclare sebuah feature adalah mendefinisikan atau membuatnya efektif.

Redeclaring From →	**Deferred**	**Effective**
↓		
Deferred	`Redefinition`	`Undefinition`
Effective	`Effective`	`Redefinition`

Deferred class :

- instansiasinya bukan objek
- Creation tidak boleh deferred

Explicit redefinition memungkinkan readability dan reliability

Class:

- Extension
- Specialization
- Combination

REDEFINE

Seringkali sebuah kelas turunan harus mendefinisikan ulang feature dari ancestor sebab harus disesuaikan dengan kondisi si turunan. Fasilitas REDEFINE memungkinkan sebuah kelas turunan untuk mendefinisikan ulang feature kelas ancestor

Contoh

```
Class Polygon
// atribut
    Count : integer
    NbOfVertices : integer

// Akses
   function perimeter return real

// Transformasi
   procedureDisplay
   procedure rotate
   procedure translate

// feature: atribut  implementation
   Vertices :linked_list[Point]

// Invariant kelas
   invariant
   at_least_three :count>3

End Class polygon;
```

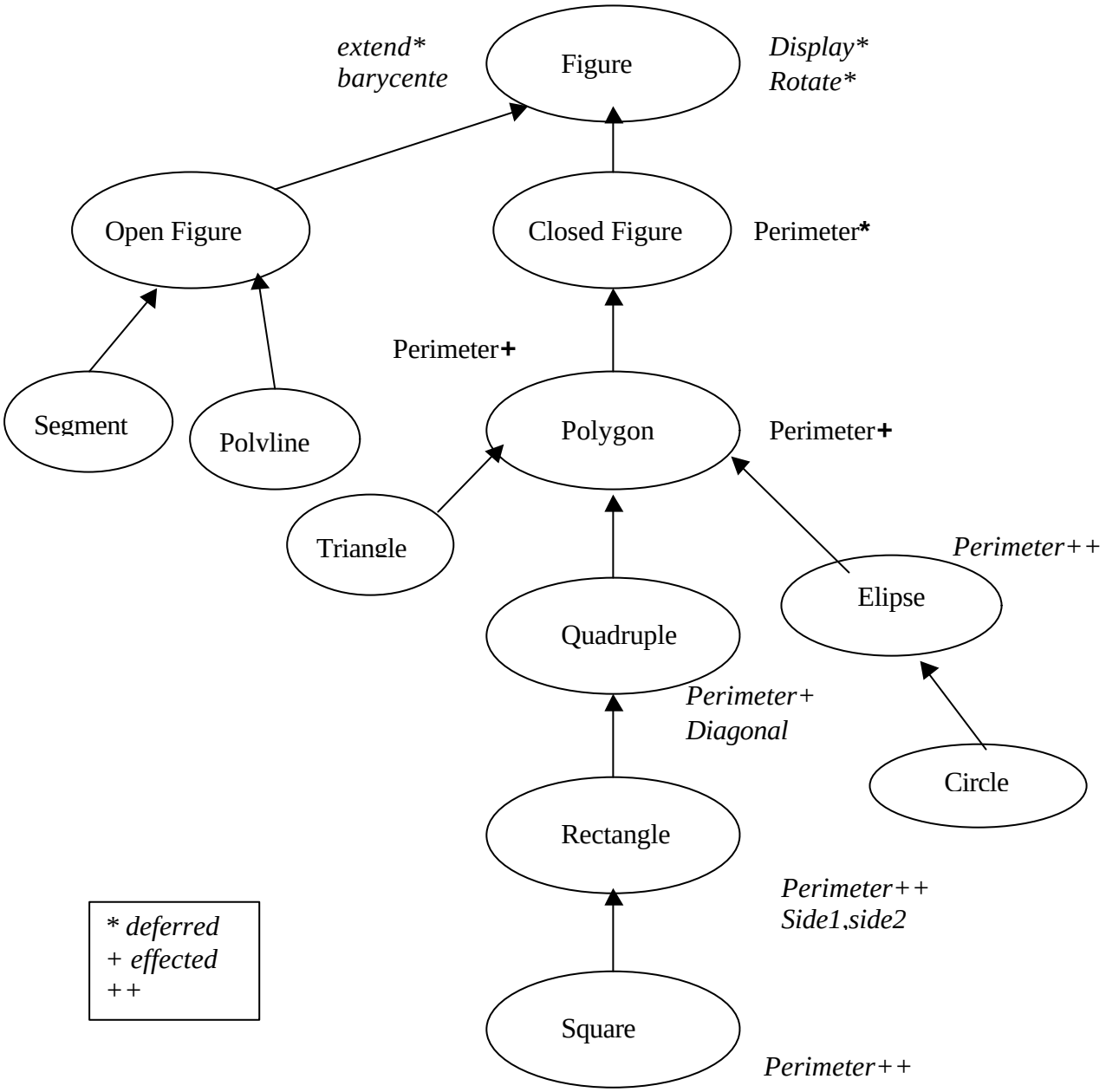
```
Class Rectangle
    Inherit polygon

// Atribut pesifik reactangle
   function perimeter return real
// Method specific
End Class rectangle;
```

Karena perimeter untuk rectangle khusus, maka dituliskan :

```
Class Rectangle
    Inherit polygon
       Redefine perimeter
```

```
// Atribut pesifik reactangle
// Akses
    side1, side2 : real;
    diagonal : real
// Method specific
   function perimeter return real is … do end;
// Invariant kelas
    four_sides : count =4
    First_side: vertices-I_th(1).distance(vertices-I_th(2))=side1
    Second_side: vertices-I_th(2).distance(vertices-I_th(3))=side2
    Third_side: vertices-I_th(3).distance(vertices-I_th(4))=side3
    Fourth_side: vertices-I_th(4).distance(vertices-I_th(1))=side4
End Class rectangle;
```

Berikut ini adalah contoh yang akan dipakai untuk menjelaskan inheritance




```
P :polygon
r : rectangle

p.perimeter
P.vertices; p.translate
r.diagonal; r.side1;r.side2
r.perimeter; // krn redefine, yg dipakai adalah yang
didefinisikan pd r
```

Pada feature call x.f, di mana x diturunkan dari C, feature f harus terdefinisi pada salah satu ancestor C (ingat bahwa C adalah ancestor dirinya sendiri pula)

Static type: pada saat deklarasi
Dynamic type : pada saat run time (objek)

Catatan :
Entity : identifier pada teks yang mendefinisikan Class pada saat runtime, nilainya adalah reference. Pada saat run time, nilainya adalah reference (kecuali jika expanded). Reference mungkin di-attach ke objek
Jadi hanya entity yang mempunyai static type, objek selalu dynamic type

Aturan ini menyebabkan
```
R:=p; // tidak valid
X:= p.diagonal;
```

CREATION pada inheritance

Creation : Jika x bertype T, maka efeknya create new objek bertype T, inisialisasi, dan attach ke x

```
!!x
!!x.make
```

Jika U merupakan turunan dari T, maka U dapat diinstansiasi langsung tanpa membuat T, dan terjadi polymorphim

```
!U!x
!U!x.make
```

jika dideklarasi :
```
f:Figure

if chosen_icon is rect_icon then
  !RECTANGLE!f
else if chosen_icon is circle_icon then
  !CIRCLE!f
```

Polimorphism

Akibat dari inheritance, maka dapat terjadi polymorphic attachment (*Poly*=banyak; *morph*=bentuk). Artinya pada saat yang sama, sebuah reference dapat mengacu ke Objek yang berbeda kelasnya (namun harus “se-keturunan”).

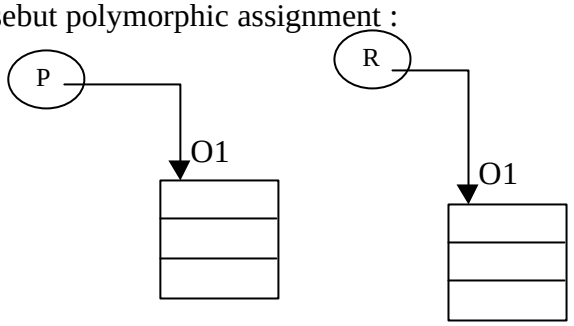
Polymorphic attachment

Jika dideklarasi
```
P :polygon
R :rectangle // rectangle is a polygon
T : triangle //triangle is a polygon
```

a. Polymorphic assignment, effeknya adalah reattach objek

Karena kecocokan type antara Bapak dan turunannya, maka kemungkinan terjadi asssignment sebagai berikut yang disebut polymorphic assignment :

```
P := R; // valid
P := T; // valid
```



Misalnya dipunyai kelas untuk Figure (lihat contoh). Jika P adalah poligon, r adalah rectangle, s adalah square.
Beberapa bahasa membatasi polimorphmisme dengan hanya memperbolehkan assignment sebagai berikut

P := r;

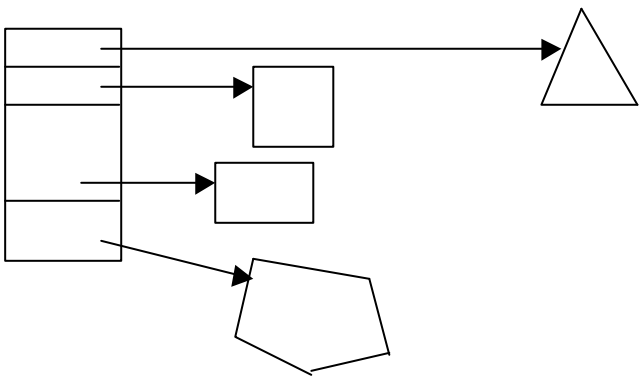
Sebaliknya, assignment sebagai berikut diperlakukan sebagai salah sintaks :

r := P

Polimorphic attachment memungkinkan misalnya, dipunyai sebuah array dengan elemen bermacam-macam, namun masih sekeluarga. Lihat contoh Array of poligon, yang pada saat run time mungkin elemennya berisi rectangle, triangle, atau salah satu turunan poligon yang lain

b.Polymorphic data structure, contoh :

```
Class array[G]
  procedure  Put (v: G; I:integer);
//mengisi elemen ke-I dengan objek G berupa Polygon


end Class array[G];
```



Dynamic binding

Polimorphism dimungkinkan jika mekanisme bahasa menyediakan fasilitas ***dynamic binding***, yaitu mengasosiasikan objek nyata pada saat eksekusi. Pada contoh di atas, jika dalam teks program mengandung feature call P.perimeter maka pada saat

eksekusi, yang mungkin diaplikasi adalah semua turunan dari P: r.perimeter (r adalah rectangle), s.perimeter (s adalah square).

Jika sebuah feature terdefinisi untuk seluruh subclass

```
!!p.make; x:= p.perimeter;
!!r.make ; x:=r.perimeter;
```

maka jelas,perimeter yang mana yang akan diaplikasi
Tetapi, pada teks sebagai berikut akan timbul masalah :

```
!!r.make
p:=r;
x:=p.perimeter; // seharusnya perimeter dari rectangle
```

Dynamic binding : adalah aturan yang menjamin bahwa operasi yang akan dilakukan terhadap objek pada saat run time selalu berdasarkan type objek pada saat run time tersebut . Ingat bahwa akibat dari kemungkinan polymorphism, maka pada teks program sebenarnya belum “nyata”, feature apa yang akan dioperasikan karena type objek baru ditentukan dengan pasti pada saat run time

Contoh bahwa dynamic binding tidak kelihatan pada teks :

```
if chosen_icon is rect_icon then
  !RECTANGLE!p.make
else if chosen_icon is triangle_icon then
  !TRIANGLE!p.make
end if
x:= p.perimeter
```

Definisi-definisi dan jenis Object dari berbagai sudut pandang dapat dilihat pada lampiran.

Jika sebuah feature terdefinisi untuk seluruh subclass

```
!!p.make; x:= p.perimeter;
!!r.make ; x:=r.perimeter;
```

maka jelas,perimeter yang mana yang akan diaplikasi
Akan menimbulkan masalah :

```
!!r.make
p:=r;
x:=p.perimeter;//seharusnya perimeter dari rectangle
```

Pada dynamic binding, harus ada aturan yang menyatakan bahwa feature yang diaplikai adalah milik objek pada saat run time

Contoh bahwa dynamic binding tidak kelihatan pada teks :

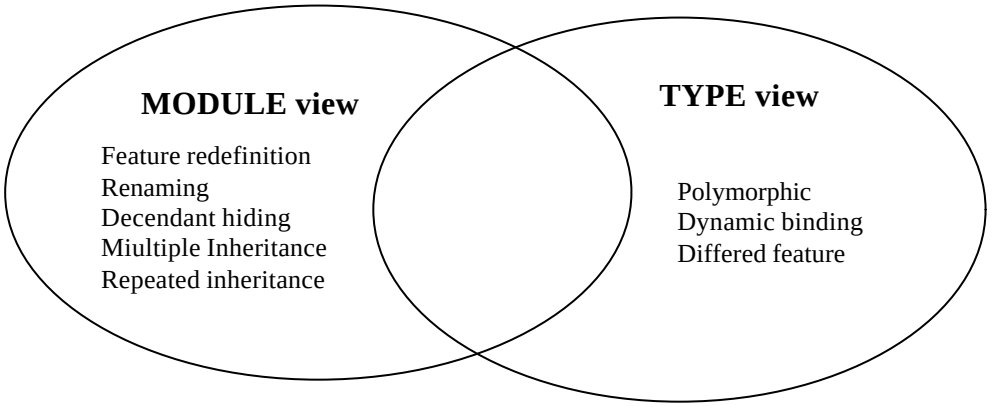
```
if chosen_icon is rect_icon then
  !RECTANGLE!p.make
else if chosen_icon is triangle_icon then
  !TRIANGLE!p.make
end if
x:= p.perimeter
```

REDEFINITION

- semantiknya harus sama
- precondition dan post condition harus sama

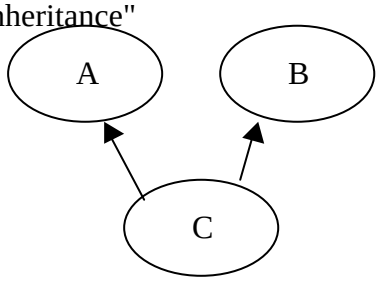
Redeclaring sebuah fungi atau atribut dimungkinkan bahwa sebuah kelas turunan mempunyai fungsi bapaknya sebagai atribut
Bagaimana jika dalam sebuah kelas diinginkan menggunakan versi asli Bapaknya : Redefine dan klausa PRECURSOR

INHERITANCE dari sudut pandang modul dan type

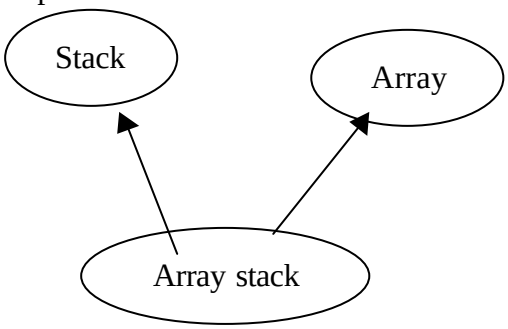


MULTIPLE INHERITANCE

Jika sebuah kelas mewarisi lebih dari satu kelas, maka terjadilah "multiple inheritance"



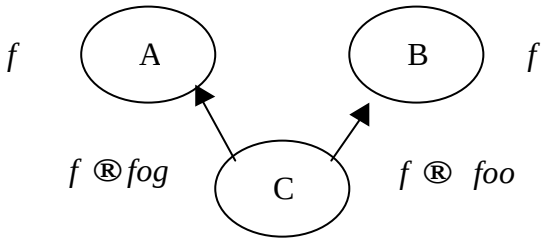
Contoh Implementasi :



```
Clas Arrayed stack
// Atribut
  Stack[G]
  Arrayed [G]
// Akses
function GetTop return G
function GetInfoTop return G
// Method
Procedure Push;
Procedure Pop
End Class arrayed Stack;
```

RENAME

Jika sebuah kelas merupakan turunan dari dua buah kelas *(multiple inheritance)*, maka kemungkinan akan terjadi konflik nama jika kedua kelas ancestor mengandung feature dengan nama yang sama. Solusi dari masalah ini adalah : RENAME

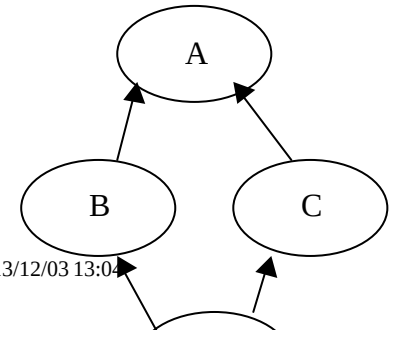



```
CLASS C
  Inherit B
      Rename f as foo
   A
// …
end Class C;
```

```
CLASS C Inherit
    B
      Rename f as foo
    A
      Rename f as fog
// …
end Class C;
```

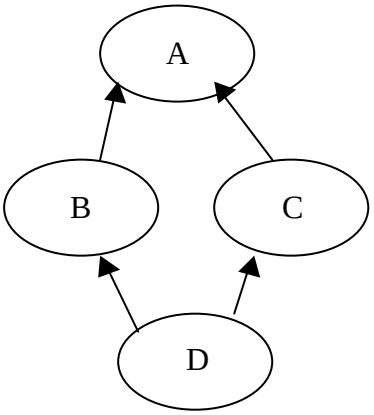
REPEATED INHERITANCE

Jika sebuah kelas yang merupakan multiple inheritance dari dua buah kelas yang mempunyai ancestor sama.

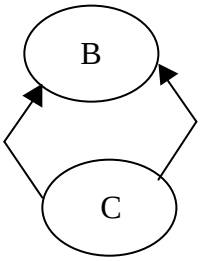


Aturan repeated inheritance:
Dalam sebuah repeated descendant, versi dari repeatedly inheritanced feature dengan nama sama merepresentasi satu feature. Versi dengan nama yang lain merepresentasi feature yang terpisah satu sama lain, setiap eksemplarnya direplikasi dari aslinya pada common ancestor
Jadi, pada Repeated inheritance :

- jika inherit nama yang sama,hanya satu single feature
- jika nama lain, redefine dari ancestor



Direct

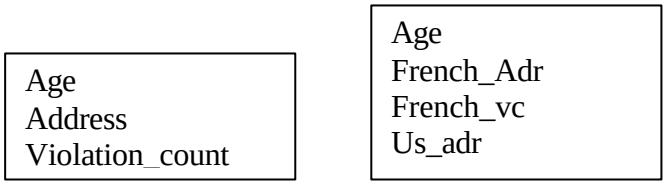


InDirect

```
Class Driver
// feature
    age
    address
    violation_count
    pass_birthday
    pay_fee

End class Driver;
```

```
Class French_US driver inherit
   French_driver
     Rename
        Address as french_adr,
        Violation_count as french_vc
        Pay_fee as pay_french_fee
   US_driver
     Rename
        Address as us_adr,
        Violation_count as us_vc
        Pay_fee as pay_us_fee

End Class French_US_driver
```

Atribut replication



RENAMING RULES

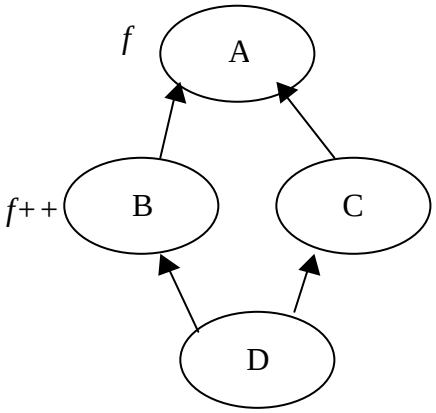
Final name :

- immediate, declared: the name itself
- for an inherited feature yang tidak di-rename: dari parentnya
- renamed feature : hasil renaming

UNDEFINED

Jika terjadi multiple inheritance dan ternyata terjadi konflik feature, maka dimungkinkan untuk melakukan undefined, yaitu "menolak/menghapus" feature yang diturunkan.

Conflicting redefinition



Situasi :

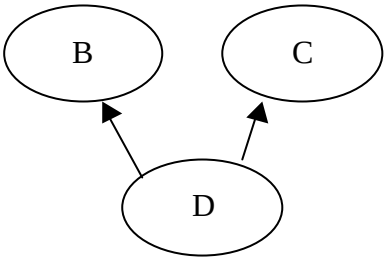
- D mendapatkan satu copy *f* dari C dan *f++* (redefine) dari B.
- Maka terjadi konflik, harus dilakukan UNDEFINED

```
Class D
  Inherit
```

```
      B
      C
        Undefine f
End Class D;
```

Konflik lain :

f

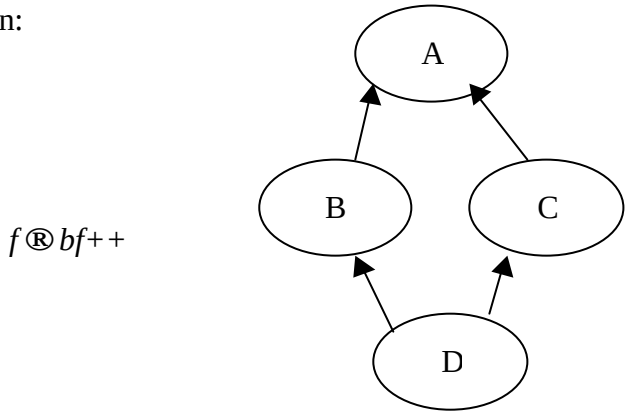


g

```
Class D
  Inherit
     B
     C
       Rename g as f
       Undefined f
End Class D;
```

Pada kasus ini, D akan punya sebuah nama f, dan hanya ada satu exemplar dari f

Contoh lain:



f ® *bf*++

Untuk A1 bertype A, A1.*f* menyebabkan call fungsi yang mana??
Misalnya pada saat runtime di-attach ke D, maka apa yang terjadi dengan A1.*f* ?
Aturan dynamic binding mengatakan bahwa akan di-aplikasi adalah f dari D, padahal D mempunyai dua versi , dikenal secara lokal dengan *bf* dan *f*

Select akan menyelesaikan persoalan ini

SELECT
Jika kelas D mendapatkan dua buah copy dari feature B dan C, maka terjadi konflik karena repeated inheritance. Solusinya adalah dengan memilih feature yang akan dipakai dengan melakukan SELECT

Aturan SELECT :
Sebuah kelas yang mewarisi dua atau lebih feature efektif dari repeated ancestor, dan tidak di-redefined harus memilih yang mana yang akan diimplementasi di kelas tsb (dengan select)

```
Class D
// akan memakai f
dari
  Inherit
     B
     C
       Select  f
End Class D;
```

```
Class D
// untuk select bf
  Inherit
     B
       Select bf
     C
End Class D;
```

Contoh :window dengan border (Stroustrup)

Resume dari redefine dan rename

Jika :

- a1 adalah objek dari kelas A, dan b1 adalah objek (entitas dinamik) bertype B.
- B adalah turunan dari A
- *f* adalah feature (nama), dengan menyebut *f'* maka d di-rename
- ϕ adalah body dari feature, jika di-redefine maka menjadi ϕ'

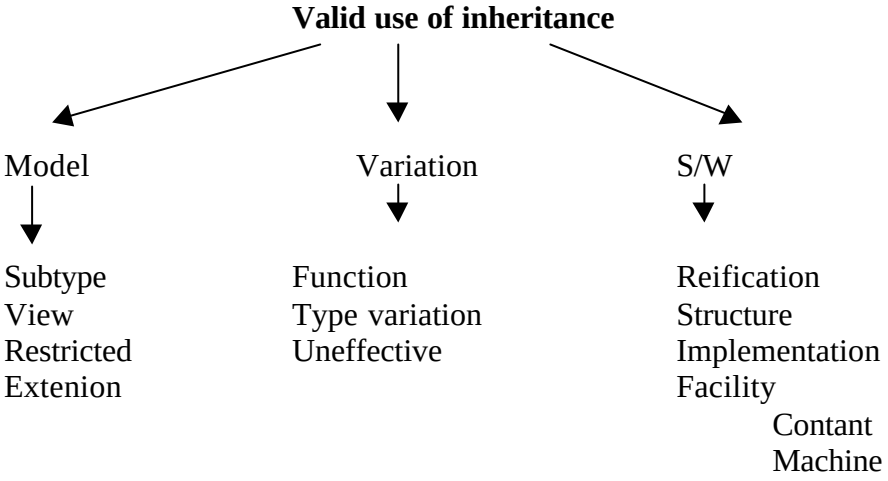
Beberapa kasus akibat kombinasi dari rename dan redefine diberikan pada tabel sebagai berikut A1.*f'* selalu tidak legal, sebab a1.*f* selalu mengacu kepada *f*. Maka kasus yang mungkin adalah ketika dynamic type dari a1 adalah b1, sehingga mungkin terjadi interpretasi b1.*f* dan b1.*f'*

No.	Case	a1.*f*	b1.*f*	b1.*f'*
1.	*f* not redefined *f* not renamed	ϕ	ϕ	illegal
2.	*f* not redefined *f* renamed *f'*	ϕ	illegal	ϕ
3.	*f* redefined ϕ' *f* not renamed	ϕ'	ϕ'	illegal
4.	*f* redefined ϕ' *f* renamed *f'*	ϕ'	ϕ'	ϕ
5.	*f* not redefined *f* renamed *f'* *f'*redefined ϕ''	ϕ''	illegal	ϕ''
6.	*f* redefined ϕ' *f* renamed *f'* *f'*redefined ϕ''	ϕ''	ϕ'	ϕ""

- Kombinasi dari invariant dan dynamic aliasing menimbulkan indirect invariant effect, yang mungkin menyebabkan objek menyalahi invariant walaupun pada objek itu sendiri tidak ada fault

Penggunaan Inheritance

Pada [Meyer-97], perancangan inheritance dijelaskan dengan sangat lengkap. Pada hakekatnya, tidak disarankan membuat hubungan inheritance yang "sembarangan". Klasifikasi penggunaan yang tepat dari inheritance diberikan sebagai berikut



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 sebagian atau lengkap Contoh : Tabel, bisa sebagai hash table atau sequential table. Sequential table bisa diimplementasi sebagai array atau linked list
	Structure	A : deferred class, represent general structural B : certain type of object, turunan dari A Contoh : A comparable, B adalah string atau integer
	Implementation	Implementasi fisik dari sebuah struktur lojik (misalnya stack dengan representasi array)
	Facility	Constraint : untuk mendefinisikan semua kontanta sistem, misalnya konstanta ASCII
		Machine : misalnya **iterator** (skema sekuensial)

Studi kasus Implementasi Program Berorientasi Objek

Berikut ini diberikan gambaran beberapa persoalan berdasarkan kelas persoalan yang dibuat sebagai latihan dalam perkuliahan IF322:

- ADT dan kelas pemakaianya: hubungan Client Supplier murni
- Mesin dan kelas pemakaianya: Client Supplier
- Persoalan dengan struktur data kompleks namun algoritma (dinamika kehidupan) sederhana. Struktur data diformulasikan menjadi sekumpulan kelas dengan hubungan yang mungkin kompleks (inheritance, client supplier)
- Persoalan dengan struktur data sederhana (diagram statik kelas sederhana), namun algoritma kompleks (diformulasikan menjadi sebuah kelas yang biasanya dinyatakan sebagai state machine)
- Persoalan objek konkuren:
    - Client-Server : Sebuah atau lebih server menyediakan sejumlah services. Client meminta request (berupa layanan) ke Server yang menyediakan. Dalam konteks ini, biasanya dibutuhkan sistem antrian dengan prioritas yang juga dimodelkan sebagai kelas. Dari model paling sederhana (single server), kasus ini dapat dikembangkan menjadi multi server, multi server dengan urutan proses tertentu (job shop), atau sebuah server menjadi client dari server lain dan melemparkan request
    - Watch dog : Sebuah “dog” mengobservasi terus menerus berjalannya sebuah proses. Jika ada sesuatu yang tidak normal, maka akan men-signalkan sesuatu, misalnya membangunkan sebuah proses lain yang dalam hal biasa “tidur”
    - Producer-Consumer : Sebuah kelas memproduksi sesuatu yang akan dimanfaatkan oleh konsumer. Konsumer hanya dapat mengkonsumsi jika produser sudah menghasilkan sesuatu. Biasanya, jika sebelum suatu produk selesai dikonsumsi, sudah dihasilkan produk lain, maka perlu disediakan kelas “Buffer”
    - Model-model konkurensi lain yang diajarkan dibagian Konkuren

Persoalan yang sama dapat dipandang dari sudut pandang yang berbeda!. Lihat contoh studi kasus yang diberikan (Diktat Pemrograman Berorientasi Objek)

Dari segi siklus hidup pengembangan PL berorientasi Objek :

- OOA : analisis persoalan dengan salah satu methodologi tertentu. Hasil utamanya adalah sekumpulan diagram sesuai dengan metodologi yang dipakai. Sentra dari produk adalah Diagram Class (dengan relasi, asosisasi antar kelas), dan diagram yang menyatakan dinamika (behaviour, kelakukan, ..) dan interaksi dengan user
- OOD : buat detail design, skema solusi secara keseluruhan
- OOP : implementasi setiap kelas (jika belum) dalam salah satu bahasa pemrograman yang dipilih dengan/tanpa memanfaatkan library yang tersedia.
- OOT :
    - unitary test: test setiap kelas yang independent,
    - kemudian test semua kelas yang mempunyai hubungan.
    - test seluruh sistem (lebih sulit jika ada banyak objek aktif)

Sarana reusability: class library, design pattern
Untuk sebuah persoalan tertentu : mulailah dari diagram kelas yang statik, kemudian tuliskan dinamika dari objek (mungkin menjadi sebuah kelas)!

Penggunaan ulang dengan orientasi objek, memungkinkan penggunaan tidak hanya tingkat kode, melainkan sampai dengan design. Maka dikenal tersedianya pattern. Semua peristilahan mengenai pattern disertakan dalam lampiran

Lampiran A. REKAPITULASI OOP

Dibuat oleh Inggriani, Achmad Imam Kistijantoro dan Avan Suenesia

Konsep	**Eiffel**	**C++**	**Java**
Ciri bahasa	– Bertrand Meyer, OO Software Cinstruction – murni – tidak case sensitive	– Stroustrup, judul buku utama – Hybrid – Case sensitive	Reference book: Java Language Specification (Sun). type ada 2 jenis: reference & primitive
Type	– Ada type dasar dengan default initialization rules: Numeric (integer; float; double: ZERO sesuai type), character(NULL), boolean (false) – type conformance rules definition yang jelas – type reference (default) dan expanded	Type dasar sama dengan C	type ada 2 jenis: reference & primitive
	static type (pada saat definisi), dynamic type (pada saat object di-attach)		
Kelas	Keyword CLASS diakhiri end, berisi definisi kelas, feature. Semua spesifikasi dan body dikode menjadi satu		class { .. }
Feature	Atribut dan method tidak dibedakan disebut sebagai FEATURE	data member function member	data member function member
Akses terhadap feature	NONE, kelas tertentu, tanpa batas (public)	PRIVATE, PUBLIC, PROTECTED	private, public, protected
Modifier (life time, dll)			
- static		static	

Konsep	**Eiffel**	**C++**	**Java**
- volatile			
- const			
- local/global			
Objek	Dihidupkan oleh Creation procedure	Dihidupkan oleh default constructor sesuai nama kelas, jika tidak ada kostruktor	dihidupkan oleh default constructor sesuai nama kelas
Reference	Semua objek punya reference (by default), tetapi tersedia type `expanded`	Reference harus dinyatakan secara eksplisit dengan deklarasi objek sebagai pointer ke kelas. Tapi ada type REFERENCE **(&)** yang artinya lain, yaitu ***address of.***	Semua objek yang diciptakan selalu mempunyai reference, tidak ada lagi deklarasi pointer
Constructor	Klausa : `creation` dinyatakan dalam creation procedure jika tidak ada !!<namakelas>,p(..) yang terjadi adalah: - instansiasi objek sesuai definisi kelas - inisialisasi semua field dengan nilai default - attach reference ke objek yang diciptakan - call prosedur p	jika tidak ada, maka ada construktor default dengan nama yang sama dengan nama kelas	jika tidak ada, maka ada constructor default dengan nama yang sama dengan nama kelas
Destructor	Ada garbage collector. Objek yang tidak dipakai lagi akan dihancurkan oleh sistem	Harus dipanggil secara eksplisit	Secara otomatis oleh garbage collector

Konsep	Eiffel	C++	Java
Manipulasi objek:			
- Assignment	boleh asignment antara reference dan expanded dengan definisi yang pasti	Assignment pointer OK Assignment objek bukan bertype pointer: hati-hati, lebih baik memakai Copy procedure	untuk tipe reference dilakukan assignment reference, untuk tipe primitif dilakukan assignment value
- Copy object	ada definisi copy objek antara reference dan expanded (dan sebaliknya)	Ditulis/dikode	ada prosedur terdefinisi
- Deep copy	perintah `deep clone`, dengan semantik terdefinisi	tidak ada, harus dikode	tidak ada, harus dikode
- object comparaison	ada definisi semantik jika kedua objek trnyata berbeda (expanded, reference)	Tergantung deklarasi	ada prosedur terdefinisi, namun prosedur ini dapat didefinisikan ulang jika diperlukan
Abstract Class: kelas yang tidak dapat diinstansiasi menjadi objek sebab ada yang belum terdefinisi	`deferred class`	`Abstract base class`	`abstract class { .. }`
Generic Class	tersedia fasilitas untuk mendefinisikan kelas generik dengan notasi khusus, yaitu parameter berkurung siku [ ]	Template	tidak ada
Kelas asal-muasal (root class): semua kelas adalah turunan dari kelas ini	Kelas `general` dengan subclass `ANY`	Tidak ada, harus didefinisikan dengan eksplisit	tidak ada
Kelas pengunci: kelas ini adalah child dari	`NONE`	Tidak ada	Object

Konsep	Eiffel	C++	Java
semua kelas yang terdefinisi			
Polimorphism & dynamic binding	Assignment parent:=Child sah	Harus bertype pointer	tidak ada
Inheritance	ada, dengan keyword inherit	Ada, dengan keyword ::	ada, dengan keyword `extends`
- penciptaan objek anak	jika kelas P adalah Bapak dari C:	Ada	tidak ada
Multiple inheritance	ada	Boleh	tidak ada karena tidak ada multiple inheritance
Repeated inheritance		Boleh	tidak ada karena tidak ada multiple inheritance
Konflik inheritance			
- Redefine: body diganti, nama tetap	ada	Ada berkat fasilitas OVERLOADING, dengan scope menjadi jelas.	bisa dilakukan dengan overiding
- Rename: nama diganti, body tetap	ada	Buat nama baru di Child, bodynya hanya call milik Parent	buat nama baru di child, bodynya hanya call milik parent
- undefine: didrop sebagai feature	ada	Tidak bisa	tidak bisa
- Select: pilih pada saat runtime jika ada konflik	ada	Tidak ada, penanganan konflik dengan operator scope ::	tidak ada, sebab tidak ada multiple inheritance
Assertion	– Class invariant	Assert	tidak ada

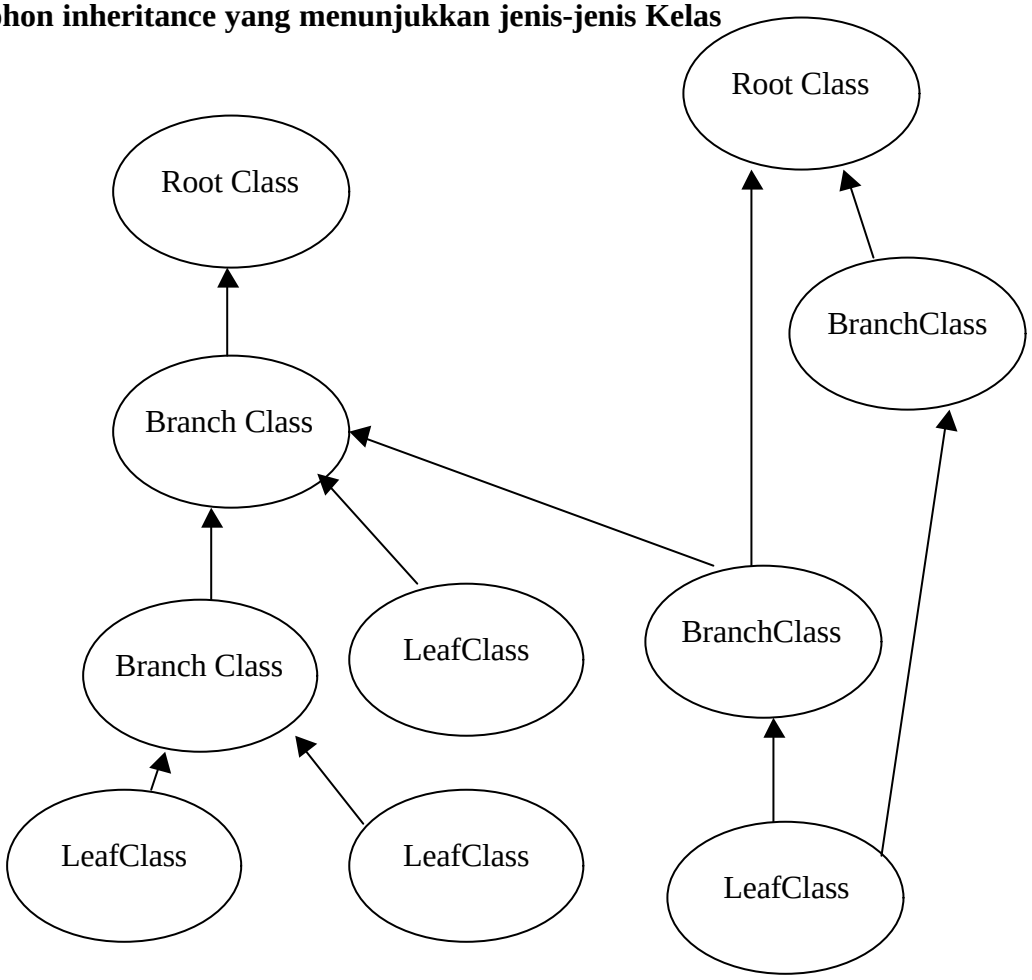
Konsep	Eiffel	C++	Java
	– Loop invariant – `require` : Precondition – `ensure`: Post condition – `check` : untuk mencek asersi		
Exception	– keyword: `rescue, retry` – ada 12 jenis exception yang didefinisikan dengan jelas – tersedia sebuah kelas bernama `exceptions`, menyediakan fasilitas utk melakukan query terhadap exception yang terjadi	Catch, retry	EXCEPTION
Component Library	tersedia: – Library BASE, terdiri dari KERNEL, STRUCTURES, ITERATOR – library lain : Math, vision, parse, net, Wel		tersedia: JavaBeans
Implementasi program			
- pembagian file, unit kompilasi	setiap kelas adalah sebuah file	Header file (.h) dan body file (.cpp)	sebuah file hanya dapat berisi 1 public class, namun dapat diisi dengan lebih dari satu non public class
- kompilasi/run program	Definisikan sistem	Definisikan makefile	javac *.java java <kelas-utama>
- root class (kelas awal) versus main program	dibedakan		

Konsep	**Eiffel**	**C++**	**Java**
(awal eksekusi)			
- dokumentasi	ada fasilitas untuk ekstraksi dokumentasi		Ada fasilitas untuk ekstraksi dokumentasi
Konkurensi	library `Thread,` dengan kelas `THREAD, PROXY. MUTEX`		Thread, synchronized
Konsep khusus:			
Interface (Java)			interface
package (Java)			
inline function (C++)			
anchored delaration (Eiffel)	- keyword : LIKE anchor -		

LAMPIRAN B. PERISTILAHAN YANG PERLU DIPAHAMI

Berikut ini diberikan daftar istilah (terminologi) yang perlu diketahui oleh mahasiswa IF322, disalin dari sebuah "kamus" tentang Orientasi Objek, dan sengaja tidak ditejemahkan (dalam bahasa Inggris) yaitu :
[Firesmith-95] : Firesmith D.G and Eykholt, E.M : "Dictionary of Object Technology", SIGS Reference Library1995
Pada kamus itu, semua peristilahan yang mirip, tumpang tindih, dengan baik diklasifikasi. Keragaman istilah itulah yang ingin diperkenalkan ke mahasiswa IF322, dan sekaligus menunjukkan bahwa OO adalah paradigma yang relatif masih berkembang.

Pohon inheritance yang menunjukkan jenis-jenis Kelas



CLASS:

1.a. any uniquely-identified abstraction (i.e. model) of a set of logically-related instances that share the same or similar characteristics [Firesmith, Lorenz, Rumbaugh]
1.b. any concept that has members [Henderson-Sellers]
2. any possibly generic factory for the instantiation of instances [Firesmith, Lorenz Jacobson, OMG, WOrfs-Brock]
3. the unit of class inheritance; an encapsulation of features (possibly class level) that maybe reused by other class via inheritance [CLOS,Firesmith]

4.a. the combination of a type interfaceand associated type implementation [Firesmith, Henderson-Seller, ODMG, OMG, Smalltalk]
4.b. any implementation of a type of objects, all the same kind [Martin/Odell, Smalltalk]
4.c. any description of a set of objects that have identical responsibilities [Coad]
4.d. any object that determine the structure and behavior of other objects: its instances [CLOS]
5. any set object that share the same o similar features [Booch, Coleman, Embley]
6. any user defined type (C++)
7. the unit of modularity, data binding and encapsulation (C++)
8. any modular unit that describes the properties of a set of possible objects, when viewd as a type.[Eiffel, Meyer]
9. any set of type that closed under derivation. If a given type is in the class, then all types derived from that type are also in the class [Ada95]

The instances of clusters are collection of objects, the instances of framework classes are frameworks, the instances of object class are objects, and the instances of scenario classes are scenario.

Definition 5 confuses the term class and extent
Definition 6 confuses the term class and type
Definition 7 ignores the fact that clusters are also units of modularity, data hiding and encapsulation

Jenis	**Keterangan**
Abstract Class	Any incomplete class that cannot therefore be used to instantiate semantically meaningful objeks Any calss that cannot have direct instances but whose descendants can have instances any class that should not have instances any class with no instance any class that can only be used as parent class from which to derive other class A base class that declares the existence of one or more features that must be implemented by its derived class prior to instantiation synonim: deferred class antonym : concrete class
Deferred Class	any abstract/base class that declares the existence of one or more features that musat be implemented by its descendant prior to instantiation
Fully Deferred Class	
Partial Deferred Class	
Mixin Class	
Aggregate Class	Synonim: composite class

Jenis	Keterangan
	Antonym : atomic class
Collection Class	Any homogeneous aggregate class of collection of objects Collection enforce strong typing Contrast with : container class, sructure class
Container Class	Any heterogeneous aggregate class of collection of objects Container sllow more freedom by violating strong typing
Structure Class	
Ancestor Class	any class from which the given class is directly or indirectly derived via inheritance either the given class or any class from which the given class is directly or indirectly derived via inheritance Synonim: base class, superclass Antonym : descendant class, derived class, subclass Contrast with: child class, heir, parent class
Direct ancestor	
Immediate ancestor	
Indirect ancestor	
Parent Class	Synonim: direct base class, direct superclass, immediate ancestor, immediate base class, immediate superclass Antonym : child class, direct derived class, direct descendant, direct subclass, heir, immediate derived class, immediate descendant, immediate subclass
Repeated parent	
Proper ancestor Class	Any ancestor of the given class other than itself (Eiffel, Meyer)
Repeated ancestor	any ancestor class from which the given class is multiply derived via multiple inheritance
Ultimate ancestor	any ancestor of the given class that is not descendant of any other ancestor
Anonymous Class	the unspecified class of the given object, the class of which is not relevant to the current discussion
Association	Any semantic relationship between two or more classses or types Any object class whose instances are set Synonim: set class

Jenis	Keterangan
	Antonym : member class, universe Cntrast with : associative class. link
Base Association	
Derived association	
Associative Class	
Atomic Class	Any class, the instance of which are atomic object ***) Menurut IL, ini adalah kelas statik dalam bhs C++ Any class that does not contain any class as component parts
Attribute Class	
Base Class	any class that is not appication specific, but instead is used to support those classes that are Synonym: Support class Antonym : domain class
Base Class (of a given Class)	C++ " ancestor class of the given class any class that is not derived via inheritance from any other class (Booch) The class from which the type of category is derived (Eiffel, Meyer_
Direct Base Class	
Indirrect Base Class	
Virtual Class	
Base Class (of a given category)	
Base Class (of a given inheritance structure)	Synonim: root class Antonym : leaf class Contrast with: branch class
Behavioural Class	
Behavioured Class	
Branch Class	
Class Object	
Class rooted at a given type	
Cluster	Any encapsulation off class or clusters that are typically analysed, designed, coded, tested and integrated as a small increment of development. A cluster has an interface exporting its protocol of visble features and an implementation containing its hidden features.
Aggregate Cluster	
Atomic Cluster	
Component Cluster	
Complete Class	Any class capturing all of the responsibilities and features needed to both accurately implement the associated abstraction and meet its current requirements. Where practical, a

Jenis	Keterangan
	complete class should also implemen all features required to support reusability and user-friendliness
Complex Class	
Component Class	
Composite Class	any class that is composed of oter classes Synonym: aggregate class, nested class Antonym : Atomic class Contrast with : aggregate class, container class
Concrete Class	any class that can be instantiated to produce semantically meaningful instances
Concurrent Class	any class whose instance are concurrent object
Active Class	
Guarded Class	any class, the instance of which provide a mechanism for ensuring (but not enforce) mutually exclusive access in concurrent environment
Synchronous Class	
Consistent Class	any class, the instances of which are instantiated in a state satisfying all invariants and having only operations which if started in a state satisfying the precondition and the invariants also terminate in a state satisfying the postcondition and invariants
Corruptible Class	any class, the instances of which do not ptovide any mechanism for ensuring mutually exclusive access in a concurrent environment
Derivation Class	
Derived Class	the C++ terms for any descendant class of a given class Synonym : child class, descendant, subclass Antonym : ancestor class, base class, parent class
Direct Derived Class	
Indirect Derived Class	
Most Derived Class	
Descendant Class	
Behaviorally compatible Descendant	
Child Class	
repeated child	
Direct descendant	
Heir Class	

Jenis	Keterangan
Repeated Heir	
Indirect Descendant	
Proper Descendant	
Repeated descendant	
Descriptor Class	
Domain Class	any class that is application domain specific
Application Class	any class of objects that model and ionterface with end user application any class of objects encapsulating entire application Synonym: wrapper, bususiness class, core class, key class, model class Contrast with: controller class, key abstraction, presentation class, support class, view class
Business Class	
Controller Class	any domainclass of controller objects that exist to either control one or more objects or to capture user input as in MVC framework Contrast with : Controller object, model class, view class,
Core Class	
Key Class	any essential class of model objects that captures a key abstraction of the application domain. A key class is central to the business domain being automated
Model Class	
View Class	any domain class of view objects that exist to provide a usr view (i.e. information about and control over) one or more model object
Presentation Class	any specalized view class of presentation objects that exist to provide a formatted view one or more model or view class
Proxy Class	any local view class that represents (i.e. acts as processor-specific variant o of an communicates with) its remote model class. Any proxy provides the local protocol of (and communicate with) its model on a remote processor. proxies are used to create a single virtual address space across multiple processors
Equivalence Class	
Executable Class	An executable class corresponds to a

Jenis	Keterangan
	fully linked, executable object module.
Execution Support Class	
Expanded Class	any class, the entities of whose corresponding type will have objects as their run-time value.
Expanded Client	
Framework Class	
Friend Class	any class whosw implementation has been granted permission by another class to reference its hidden parts in violation of information hiding
Generating Class	
Guardable Class	
High level Object Class	
Dominant High level Object Class	
Independent High level Object Class	
Join Class	any class that multiply inherits from more than one parent class. A join class is analogous to the joins that occur between tables in a relational database
leaf Class	
Local Class	
Member Class	
Metaclass	any class, the instance of which are themselves classes
Generic Class	any abstract metaclass that is parametrized with formal generic parameters
Parametrized Class	
Power Class	
Discriminator Class	
Standard Class	
Template Class	C++
Nested Class	C++ any class declared with in another class
Object Class	any class of objects; the instance of some classes need not be object
Origin Class	the class in which the given feature was originally declared or last implemented
Declaring Class	
Implementing Class	
Passive Class	
Peripheral Class	any class that provides part of the supporting framework for the business' key class
Persistent Class	Any class, the instance of which are

Jenis	Keterangan
	persistent object (i.e. object that exist after the execution of the program, process, or thread that created them) A persistent object typically survives the process or thread that created it, existing until explicitly deleted. A persistent object also typically exists outside the address space in which it is created. Persistents objects can be stored in Object Oriented databases, extended relational databases, relational databases, Smalltalk images, files, etc Antonym : transient class
Reacheable Class	
Relational Object Class	
Role Class	
Root Clas	1. Any class that does not inherit from any other calss (Smalltalk) 2. The master class in a system or class that depends on (i.e. is a direct or indirect client or heir of) all the other classes in the system (Eiffel) Def.1) is based on inheritance, def 2. is based on inheritance and Client-Server dependencies
Scenario Class	
Use Case	
Abstract Use Case	
Concrete Use Case	
Sequential Class	
Set Class	
Singleton Object Class	
Standard-Object Class	
Storage Class	C++
Automatic Calss	
Static Class	
SubClass	
Direct Subclass	
Indirect Subclass	
Subpart class	
SuperClass	
Direct Superclass	
Indirect Superclass	
Superpart Class	
Built in class	
Class	
T Cclass	
Transient Class	

Jenis	Keterangan
Dynamic Class	
Static Class	
Universe	
User Defined Class	
Virtual Node Class	

OBJECT:

1.a. any abstraction that models a single thing [Coad, Coleman, Eiffel, Firesmith, Lorenz, Meyer, OMG, Rumbaugh]
1.b. during design, any abstraction of real world thing [Shlaer-Mellor]
1.c. any real or abstract thing about which we store data and the operations to manipulate those data [Martin/Odell]
2.a. any identifiable, encapsulated entity that provides one or more services that can be requested by a Client [Jacobson, OMG]
2.b. any encapsulation of properties (e.g. data, state) and behavior (e.g. operations) [Booch, Coad, Firesmith, Lorenz, Jacobson, OMG, Smalltalk, Wirfs-Broc]
3. anything with identity [Booch, EMbley, Firesmith, Jacobson, OMG, OADSIG]
4. anything that can send/receive message
5.a. any instance of one or more (possibly anonimous) classs or types [Booch, CLOSS, Eiffel, Firesmith, Henderson-Sellers, Martin/Odell, Meyer, Rumbaugh, Wirfs-Broc]
5.b. During analysis, any typical but unspecified instance [Shlaer-Mellor]
5.c. any member of an extent (a.k.a. extension) [Firesmith, Henderson-Sellers]
5.d. anything to which a type applied [Martin/Odell]
6. the primary type of inheritance structure [OMG]
7. any region of storage [C++]
8. during aalysis, any abstraction of a set of real-world things such that all teh real world things in the set (the instances) have the same characteristics and all instances are subject to and conform to the same rules [Shlaer/Mellor]
9. any person, place, or thing [Embey]
10. any run time entity of a given type that has a value of that type [Ada95]

As a model of a single thing, object must have an identifier that unique within their scope. As a complete model, an object encapsulates responsibilities and the features necessary to implement them. These features typically include both properties (e.g. attrubutes, links to other objects, component objects, and invariants) and behaviour (e.g. message that may be received, exception,s that maybe raised, opertaion that execute). An object collaborates with others by sending and/or receiving message. Object may occur as part of either the system or software requirements, design, and/or implementation. Objects maybe discovered in the real world during requirement analysis or invented as part of the solution space during design
Shlaer Mellor appear to use the term object means class, and the term instance to mean object. However, Mellor has stated that this view is incorrect and that an object during analysis should rather be considered to be a representative, but unspecified instance.

Jenis Object	**Keterangan**
Actor	
Aggregate Object	
Collection Object	
Container Object	
Structure Object	
Associative Object	
Atomic Object	
Automatic Object	

Jenis Object	**Keterangan**
Binary Large Object	
Class Descriptor	
Class Object	
Client Object	
Collaborator Object	
Complete Object	
Complex Object	
Component Object	
Composite Object	
Homogeneous Composite Object	
Compound Object	
Concurrent Object	
Active Object	
Guarded Object	
Constrained Object	
Context Object	
Control Object	
Coordinator Object	
Current Object	
Dependent Object	
Direct Dependent Object	
Indirect Dependent Object	
Descriptor Object	
Dispatcher Object	
Server Object	
"DM Object Server" Object	
Domain Object	
Application Object	
Business Object	
Entity Object	
Model Object	
Problem Domain Object	
View Object	
Pane Object	
Presentation Object	
Proxy Object	
External Object	
Frame	
Functor Object	
Future Object	
Generic Object	
Ideal Object	
Immutable Object	
Implementation Object	
Incident Object	
Instance of a given Class	
Direct instance of a given Class	

Jenis Object	**Keterangan**
Indirect instance of a given Class	
Intangible Object	
Interaction Object	
Interface Object	
Central interface Object	
Literal Object	
Maillon Object	
Managed Object	
Manager Object	
Master Object	
Meta Object	
Method-Combination Object	
Method Object	
Monolithic Object	
Mutable Object	
Passive Object	
Blocking Object	
Sequential Object	
Persistent Object	
Primitive Object	
Real Object	
Receiver	
Relationship Object	
Replicant Object	
Resilient Object	
Role Object	
Root Object	
Server Object	
Special Object	
Specification Object	
Standard Specification Object	
State-Controlled Object	
State-dependent Object	
State-independent Object	
Stimulus-controlled Object Object	
Strategy	
Suboject	
Suppart Object	
Subtype Object	
Superpart Object	
Supertype Object	
Support Object	
Utility Object	
Tangible Object	
Transaction Object	
Transient Object	
Dynamic Object	

Jenis Object	Keterangan
Orphaned Object	
Sraric Object	
Type Object	
user-interface Object	

Reference :

- attached reference : mengacu ke sebuah objek yang ada pada saat runtime
- dangling reference : refer ke object yang tidak ada
- imported reference : reference yang disimpan dalam object instance variable, bukan digenerate oleh object melalui creation
- ingenerated reference : reference yang digenerate oleh object melalui creation method
- void reference: reference yang tidak mengacu ke object apapun

Object orientation:

1.a. the paradigm that use objects with identity that encapsulate properties and oeprations, message passing, class, inheritance, polymorphism, and dynamic binding to develop solution that model problem domains [Firesmith, Lorenz]

1.b. any technique based on the concept of object, class, instances and inheritance [Jacobson]

2. the use of objects as the atom of modeling [Coleman]

OO : describing something based on the following concept:

- encapsulation
- Object: identity, properties, operations
- message passing
- classes
- inheritance
- polymorphism
- dynamic binding [Fireman]

Inheritance :

1.a. the incremental construction of a new definition in terms of existing definitions without disturbing the original definitions and their clients [Ada95, Firesmith]
1.b. the construction of a definition by incremental modification of other definition [Jacobson, OMG]
2. the definition of the derived class in terms of one or more base classes [Booch, Coleman, Embley, Jacobson, Wirfs-Brock]
3.a. a mechanism for expressing commonality between class so that new classes inherit responsibilities from existing classes. [Coad]
3.b. a mechanism that permit class share characteristics [Rumbaugh]
4.a. the organization of similar types of classes of objects into categories [Lorenz]
4.b. the taxonomy relationship between parents and children, possibly over many generations [Henderson-Seller]

Contrast with : specialization, subtyping

A new definition may :

- add feature
- modify or replace some of the inheritaed features
- define deferred inheritated features
- delete inheritaed features

Because of danger involved, few methodologist or languages allow the deletion of inheritaed features

Jenis	**Keterangan**
Class Inheritance	
Class-Instance Inheritance	
Dynamic Inheritance	dynamic binding
Event Inheritance	
Implementation Inheritance	
Interface Inheritance	
Specialization Inheritance	
Strict Inheritance	
Inverted Inheritance	any inheritance whereby the parent defines all of the feature needed by the children and thsee individual children delete the unnecessary features of the parent rather than add new feature or override inherited feature Synonim : selected inheritance
Multiple Inheritance	
Mixin Inheritance	
Repeated Inheritance	
Direct repeated Inheritance	
Indirect repeated Inheritance	
Replicated Repeated Inheritance	
Shared Repeated Inheritance	

Jenis	**Keterangan**
Virtual Inheritance	
Object-object Inheritance	
Private Inheritance	
Protected Inheritance	
Public Inheritance	
Selective Inheritance	
Single Inheritance	
Specification Inheritance	
Type Inheritance	

Macam-macam PATTERN

Pattern :

1. any reeusable architecture that experience has shown to solve a common problem in a specific context [Firesemith]
2. any reusable template of objects with stereotype responsibilities and interactions [Coad]
3. any basic design rule that can be used to guide the development of framework [Lorenz]

Pattern are useful during analysis, design, coding and testing. Patterns provide larger building blocks than individual class or objects. Patterns are usually based on experience and discovery rather than invention. A pattern provides a common solution to a problem in a specific context [Booch]

Jenis	**Keterangan**
Design Pattern	
Behavioural Pattern	
behavioural Class Pattern	
Interpreter	
Template method	
Behavioural Object Pattern	
Chain of responsibility	
Command	
Iterrator	
Mediator	
Memento	
Observer	
State	
Strategy	
Visitor	
Creational Pattern	
Creational Class Pattern	
Factory Method	
Creational Object Pattern	
Abstract Factory	
Builder	
Prototype	
Singleton	
Structural pattern	
Structural Class Pattern	
Adapter	
Structural Object Pattern	
Adapter	
Bridge	
Composite	
Decorator	
Facade	
Flyweight	
Proxy	

Jenis	**Keterangan**
Framework	
Fundamental pattern	
Idiom	
Mechanism	
Key mechanism	
Message Pattern	
Transaction Pattern	

Lampiran C. CONTOH PROGRAM DALAM BAHASA EIFFEL

Definisi Kelas, atribut, operasi sederhana

```
indexing
   description:
      " Kelas KELAS1 menunjukkan contoh pendefinisian kelas %
      % dari bab 3.1 yang meliputi pendefinisian nama kelas, %
      % atribut, operasi, dan enkapsulasi atribut dan operasi "
   date  : "4 Jan 1998"
   file  : "kelas1.e"

class KELAS1
-- contoh klausa creation di mana prosedur_creation dideklarasi
  -- sbg prosedur creation untuk kelas Kelas1, bab 3.1.1
  creation
     prosedur_creation
  -- klausa-klausa feature yg menunjukkan status ekspor yang berbeda
  -- bab 3.1.3
  -- PUBLIK

  feature
  -- feature-feature pada klausa ini bersifat publik dan dapat
  -- digunakan oleh semua kelas

    -- contoh atribut bab 3.1.2
    atribut_integer   : INTEGER
    atribut_character : CHARACTER

    -- contoh atribut konstanta , bab 3.1.2
    -- dan contoh konstanta manifest, bab 3.6.2
    konstanta_boolean   : BOOLEAN   is True
    konstanta_integer   : INTEGER   is 10
    konstanta_real      : REAL      is 2.5
    konstanta_character : CHARACTER is 'Z'
    konstanta_string    : STRING    is "ABC"
  -- Feature-feature yang hanya dapat diakses kelas tertentu

  feature {KELAS_AWAL}
  -- feature-feature pada klausa ini hanya dapat digunakan
  -- oleh KELAS_AWAL, bab 3.1.3

    -- contoh pendefinisian operasi, bab 3.1.2

    prosedur_set (i : INTEGER; c : CHARACTER) is
          -- prosedur prosedur_set adalah contoh operasi biasa yang
         -- akan mengubah nilai atribut_integer menjadi argumen i dan
          -- mengubah nilai atribut_character menjadi argumen c
       do
          atribut_integer := i
          atribut_character := c
       end

    fungsi_sekali : REAL is
     -- fungsi fungsi_once adalah contoh operasi yang hanya dipanggil
  -- sekali saja untuk semua instans kelas KELAS1, bab 3.1.2
       once
          -- entitas Result akan dikembalikan oleh fungsi pada akhir
          -- eksekusi fungsi
          Result := konstanta_integer + konstanta_real
```

```
       end  -- fungsi_sekali

  -- PRIVAT
  feature {NONE}
  -- feature-feature pada klausa ini bersifat privat dan tidak
  -- dapat digunakan oleh kelas lain

    -- prosedur_creation didefinisikan pada klausa yang bersifat
    -- privat berarti hanya dapat dipanggil pada saat instansiasi

    prosedur_creation is
          -- prosedur_creation digunakan untuk menginisialisasi
          -- atribut atribut_integer dan atribut_character
       -- klausa local berisikan deklarasi entitas lokal untuk
       -- prosedur prosedur_creation
       local
          nilai_awal : INTEGER
       do
          nilai_awal := konstanta_integer + 1
          atribut_integer := nilai_awal
          atribut_character := konstanta_character
       end  -- prosedur_creation
end -- KELAS1
```

Asersi, Exception

```
indexing
   description:
      " Kelas PENGHITUNG menyimpan suatu  nilai berkisar antara %
      % nol sampai nilai tertentu; %
      % nilai hitungan ini dpt dinaikkan/diturunkan sebesar satu %
      % atau dibagi dengan nilai tertentu. %
      % Kelas ini memperlihatkan contoh asersi& penanganan eksepsi %
      % bab 3.3 "

   date  : "4 Jan 1998"
   file  : "asersi.e"

class PENGHITUNG
  -- PUBLIK
  feature
    -- hitungan_maksimum adalah konstanta nilai maksimum hitungan
    hitungan_maksimum : INTEGER
    -- hitungan adalah integer yang menyimpan nilai hitungan
    hitungan : INTEGER
    nol : BOOLEAN is
          -- fungsi nol bernilai True jika hitungan = 0
       do
          Result := (hitungan = 0)
       end  -- nol

    maksimum : BOOLEAN is
          -- fungsi maksimum bernilai True jika hitungan maksimum
       do
          Result := (hitungan = hitungan_maksimum)
       end  -- maksimum
naik is
          -- prosedur naik menaikkan hitungan sebanyak 1 nilai

       -- klausa precondition menunjukkan bahwa sebelum
       -- eksekusi prosedur, nilai hitungan harus belum maksimum
       require
          tidak_maksimum :  not maksimum
       do
          hitungan := hitungan + 1
          -- contoh penggunaan instruksi check untuk
          -- memeriksa suatu asersi
          check
             hitungan <= hitungan_maksimum
          end -- check
          -- klausa postcondition menunjukkan bahwa sesudah
          -- eksekusi prosedur, nilai hitungan tidak mungkin nol
          ensure
             tidak_nol : not nol
       end  -- naik
    turun is
          -- prosedur turun menurunkan hitungan sebanyak 1 nilai
       -- klausa precondition menunjukkan bahwa sebelum
       -- eksekusi prosedur, nilai hitungan harus tidak nol
       require
          tidak_nol : not nol
       do
          hitungan := hitungan - 1
       -- klausa postcondition menunjukkan bahwa sesudah
```

```
      -- eksekusi prosedur, nilai hitungan tidak mungkin maksimum
      ensure
         tidak_maksimum : not maksimum
      end  -- turun

   bagi (pembagi : INTEGER) is
         -- prosedur bagi membagi nilai hitungan dengan pembagi
      do
         -- pembagian integer hitungan div pembagi
         hitungan := hitungan // pembagi
         -- klausa rescue akan menset hitungan menjadi maksimum
         -- jika terjadi eksepsi akibat pembagian dengan nol
      rescue
         hitungan := hitungan_maksimum
      end  -- bagi
   invariant
     -- invarian kelas menunjukkan bahwa
     -- fungsi nol dan maksimum tidak mungkin bernilai true sekaligus
 -- dan nilai hitungan harus berada di antara 0 dan hitungan_maksimum
      nol_dan_maksimum_eksklusif : not (nol and maksimum)
      batas_hitungan : (0 <= hitungan) and (hitungan <=
hitungan_maksimum)
end  -- PENGHITUNG
```

Pendefinisian Root Class, sekaligus main program

```
indexing
   description:
     " Kelas KELAS_AWAL :root class (bab 3.9, kelas yg pertama kali %
   %diinstansiasi sebelum kelas-kelas lain dalam program contoh ini %
      % dan berisikan contoh-contoh instruksi %
      % bab 3.6.1 "
   date  : "4 Jan 1998"
   file  : "awal.e"
class KELAS_AWAL
   creation
      mulai
  -- PUBLIK
   feature
      -- contoh deklarasi atribut
      objek1           : KELAS1
      objek_penghitung : PENGHITUNG
      objek_a          : KELAS_A
      -- contoh penggunaan kelas generik, bab 3.5
      objek_character  : KELAS_GENERIK[CHARACTER]
      titik_integer    : TITIK_GENERIK[INTEGER]
      -- contoh struktur kendali dalam fungsi-fungsi berikut
      -- bab 3.6.1
      tanda (i:INTEGER):CHARACTER is
      -- fungsi tanda akan mengembalikan karakter '+' jika i positif,
      -- karakter '-' jika i negatif, atau karakter '0' jika i nol
         do
            -- contoh eksekusi kondisional
            if (i > 0) then
               Result := '+'
            elseif (i < 0) then
               Result := '-'
            else
               Result := '0'
            end  -- if
```

```
         end -- tanda

      angka (i:INTEGER):STRING is
      -- fungsi angka akan mengembalikan string "satu" jika i=1,
      -- string "dua" jika i=2,atau string "bukan"jika tidak keduanya
         do
            -- contoh instruksi percabangan
            inspect i
            when 1 then
               Result := "satu"
            when 2 then
               Result := "dua"
            else
               Result := "bukan"
            end -- inspect
         end  -- angka
      deret(n : INTEGER) : INTEGER is
       -- fungsi deret akan mengembalikan nilai deret dari 1+2+..+n
         local
            i : INTEGER
         do
            -- contoh instruksi iterasi
            from
               i := 1
               Result := 0
            until
               i > n
            loop
               Result := Result + i
            end  -- loop

         end  -- deret
   -- PRIVAT
   feature {NONE}
      mulai is
            -- prosedur creation untuk kelas ini
         do
            -- contoh instruksi instansiasi, bab 3.2.2
            !!objek1.prosedur_creation
            !!objek_penghitung
            -- contoh pemanggilan feature, bab 3.2.2
            objek_penghitung.naik
            objek1.prosedur_set(objek_penghitung.hitungan,tanda(1))
            -- contoh instansiasi polimorfik, bab 3.4.2
            !KELAS_AB!objek_a
         end -- mulai
end  -- KELAS_AWAL
```

```
system
     contoh
root
     KELAS_AWAL ("CONTOH_CLUSTER"): "mulai"
default
      precompiled ("$EIFFEL4\precomp\spec\$PLATFORM\base")
cluster
        CONTOH_CLUSTER:           "C:\Eiffel\Contoh";
end
```

Mesin Karakter

```
indexing
   description:
      " CharMachine adalah suatu mesin abstrak yang terdiri dari %
      % sebuah pita yang berisi deretan karakter dan diakhiri dgn %
      % karakter titik '.', tombol START dan ADV, lampu EOP dan %
      % jendela CC yang berisi satu karakter. "

   date : "4 Jan 1998"
   file : "char_machine.e"

class interface CHAR_MACHINE
   creation make
  -- PUBLIK
   feature     -- Status
      cc   : CHARACTER          -- karakter pada jendela sekarang
      eop  : BOOLEAN is
            -- True jika cc = mark atau pada akhir pita
   feature     -- Konstanta
      Mark : CHARACTER is '.'   -- karakter penanda akhir pita
   feature     -- Tombol / Servis
      start is
            -- mulai menjalankan mesin karakter dengan membuka
            -- membuka file pita

      adv is
            -- maju satu karakter
         -- Precondition : belum mencapai akhir pita
         require
            not_eop : not eop

      set_next(fn : STRING) is
            -- menset nama file pita berikutnya yang akan digunakan
   -- Invarian : atribut path tidak boleh void
   invariant
      path_not_void : path /= Void

end  -- CHAR_MACHINE
```

Mesin Kata : memakai Mesin Karakter

```
indexing
   description:
      " WORD_MACHINE adalah suatu mesin abstrak yang menggunakan %
      % sebuah mesin karakter untuk akses per kata dan mempunyai %
      % tombol STARTWORD dan ADVWORD, lampu EOP %
      % dan jendela CW yang berisi satu kata. "

   date : "4 Jan 1998"
   file : "word_machine.e"

class WORD_MACHINE

   creation make
  -- PUBLIK
   feature     -- Status
      cw   : STRING           -- kata sekarang
      eow  : BOOLEAN is
            -- True jika telah mencapai akhir pita
   feature     -- Tombol / Servis
      start_word is
            -- mulai menjalankan mesin kata dengan
            -- membuka file pita
      adv_word is
            -- maju satu kata
         -- Precondition : belum mencapai akhir pita
         require
            not_eow : not eow
      set_next (fn : STRING) is
            -- menset nama file pita berikutnya yang akan digunakan
   -- Invarian : atribut cw tidak boleh void
   invariant
     cw_not_void         : cw /= Void
end  -- WORD_MACHINE
```

Kelas penguji Mesin Kata

```
class TEST_WM creation
      -- kelas untuk menguji kelas WORD_MACHINE
   make
feature
   wm : WORD_MACHINE
   make is
   do
     io.print("Demo Mesin Kata : %N")
     !!wm.make
     wm.set_next("c:\eiffel\jenisobj\wordm\pita.txt")
     from
        wm.start_word
     until wm.eow
     loop
        io.print(wm.cw)
        io.print("%N")
        wm.adv_word
     end
   end
end
```

```
system
     test_wm
root
     TEST_WM ("WORDM_CLUSTER"): "make"
default
      precompiled ("$EIFFEL4\precomp\spec\$PLATFORM\base")
cluster
        WORDM_CLUSTER:          "C:\Eiffel\JenisObj\WordM";
        CHARM_CLUSTER:          "C:\Eiffel\JenisObj\CharM";
end
```

Kelas generik

```
indexing
   description:
    " Kelas KELAS_GENERIK adalah contoh pendefinisian kelas generik %
      % bab 3.5 "
   date  : "4 Jan 1998"
   file  : "generik1.e"
class KELAS_GENERIK [G]
  -- PUBLIK
  feature
     -- KELAS_GENERIK mempunyai atribut item yang bertipe generik G
     item : G   -- atribut bertipe generik
     set (new_item : G) is
          -- prosedur set mengubah atribut item menjadi new_item yang
           -- juga bertipe generik G
        do
          item := new_item
        end  -- set
end -- KELAS_GENERIK
```

```
indexing
   description:
    " Kelas TITIK_GENERIK adalah contoh pendefinisian kelas generik %
 %yg parametergeneriknya terbatas pd kelas NUMERIC atau subkelasnya %
      % bab 3.5 "
   date  : "4 Jan 1998"
   file  : "generik2.e"
class TITIK_GENERIK [N->NUMERIC]
  -- PUBLIK
  feature    -- akses
    absis : N is
          -- fungsi absis mengembalikan nilai atribut x
       do
          Result := x
       end  -- absis
    ordinat : N is
          -- fungsi ordinat mengembalikan nilai atribut y
       do
          Result := y
       end  -- ordinat
  feature  -- perubahan
    set (new_x, new_y : N) is
          -- prosedur set mengubah atribut x,y menjadi new_x,new_y
       do
          x := new_x
          y := new_y
       end  -- set
    geser (dx, dy : N) is
          -- prosedur geser menambahkan dx dan dy ke x dan y
       do
          x := x + dx
          y := y + dy
       end  -- geser
  -- PRIVAT
  feature {NONE}
    x : N   -- atribut x menunjukkan absis titik
    y : N   -- atribut y menunjukkan ordinat titik
end  -- TITIK_GENERIK
```

Kelas Abstrak

```
indexing
   description:
      "Kelas KELAS_ABSTRAK : contoh pendefinisian kelas abstrak %
      % yang mempunyai operasi-operasi yang belum diimplementasikan %
      % bab 3.4.3 "
   date  : "4 Jan 1998"
   file  : "abstrak.e"
deferred class KELAS_ABSTRAK
  -- PUBLIK
  feature
     -- dua atribut KELAS_ABSTRAK berupa string dan integer
     string_turunan_abstrak  : STRING
     integer_turunan_abstrak : INTEGER
     -- contoh pendefinisian operasi yang belum diimplementasi
     -- bab 3.1.2
     -- prosedur prosedur_turunan_abstrak adalah prosedur
     -- yang belum diimplementasikan
     prosedur_turunan_abstrak is
        deferred
        end  -- prosedur_turunan_abstrak
     -- fungsi fungsi_turunan_abstrak adalah fungsi yang
     -- belum diimplementasikan
     fungsi_turunan_abstrak : INTEGER is
        deferred
        end  -- fungsi_turunan_abstrak
end -- KELAS_ABSTRAK
```

Pemakaian kelas Abstrak lewat inheritance

```
indexing
   description:
   " Kelas KELAS_A: contoh deklarasi inheritance dari KELAS_ABSTRAK %
      % dan mendefinisikan operasi-operasi yg  belum diimplementasi %
      % bab 3.4.1 "
   date  : "4 Jan 1998"
   file  : "kelas_a.e"
class KELAS_A
  -- klausa inheritance berisi deklarasi nama super kelas
  inherit
    KELAS_ABSTRAK
  -- PUBLIK
  feature
     -- selain dari dua atribut KELAS_A yang diturunkan dari
     -- KELAS_ABSTRAK yaitu : string_turunan_abstrak dan
     -- integer_turunan_abstrak, KELAS_A menambahkan satu atribut
     -- lagi berupa real
     real_turunan_kelas_A : REAL
     -- pendefinisian prosedur prosedur_turunan_abstrak
     -- yang belum diimplementasikan pada KELAS_ABSRTAK
     prosedur_turunan_abstrak is
        do
           string_turunan_abstrak := "Kelas A"
        end
     -- pendefinisian prosedur prosedur_turunan_abstrak
     -- yang belum diimplementasikan pada KELAS_ABSRTAK
     fungsi_turunan_abstrak : INTEGER is
        do
           Result := integer_turunan_abstrak * 2
        end
end -- KELAS_A
```

```
indexing
   description:
   " Kelas KELAS_B serupa dg KELAS_A dlm contoh deklarasi %
  % inheritance dr KELAS_ABSTRAK &mendefinisikan operasi-operasi yg %
      % belum diimplementasikan %
      % bab 3.4.1 "
   date  : "4 Jan 1998"
   file  : "kelas_b.e"
class KELAS_B
  -- klausa inheritance berisi deklarasi nama super kelas
  inherit
    KELAS_ABSTRAK
  -- PUBLIK
  feature
     -- selain dari dua atribut KELAS_B yang diturunkan dari
     -- KELAS_ABSTRAK yaitu : string_turunan_abstrak dan
     -- integer_turunan_abstrak, KELAS_B menambahkan satu atribut
     -- lagi berupa boolean
     boolean_turunan_kelas_B : BOOLEAN
     -- pendefinisian prosedur prosedur_turunan_abstrak
     -- yang belum diimplementasikan pada KELAS_ABSRTAK
     prosedur_turunan_abstrak is
     do
        string_turunan_abstrak := "Kelas B"
     end
     -- pendefinisian prosedur prosedur_turunan_abstrak
     -- yang belum diimplementasikan pada KELAS_ABSRTAK
     fungsi_turunan_abstrak : INTEGER is
        do
           Result := integer_turunan_abstrak * 3
        end
end -- KELAS_B
```

Multiple inheritance

```
indexing
   description:
   " Kelas KELAS_AB : contoh deklarasi multiple inheritance dari%
      % KELAS_A dan KELAS_B dan klausa-klausa adaptasi feature %
      % bab 3.4.1 "
   date  : "4 Jan 1998"
   file  : "kelas_ab.e"
class KELAS_AB
  -- feature-feature yang diturunkan dari KELAS_A dan KELAS_B
  -- dan tidak perlu diadaptasi karena tidak terjadi
  -- konflik nama adalah :
  -- string_turunan_abstrak, integer_turunan_abstrak,
  -- real_turunan_A, boolean_turunan_B
  -- feature-feature yang memerlukan adaptasi feature karena
  -- terjadi konflik nama adalah :
  -- prosedur_turunan_abstrak dan fungsi_turunan_abstrak
  inherit
    KELAS_A
       -- klausa perubahan nama feature yang diturunkan dari KELAS_A
       rename
          fungsi_turunan_abstrak as fungsi_turunan_A
       -- feature fungsi_turunan_abstrak akan didefinisi ulang
       redefine
          prosedur_turunan_abstrak
       -- dua versi fungsi_turunan_abstrak dari KELAS_A dan KELAS_B
       -- dipilih fungsi_turunan_abstrak KELAS_A
       select
          fungsi_turunan_A
    end -- klausa adaptasi feature KELAS_A
    KELAS_B
       -- klausa perubahan nama feature yang diturunkan dari KELAS_B
       rename
          fungsi_turunan_abstrak as fungsi_turunan_B
       -- klausa export ini menyebabkan perubahan status ekspor
       -- feature boolean_turunan_B menjadi privat
       export {NONE}
          boolean_turunan_kelas_B
       -- klausa undefine ini  menyebabkan fungsi_turunan_abstrak
       -- dari KELAS_B dihapus sehingga tidak terjadi konflik nama
       -- dengan versi dari KELAS_A
       undefine
          prosedur_turunan_abstrak
    end -- klausa adaptasi feature KELAS_B
  -- PUBLIK
  feature
     -- pendefinisian ulang feature prosedur_turunan_abstrak
     -- dari KELAS_A
     prosedur_turunan_abstrak is
        do
           string_turunan_abstrak := "Kelas AB"
        end
end -- KELAS_AB
```

Kelas Buffer dan turunannya : Contoh Precondition, post Condition

```
indexing
      description: " Kelas BUFFER merepresentasikan suatu buffer yang
menyimpan item berupa  integer yang diakses sebagai antrian. "
      date: "4 Jan 1998"
      file: "buffer.e"

class interface
      BUFFER
creation
      make
feature -- operasi
      get: INTEGER
                  -- mengambil item pada buffer (terlama)
                  -- Precondition : tidak kosong
            require
                  not_empty: not empty
            ensure
                  not_full: not full
      put (i: INTEGER)
                  -- menaruh item pada buffer (terbaru)
                  -- Precondition : tidak penuh
            require
                  not_full: not full
            ensure
                  not_empty: not empty

feature -- status buffer
      empty: BOOLEAN
                  -- True jika buffer kosong
      full: BOOLEAN
                  -- True jika buffer penuh
end -- class BUFFER
```

```
indexing
      description: " Kelas abstrak BUFFER_USER merepresentasikan
proses yang  menggunakan suatu buffer. "
      date: "4 Jan 1998"
      file: "buffer_user.e"
deferred class interface
      BUFFER_USER
feature
      execute                 -- memulai eksekusi proses
end -- class BUFFER_USER
```

```
indexing
      description: " Kelas PRODUCER adalah subkelas dari BUFFER_USER
yang  memproduksi dan meletakkan item ke buffer. "
      date: "4 Jan 1998"
      file: "producer.e"
class interface
      PRODUCER
creation
      make
feature
      execute
                  -- menjalankan aktivitas produksi
end -- class PRODUCER
```

```
indexing
      description: " Kelas CONSUMER adalah subkelas dari BUFFER_USER
yang  mengambil dan mengkonsumsi item dari buffer. "
      date: "4 Jan 1998"
      file: "consumer.e"
class interface
      CONSUMER
creation
      make
feature
      execute
                  -- eksekusi proses konsumsi
end -- class CONSUMER
```

```
system
     prodcons

root
     TEST_PC ("PRODCONS_CLUSTER"): "make"

default
      precompiled ("$EIFFEL4\precomp\spec\$PLATFORM\base-mt")

cluster
      PRODCONS_CLUSTER:           "C:\Eiffel\JenisObj\ProdCons";

end
```