

Class Diagram

Materi :

- Pendahuluan
- Konsep Object dan Class
- Menggambar Class
- Class Diagram
- Study Kasus

Class Diagram

- *Class* adalah sebuah spesifikasi yang jika di-instansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).
- *Class diagram* menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.
- Diagram Class memberikan pandangan secara luas dari suatu sistem dengan menunjukan kelas-kelasnya dan hubungan mereka. Diagram Class bersifat statis; *menggambarkan hubungan apa yang terjadi bukan apa yang terjadi jika mereka berhubungan.*

- Class diagram dapat membantu dalam memvisualisasikan struktur kelas-kelas dari suatu sistem dan merupakan tipe diagram yang paling ditemui dalam pemodelan system berbasis *object-oriented*. Class Diagram memperlihatkan sekumpulan *class*, *interface*, dan *collaborations* dan relasi yang ada didalamnya.
- Selama proses analisa, class diagram memperhatikan aturan-aturan dan tanggung jawab entitas yang menentukan perilaku sistem. Selama tahap desain, class diagram berperan dalam menangkap struktur dari semua kelas yang membentuk arsitektur sistem yang dibuat. Kita memodelkan *class diagram* untuk memodelkan *static design view* dari suatu system.
-

Object

- **Object** adalah gambaran dari entity, baik dunia nyata atau konsep dengan batasan-batasan yang tepat.
- Object bisa mewakili sesuatu yang nyata dalam *domain problem* kita seperti komputer, barang, konsumen, dapat berupa konsep seperti proses penarikan uang, pembayaran, pengembalian buku dan lain-lain.
- Dari object-object ini kita bisa mengabstraksikan candidate class yang mungkin terlibat.

Karakteristik Object

1. **State**, merupakan suatu kondisi / keadaan dari object yang mungkin ada. Status dari object akan berubah setiap waktu dan ditentukan oleh sejumlah property dan relasi dengan object lainnya.
2. **Behavior (sifat)** menentukan bagaimana object merespon permintaan dari object lain dan melambangkan setiap hal yang dapat dilakukan. Sifat ini diimplementasikan dengan sejumlah operasi untuk object.
3. **Identity (identitas)** artinya setiap object yang ada dalam suatu system adalah “unik”.

Cara menemukan Object

1. Pengelompokan berdasarkan kata/frasa benda pada skenario / dokumentasi use case
2. Berdasarkan daftar kategori objek, antara lain:
 - Objek fisik : pesawatTelepon
 - Spesifikasi/rancangan/deskripsi : deskripsiPesawat
 - Tempat : gudang
 - Transaksi : penjualan
 - Butir yang terlibat pada transaksi : barang jualan
 - Peran : pelanggan
 - Wadah : pesawatTerbang
 - Benda yang diwadahi : penumpang

- Organisasi : departemen
- Kejadian : pendaratan
- Proses : reservasi
- Aturan atau kebijakan : aturanDiskon
- Katalog atau rujukan : daftarPelanggan

Candidate Class

- *Candidate class* dapat kita tentukan dengan melihat skenario use case yang telah kita buat. Candidate class tersebut dapat diambil dari kata benda yang muncul pada skenario use case.

Barang

Pembayaran

CashRegister

Struk

Kasir

Penjualan

Class

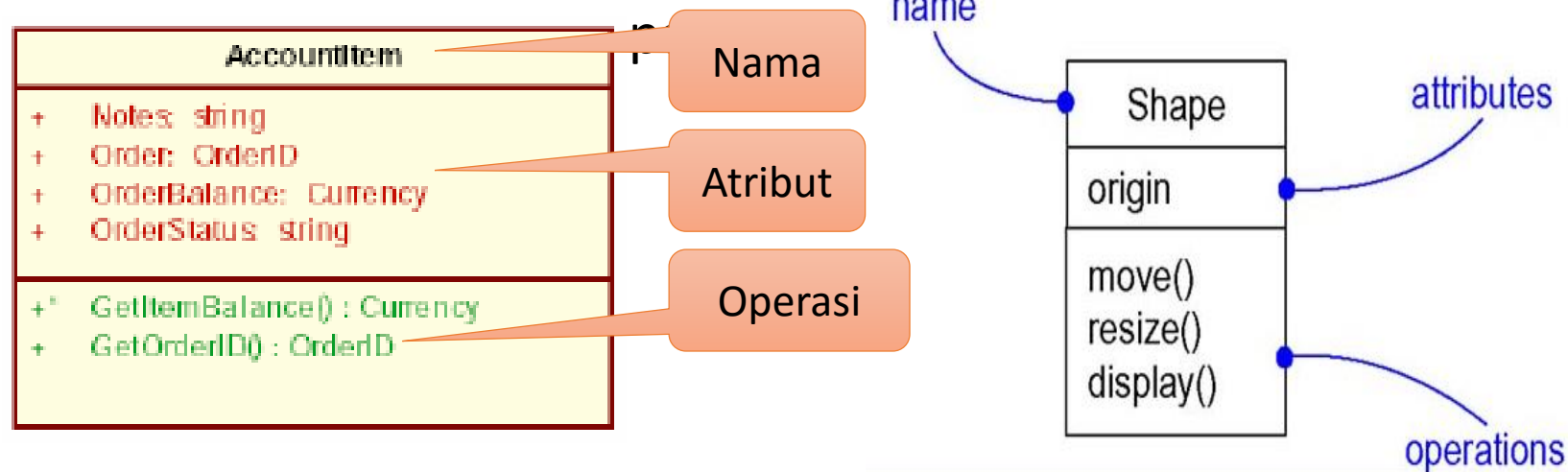
- **Class** adalah deskripsi sekelompok *object* dari property (atribut), sifat (operasi), relasi antar *object* dan semantik yang umum.
- *class* merupakan *blueprint* / *template* / cetakan dari satu atau lebih *object*.
- Setiap *object* merupakan contoh dari beberapa *class* dan *object* tidak dapat menjadi contoh lebih dari satu *class*.

Notasi Class

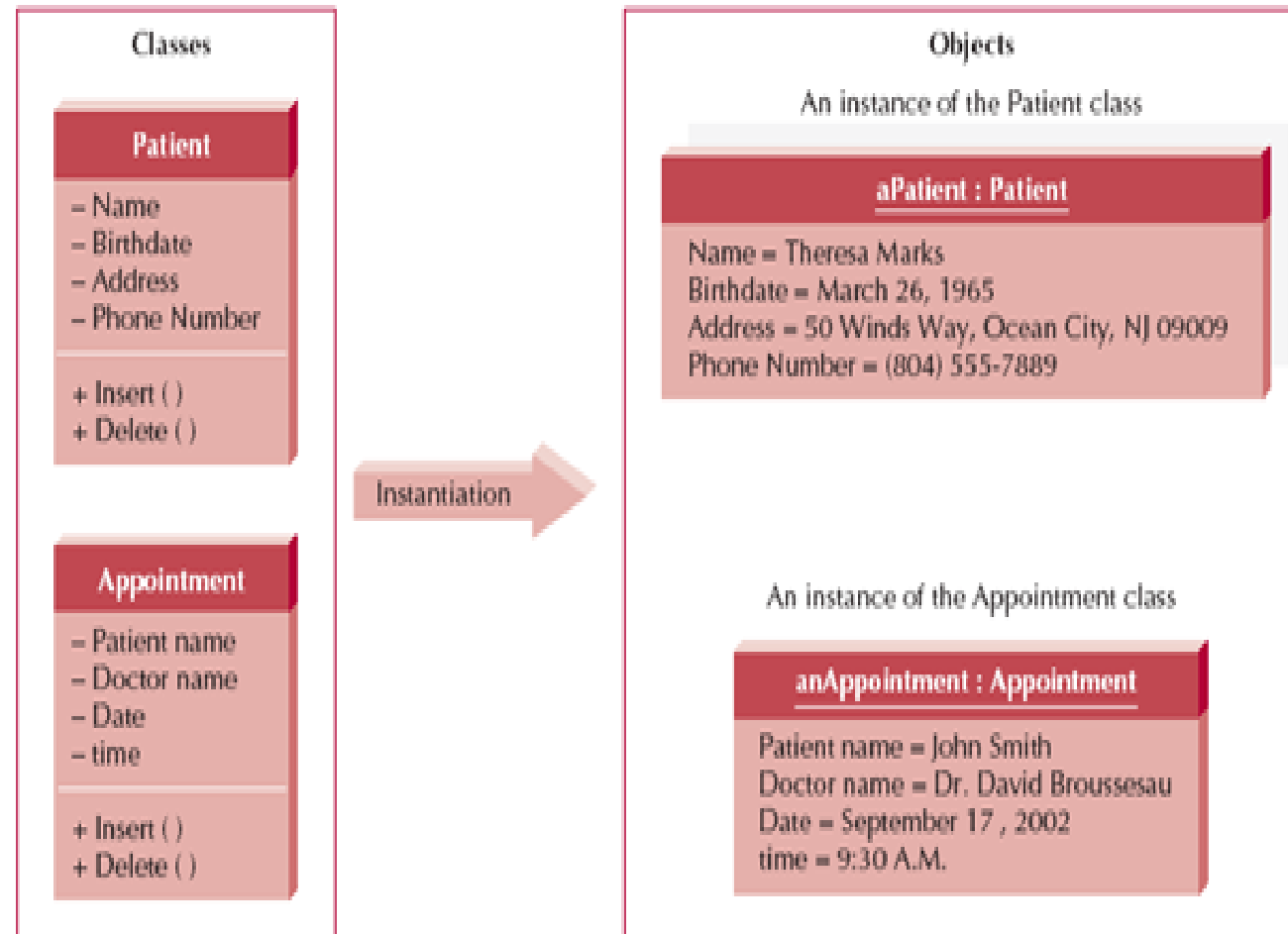
- Penamaan *class* menggunakan kata benda tunggal yang merupakan abstraksi yang terbaik.
- Pada UML, *class* digambarkan dengan segi empat yang dibagi.
- Bagian atas merupakan nama dari *class*. Bagian yang tengah merupakan struktur dari *class* (atribut) dan bagian bawah merupakan sifat dari class (operasi).

Struktur Class

- *Class* memiliki tiga area pokok : Nama (dan stereotype), Atribut, dan Metoda (operasi)
- Atribut dan metoda dapat memiliki salah satu sifat berikut:
 - *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
 - *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
 - *Public*



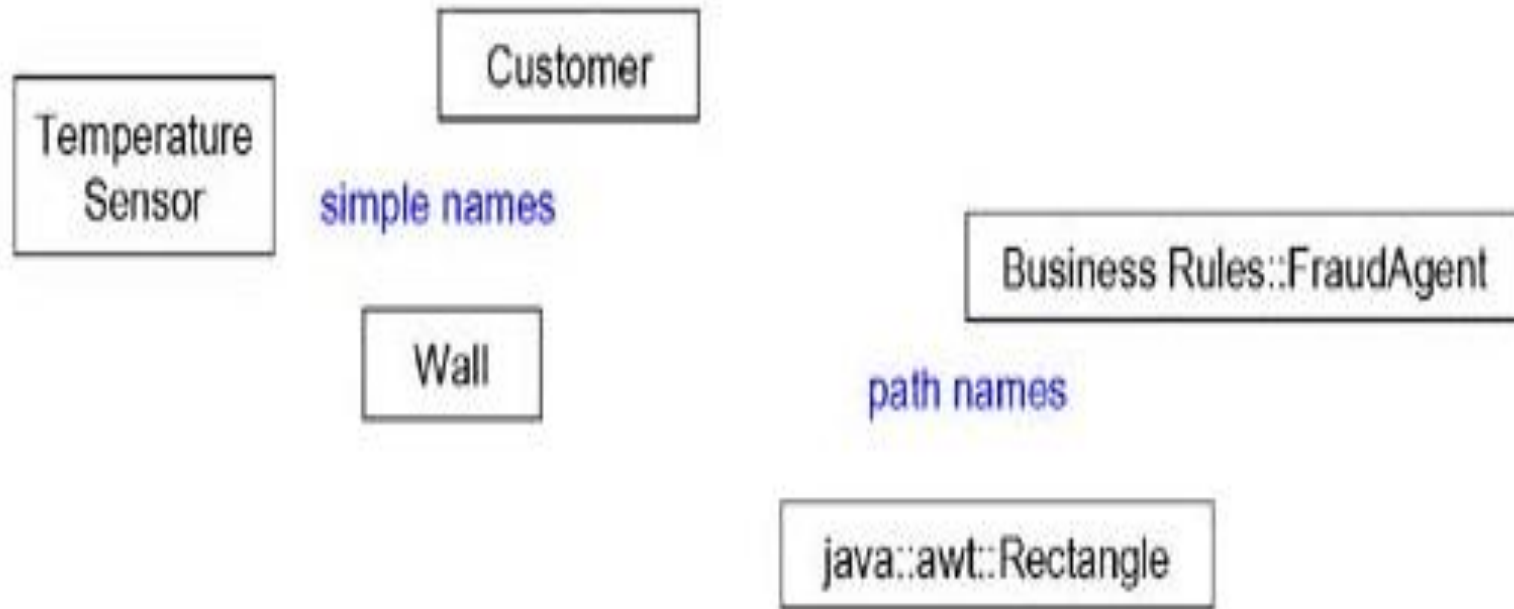
Instance



Penamaan Class

- Setiap kelas harus memiliki sebuah nama yang dapat digunakan untuk membedakannya dari kelas lain.
- Penamaan class menggunakan kata benda tunggal yang merupakan abstraksi yang terbaik.
- Nama kelas dapat dituliskan dengan 2 cara :
 1. Hanya menuliskan nama dari kelas (*simple name*).
 2. Nama kelas diberi *prefix* nama *package* letak class tersebut (*path name*).
- Penulisan nama kelas, huruf pertama dari setiap kata pada nama kelas ditulis dengan menggunakan huruf kapital. Contohnya, Customer dan FraudAgent.

Contoh penamaan Class



Attribute

- Sebuah class mungkin memiliki beberapa *attribute* atau tidak sama sekali.
- Atribut merepresentasikan beberapa *property* dari sesuatu yang kita modelkan, yang dibagi dengan semua *object* dari semua *class* yang ada.
- Contohnya, setiap tembok memiliki tinggi, lebar dan ketebalan
- Untuk penulisan atribut kelas, biasanya huruf pertama dari tiap kata merupakan huruf kapital, kecuali untuk huruf awal. **Contoh :**
birthDate, length.

Cara menemukan atribut

1. Dari dokumentasi use case.

- Contoh : “Pemakai memasukkan nm pegawai, alamat, no ktp
- Di apotik “ Penjualan memasukkan data obat meliputi kode, nama, jenis”

2. Dari memeriksa struktur basisdata

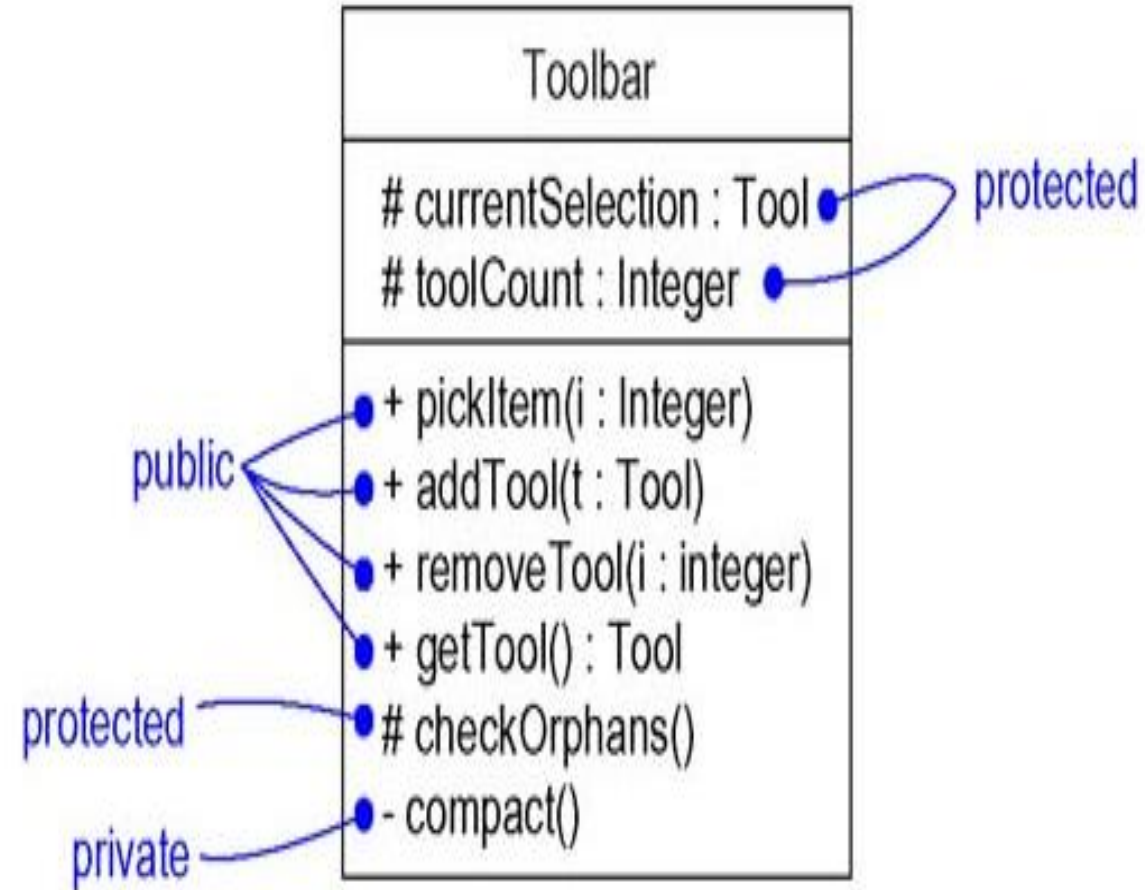
Methods / Operasi

- Methods / Operasi adalah abstraksi dari segala sesuatu yang dapat kita lakukan pada sebuah *object* dan ia berlaku untuk semua *object* yang terdapat dalam *class* tersebut.
- *Class* mungkin memiliki beberapa operasi atau tanpa operasi sama sekali.
- Contohnya adalah sebuah *class* Kotak dapat dipindahkan, diperbesar atau diperkecil.
- Biasanya, pemanggilan operasi pada sebuah *object* akan mengubah data atau kondisi dari *object* tersebut.

Visibility / Sifat Class

- *Visibility* merupakan property yang sangat penting dalam pendefinisian atribut dan operasi pada suatu *class*.
- *Visibility* menspesifikasikan apakah atribut/operasi tersebut dapat digunakan/diakses oleh *class* lain. UML menyediakan 3 buah tingkat *visibility*, yaitu:
 - *Private* (-), tidak dapat dipanggil dari luar *class* yang bersangkutan
 - *Protected* (#), hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
 - *Public* (+), dapat dipanggil oleh siapa saja

Contoh sifat Class



Menggambar Class

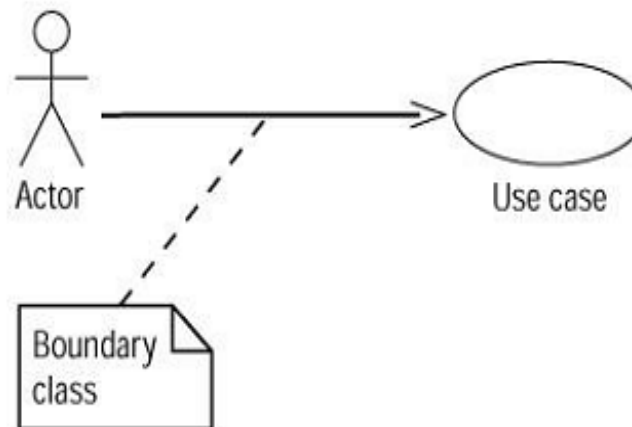
- Ketika menggambarkan sebuah class kita tidak perlu menampilkan seluruh atribut atau operasi.
- Karena dalam sebagian besar kasus kita tidak dapat menampilkannya dalam sebuah gambar, karena terlalu banyaknya atribut atau operasinya, bahkan terkadang tidak perlu karena kurang relevannya atribut atau operasi yang akan ditampilkan.
- Sehingga kita dapat menampilkan atribut dan operasinya hanya sebagian atau tidak sama sekali. Kosongnya tempat pengisian bukan berarti tidak ada. Karena itu kita dapat menambahkan tanda ellipsis (...) pada akhir daftar yang menunjukkan bahwa masih ada atribut atau operasi yang lain.

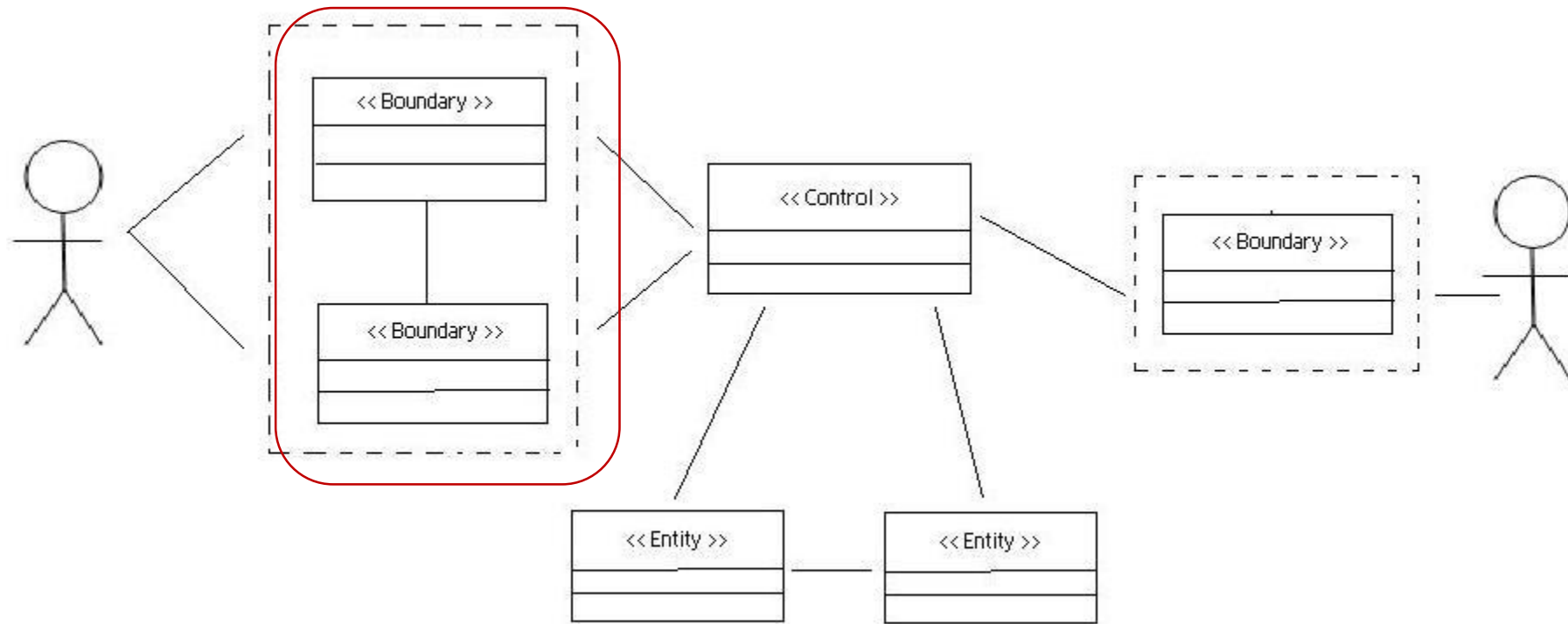
Stereotype Class

- *Stereotype* adalah sebuah mekanisme yang digunakan untuk mengkategorikan sebuah *class*. Misal, kita ingin mencari *form* dalam model. Kita dapat menciptakan *Form stereotype*, dan dapat menemukan seluruh form dalam *stereotype* Form. Fitur ini memudahkan kita dalam mengorganisasi *responsibility* dari tiap-tiap *class*. Sebagai contoh, beberapa *class* yang memiliki stereotype Form memiliki *responsibility* untuk menampilkan dan menerima informasi dari *user*.
- Terdapat 3 *stereotype* utama dalam UML, yaitu *boundary*, *entity* dan *control*.

Boundary Class

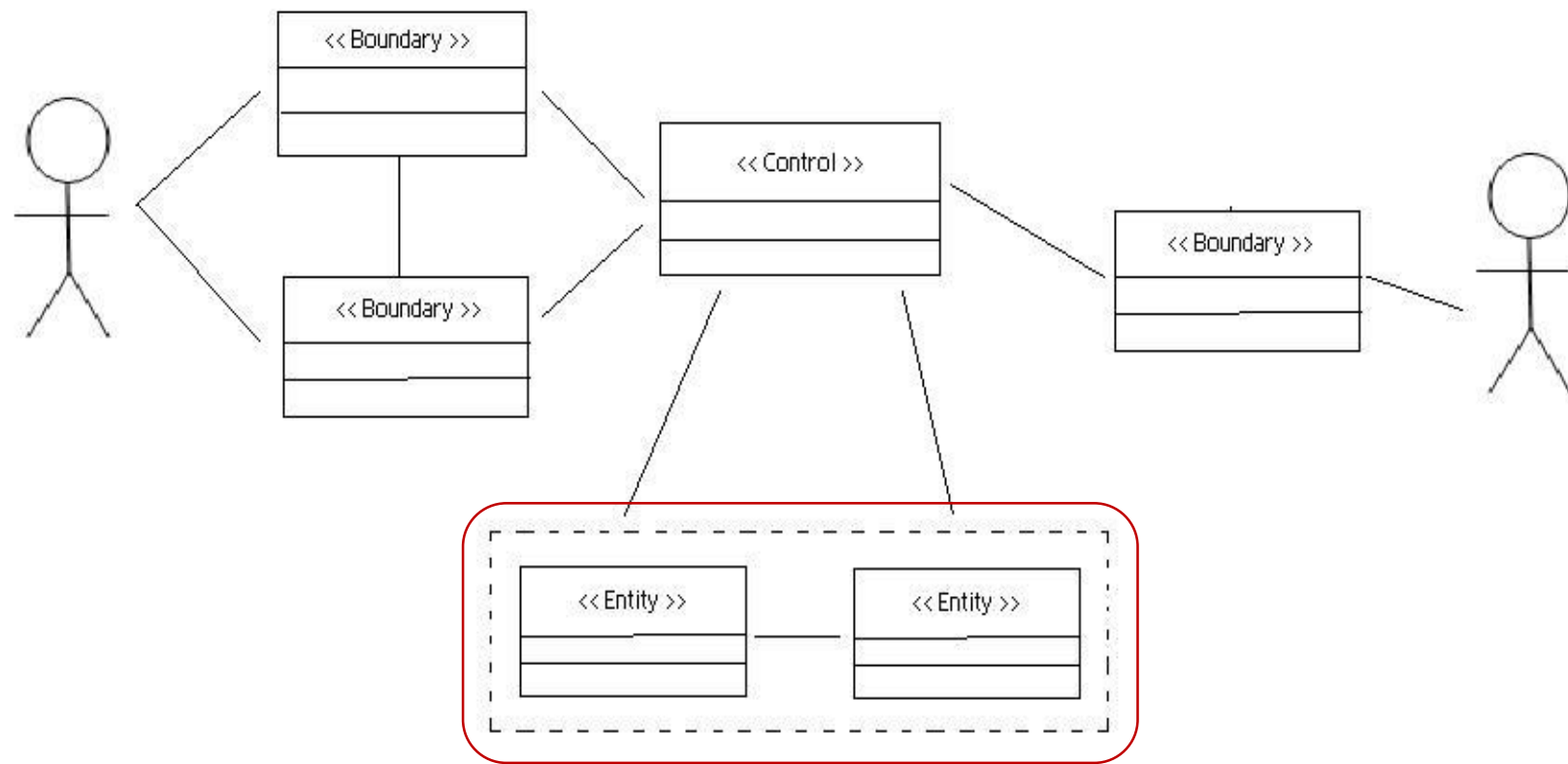
- *Boundary class* adalah *class* yang terdapat batasan sistem dan dunia nyata. Hal ini mencakup semua *form*, *report*, *hardware interface* seperti printer atau scanner.
- *Boundary class* dapat diidentifikasi dari *Use Case Diagram*. Minimal terdapat satu buah *boundary class* dalam relasi *actor* dengan use case. *Boundary class* adalah yang mengakomodasi interaksi antara *actor* dengan sistem.





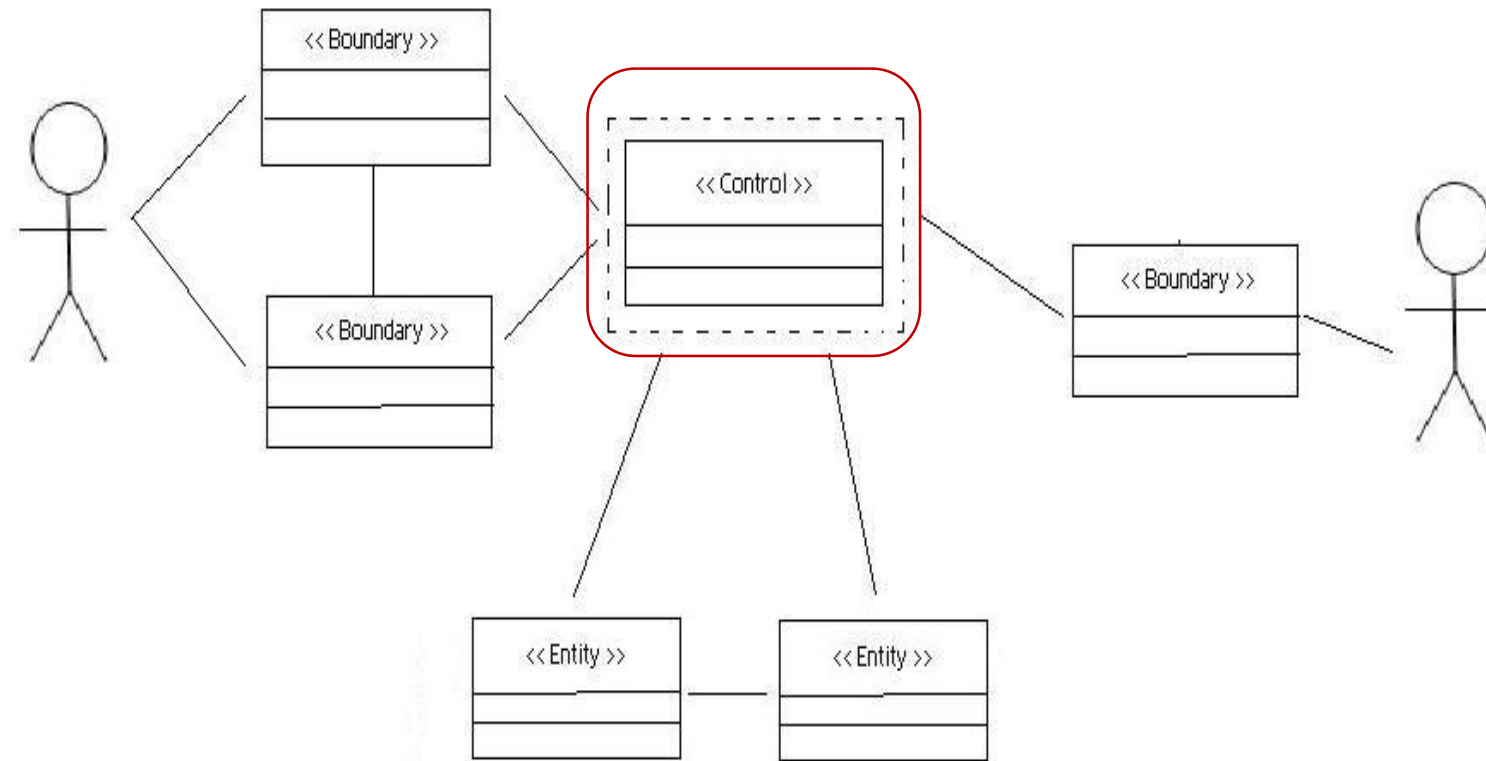
Entity Class

- *Entity class* menyimpan informasi yang mungkin akan disimpan ke sebuah *storage*. Class dengan *stereotype entity* dapat ditemukan di *flow of event* (scenario dari *use-case diagram*) dan *interaction diagram*.
- *Entity class* dapat diidentifikasi dengan mencari kata benda (*noun*) yang ada pada *flow of events*. Selain itu, dapat juga diidentifikasi dari struktur *database* (dilihat dari nama-nama tabelnya). Sebuah *entity class* mungkin perlu dibuat untuk sebuah tabel. Bila sebuah table menyimpan informasi secara permanen, maka *entity class* akan menyimpan informasi pada *memory* ketika sistem sedang *running*.



Control Class

- *Control class* bertanggung jawab dalam mengatur kelas-kelas yang lain. Seperti yang terlihat pada gambar di bawah, *control class* bertanggung jawab dalam mendelegasikan *responsibility* kepada kelas lain.
- *Control class* juga bertanggung jawab dalam mengetahui dan menyampaikan *business rule* dari sebuah organisasi. *Class* ini menjalankan *alternate flow* dan mampu mengatasi *error*. Karena alasan ini *control class* sering disebut sebagai *manager class*.



Relationship

- Relasi atau *relationship* menghubungkan beberapa objek sehingga memungkinkan terjadinya interaksi dan kolaborasi diantara objek-objek yang terhubung.
- Dalam pemodelan *class diagram*, terdapat tiga buah relasi utama yaitu ***association***, ***agregation*** dan ***generalization***.

Bentuk Relationships

Diagram Class mempunyai 3 macam relationships (hubungan):

- **Association.** Suatu hubungan antara bagian dari dua kelas. Terjadi *association* antara dua kelas jika salah satu bagian dari kelas mengetahui yang lainnya dalam melakukan suatu kegiatan. Di dalam diagram, sebuah *association* adalah penghubung yang menghubungkan dua kelas.
- **Aggregation.** Suatu association dimana salah satu kelasnya merupakan bagian dari suatu kumpulan. Aggregation memiliki titik pusat yang mencakup keseluruhan bagian. Sebagai contoh : OrderDetail merupakan kumpulan dari Order.
- **Generalization.** Suatu hubungan turunan dengan mengasumsikan satu kelas merupakan suatu *superClass* dari kelas yang lain. *Generalization* memiliki tingkatan yang berpusat pada *superClass*. *Contoh* : Payment adalah *superClass* dari Cash, Check, dan Credit.

Asosiasi

- Relasi asosiasi merupakan relasi structural yang menspesifikasikan bahwa satu objek terhubung dengan objek lainnya. Relasi ini tidak menggambarkan aliran data, sebagaimana yang terdapat pada pemodelan desain pada analisa terstruktur.
- Relasi asosiasi dapat dibagi menjadi 2(dua) jenis, yaitu *uni-directional association* dan *bi-directional association*

Uni-directional

- Objek pilot memiliki *uni-directional association* dengan objek pesawat.
- Relasi *uni-directional* diatas memungkinkan objek pilot untuk memanggil *property* dari objek pesawat. Namun tidak berlaku sebaliknya. Objek pesawat tidak dapat mengakses *property* dari objek pilot.



Bi-directional

- Objek pilot dapat memanggil *property* yang dimiliki oleh objek pesawat. Begitu juga sebaliknya, objek pesawat juga dapat memanggil *property* dari objek pilot.

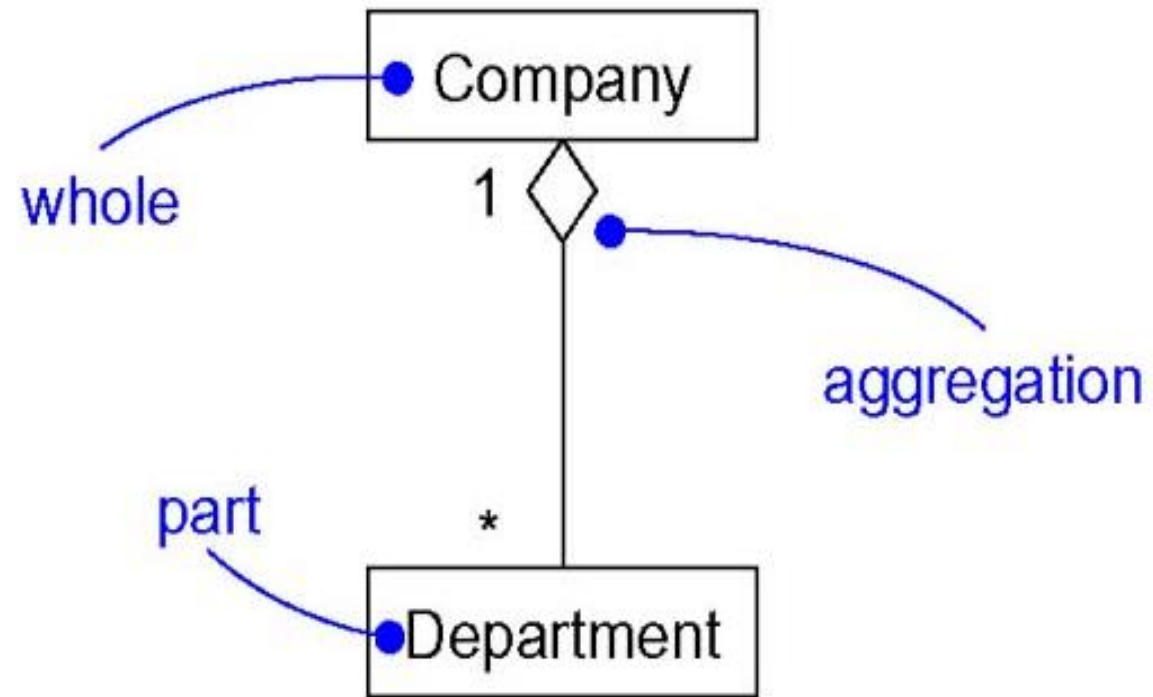


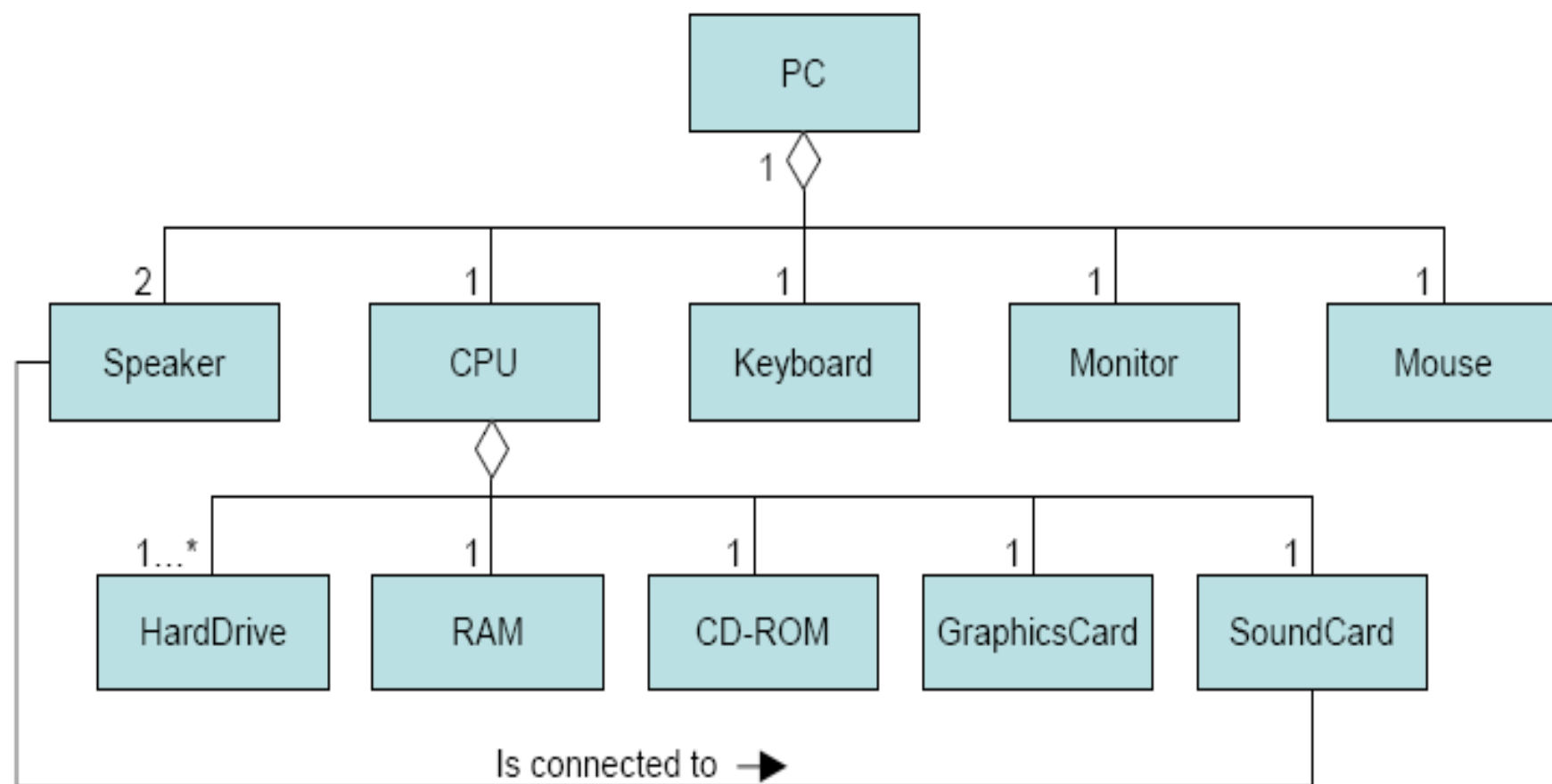
Asosiasi

- Hubungan *association* mempunyai 2 titik. Salah satu titik bisa memiliki label untuk menjelaskan *association* tersebut. *Contoh* : OrderDetail adalah line Item untuk setiap permintaan.
- Panah *navigability* (pengatur alur arah) dalam suatu *association* menggambarkan arah mana *association* dapat ditransfer atau disusun.
- Seperti dalam contoh : OrderDetail dapat disusun dari item-nya, namun tidak bisa sebaliknya. Panah ini juga menjelaskan siapa “memiliki” implementasi dari *association*; dalam kasus ini OrderDetail memiliki Item. *Association* tanpa arah panah merupakan *bidirectional* (bolak-balik).

Aggregation

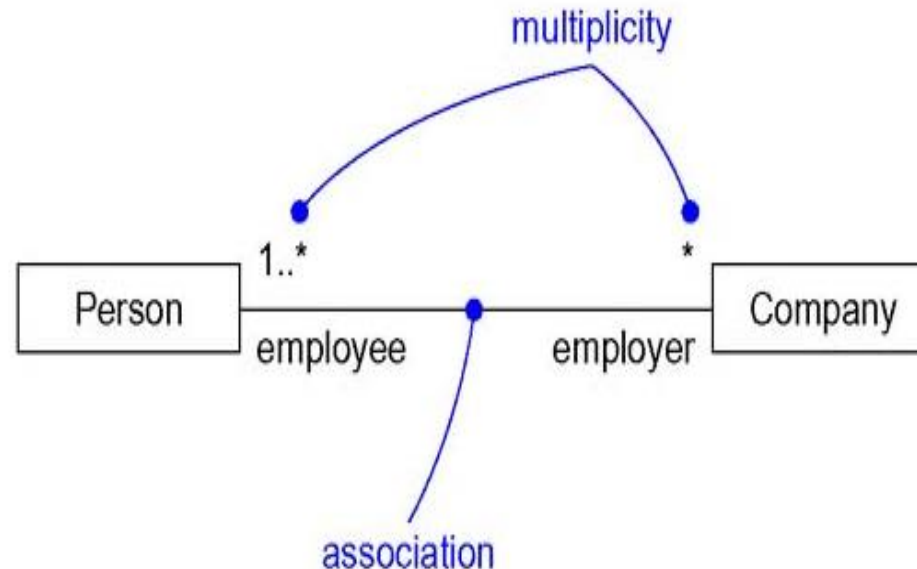
- Aggregation merupakan bentuk khusus dari asosiasi dimana induk terhubung dengan bagian-bagiannya.
- *Aggregation* merepresentasikan relasi “*has-a*”, artinya sebuah *class* memiliki/terdiri dari bagian-bagian yang lebih kecil.
- Dalam UML, relasi agregasi digambarkan dengan *open diamond* pada sisi yang menyatakan induk (whole)





Multiplicity

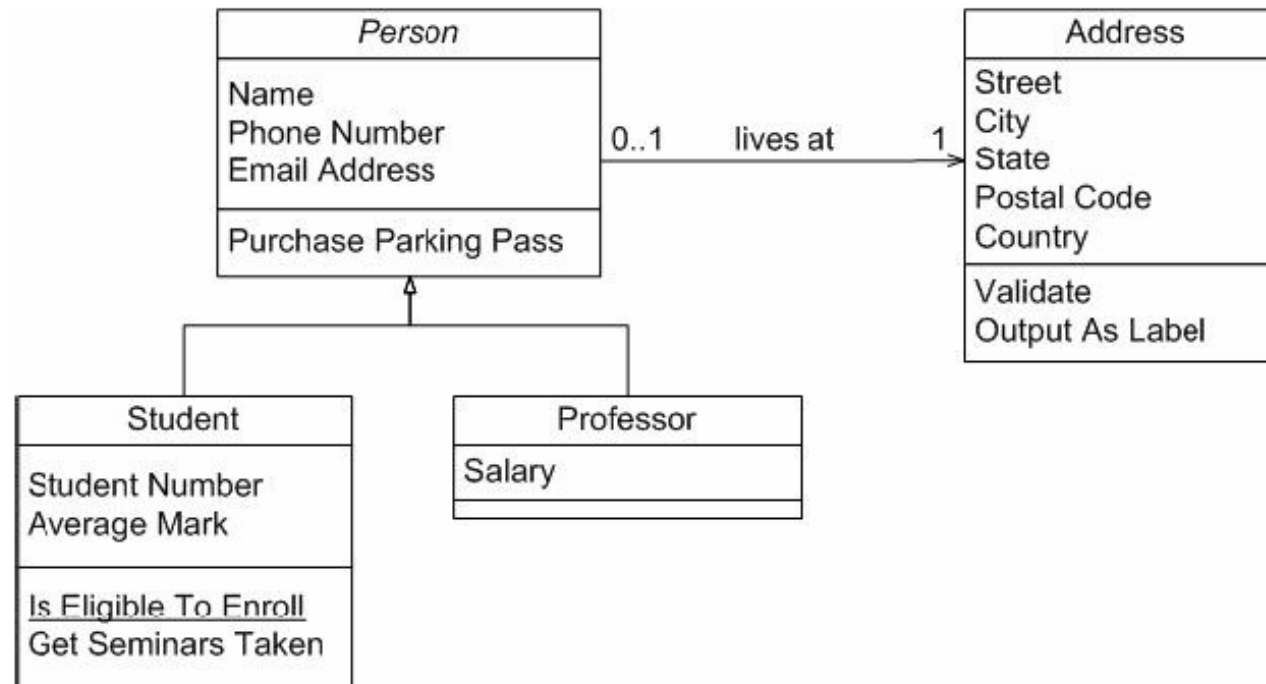
- *Multiplicity* menentukan/mendefinisikan banyaknya *object* yang terhubung dalam suatu relasi.
- Indikator *multiplicity* terdapat pada masing-masing akhir garis relasi, baik pada **asosi**



Multiplicity

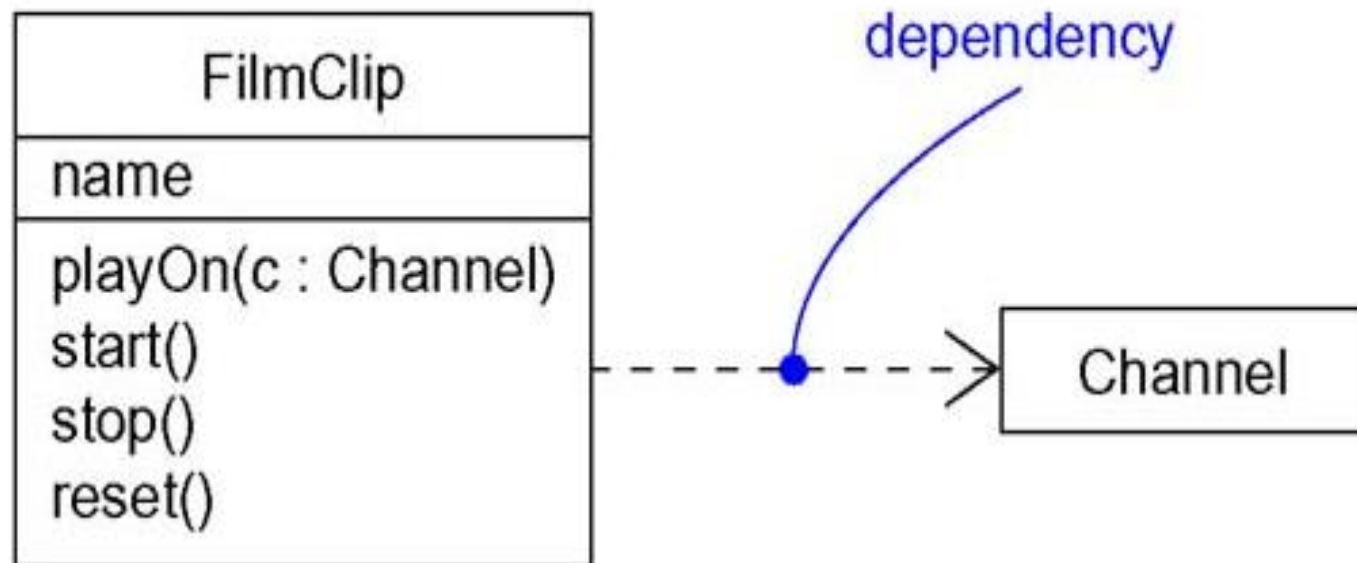
- *Multiplicity* dari suatu titik *association* adalah angka kemungkinan bagian dari hubungan kelas dengan single *instance* (bagian) pada titik yang lain.
- *Multiplicity* berupa single number (angka tunggal) atau range number (angka batasan). Pada contoh, hanya bisa satu 'Customer' untuk setiap 'Order', tapi satu 'Customer' hanya bisa memiliki beberapa 'Order'.

Multiplicities	Artinya
0..1	Nol atau satu bagian. Notasi $n \dots m$ menerangkan n sampai m bagian.
0..* or *	Tak hingga pada jangkauan bagian (termasuk kosong).
1	Tepat satu bagian
1..*	Sedikitnya hanya satu bagian



Dependency

- *Dependency* merupakan sebuah relasi yang menyebutkan bahwa **perubahan pada satu *class*** (misal *class* event), **maka akan mempengaruhi *class* lain yang menggunakannya** (misal *class* window), tetapi tidak berlaku sebaliknya.
- Pada umumnya, relasi *dependency* dalam konteks *Class* Diagram, digunakan apabila terdapat satu *class* yang menggunakan / meng-*instance class* lain **sebagai argumen dari sebuah method**.
- Perhatikan contoh dibawah, bila spesifikasi dari *class* Channel berubah, maka method playOn pada *class* FilmClip juga akan berubah.

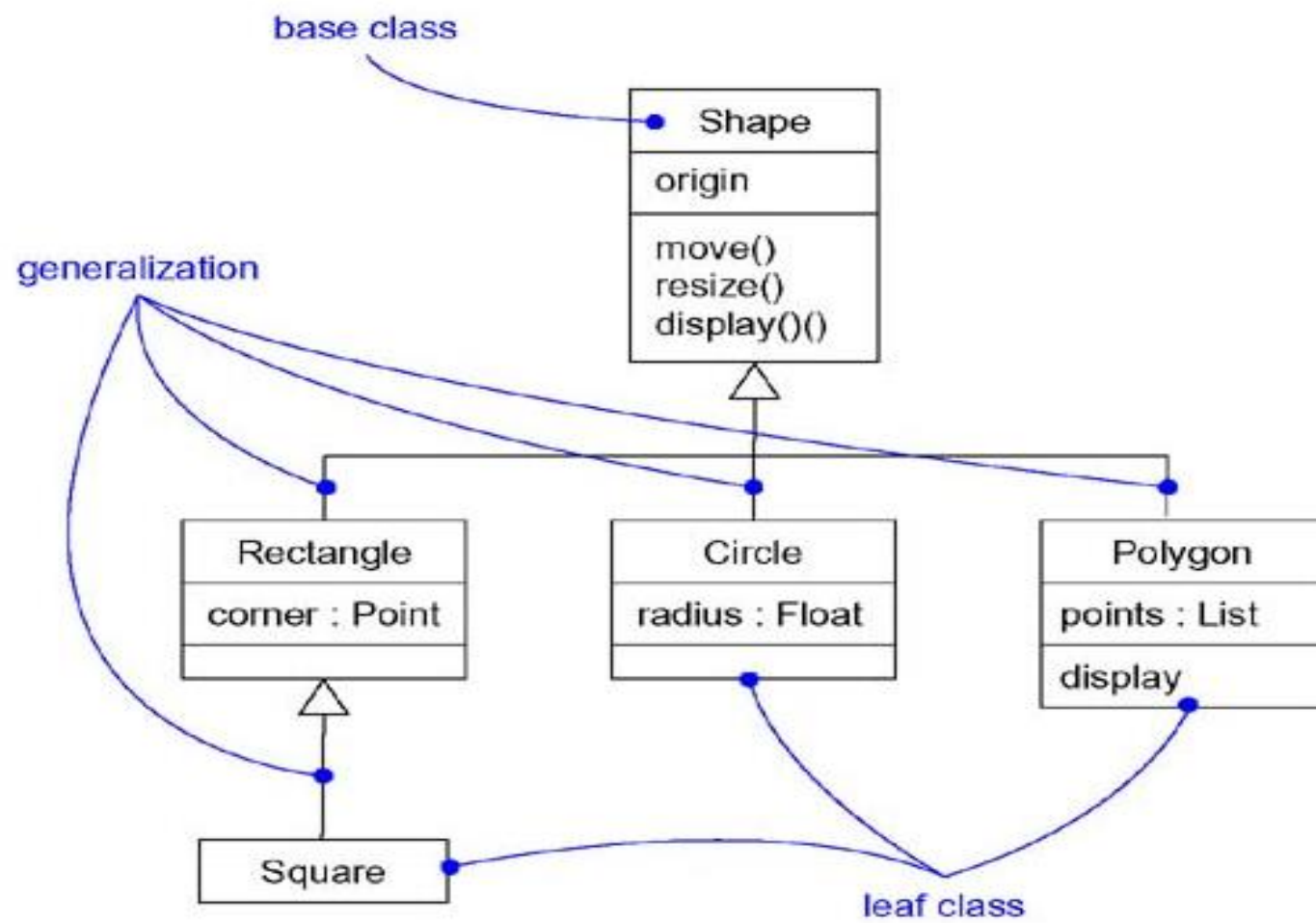


Inheritance

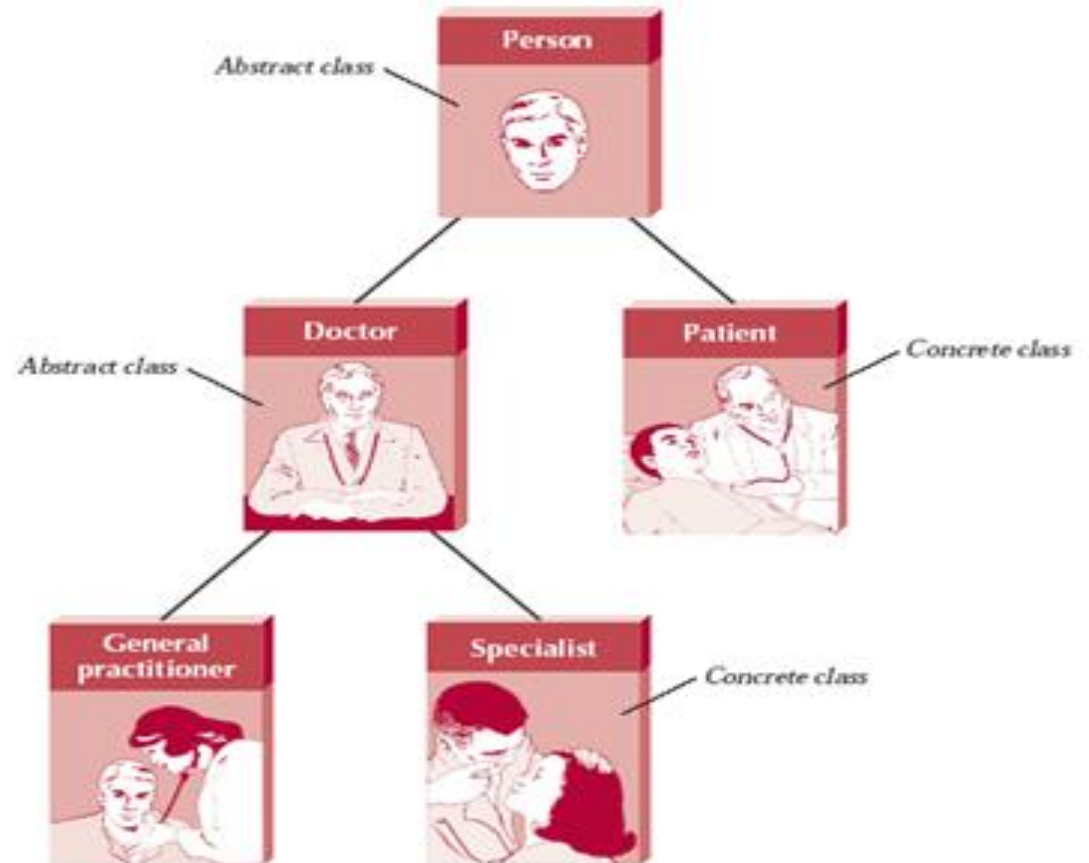
- *Inheritance* merupakan salah satu karakteristik dalam pemrograman berorientasi objek, dimana sebuah *class* mewarisi / *inherit* sifat-sifat (dalam hal ini atribut & operasi) dari *class* lain yang merupakan *parent* dari *class* tadi. *Class* yang menurunkan sifat-sifatnya disebut ***superclass***, sedangkan *class* yang mewarisi sifat dari *superclass* disebut ***subclass***.
- *Inheritance* disebut juga hierarki “is-a” (adalah sebuah) atau “*kind-of*” (sejenis). *Subclass* dapat memiliki atau menggunakan atribut & operasi tambahan yang hanya berlaku pada tingkat hierarkinya.
- Karena *inheritance relationship* bukan merupakan *relationship* diantara objek yang berbeda, maka *relationship* ini tidak diberi nama. Begitu pula dengan penamaan *role* dan *multiplicity*

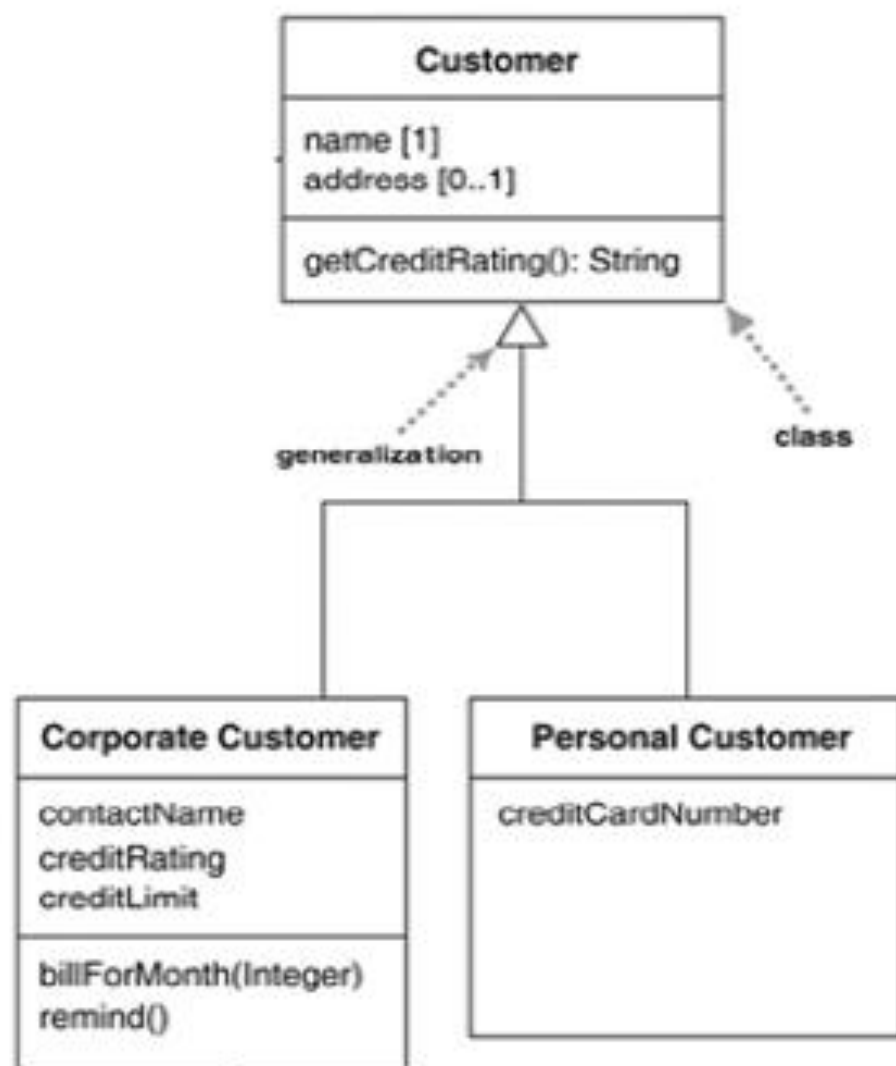
Generalization

- Abstract class: jika satu class hanya diperlukan sebagai template untuk class-class yang lebih spesifik (dalam sistem tidak akan ada object dari class tsb)
- Tidak memiliki instance
- Dalam notasi namanya dituliskan huruf miring (*italic*)

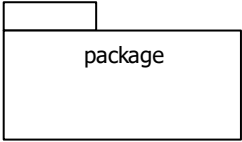
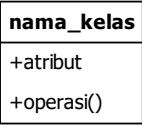


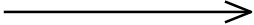


- class person merupakan generalisasi, class doctor dan class patient adalah spesialisasi.
- class yang menjadi superclass (dalam hal ini adalah class person) dinamakan abstract class, sedangkan class yang menjadi subclass (class doctor dan class patient) dinamakan concrete class


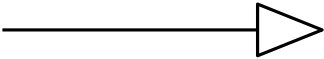






Simbol Class Diagram

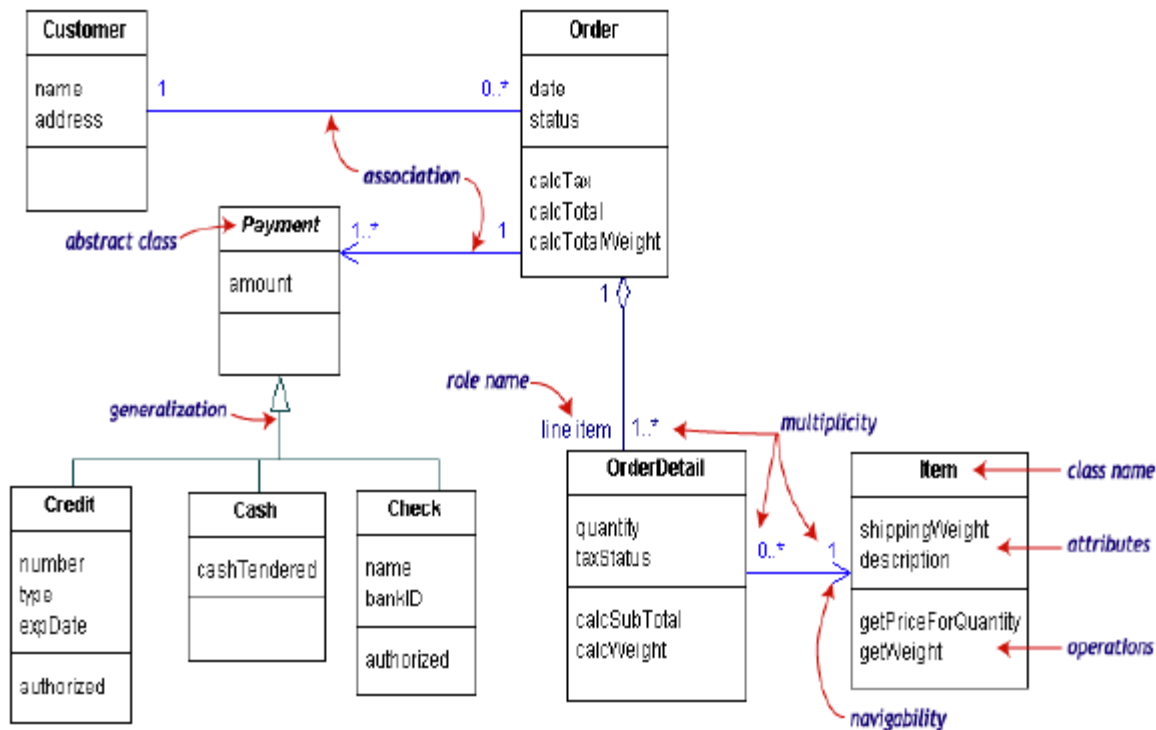
Simbol	Deskripsi
<p><i>package</i></p> 	<p><i>package</i> merupakan sebuah bungkus dari satu atau lebih kelas</p>
<p>kelas</p> <p>-</p> 	<p>kelas pada struktur sistem</p>
<p>antarmuka / <i>interface</i></p> 	<p>sama dengan konsep <i>interface</i> dalam pemrograman berorientasi objek</p>
<p>asosiasi / <i>association</i></p> 	<p>relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan <i>multiplicity</i></p>
<p>asosiasi berarah / <i>directed association</i></p> 	<p>relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan <i>multiplicity</i></p>

Simbol (2)

asosiasi berarah / <i>directed association</i> 	relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan <i>multiplicity</i>
generalisasi 	relasi antar kelas dengan makna generalisasi-spesialisasi (umum khusus)
kebergantungan / <i>dependency</i> 	relasi antar kelas dengan makna kebergantungan antar kelas
agregasi / <i>aggregation</i> 	relasi antar kelas dengan makna semua-bagian (<i>whole-part</i>)

Contoh class diagram

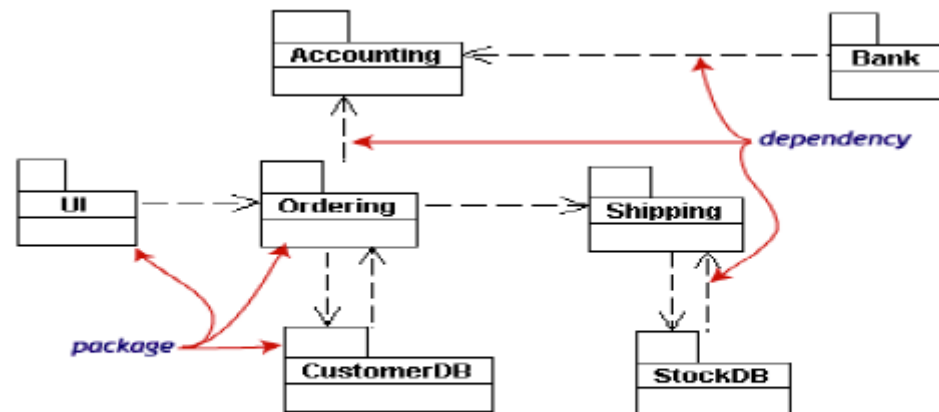
- Setiap diagram Class memiliki *Class* (kelas), *association*, dan *multiplicity*. Sedangkan *navigability* (alur arah) dan *role* (kegiatan) merupakan optional (tidak diharuskan)



Gambar 2. Contoh Diagram Class transaksi Pembelian barang

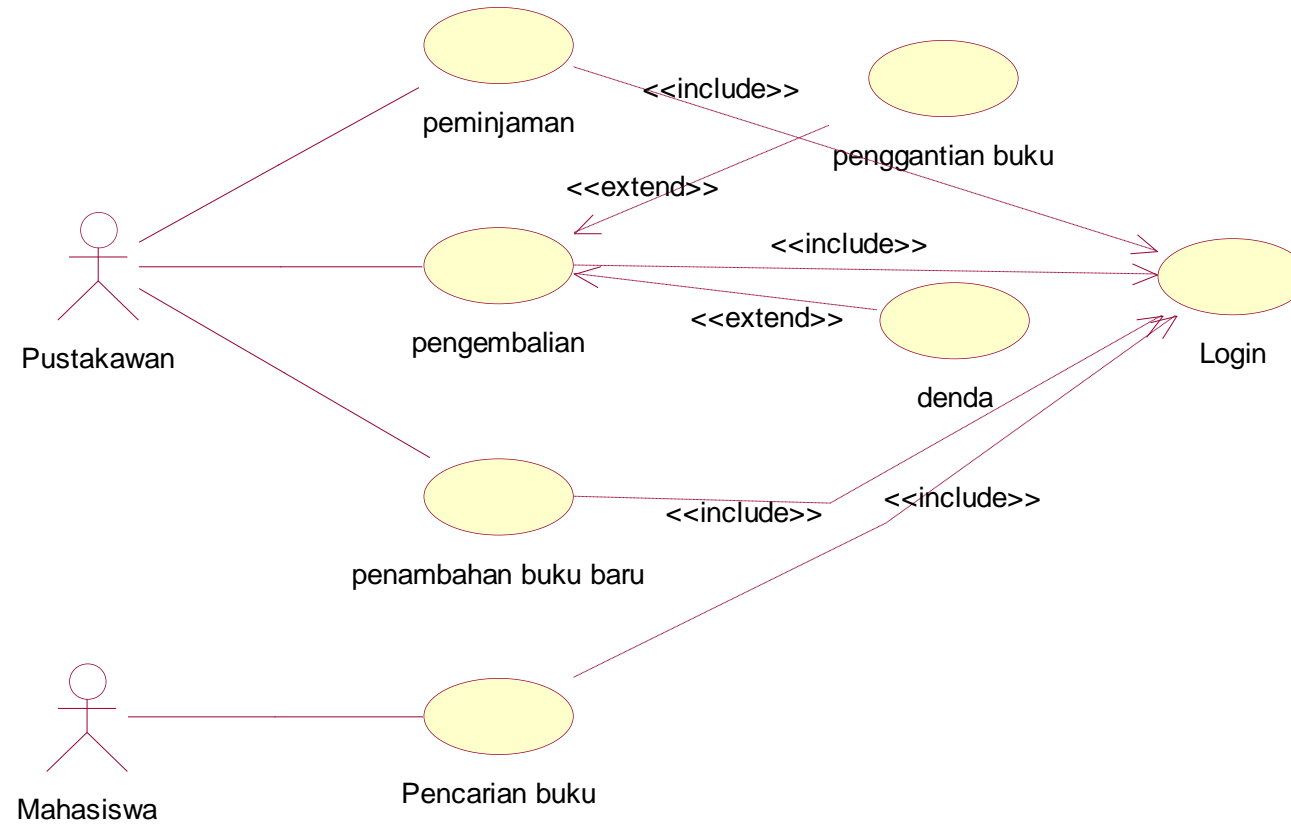
Package Diagram

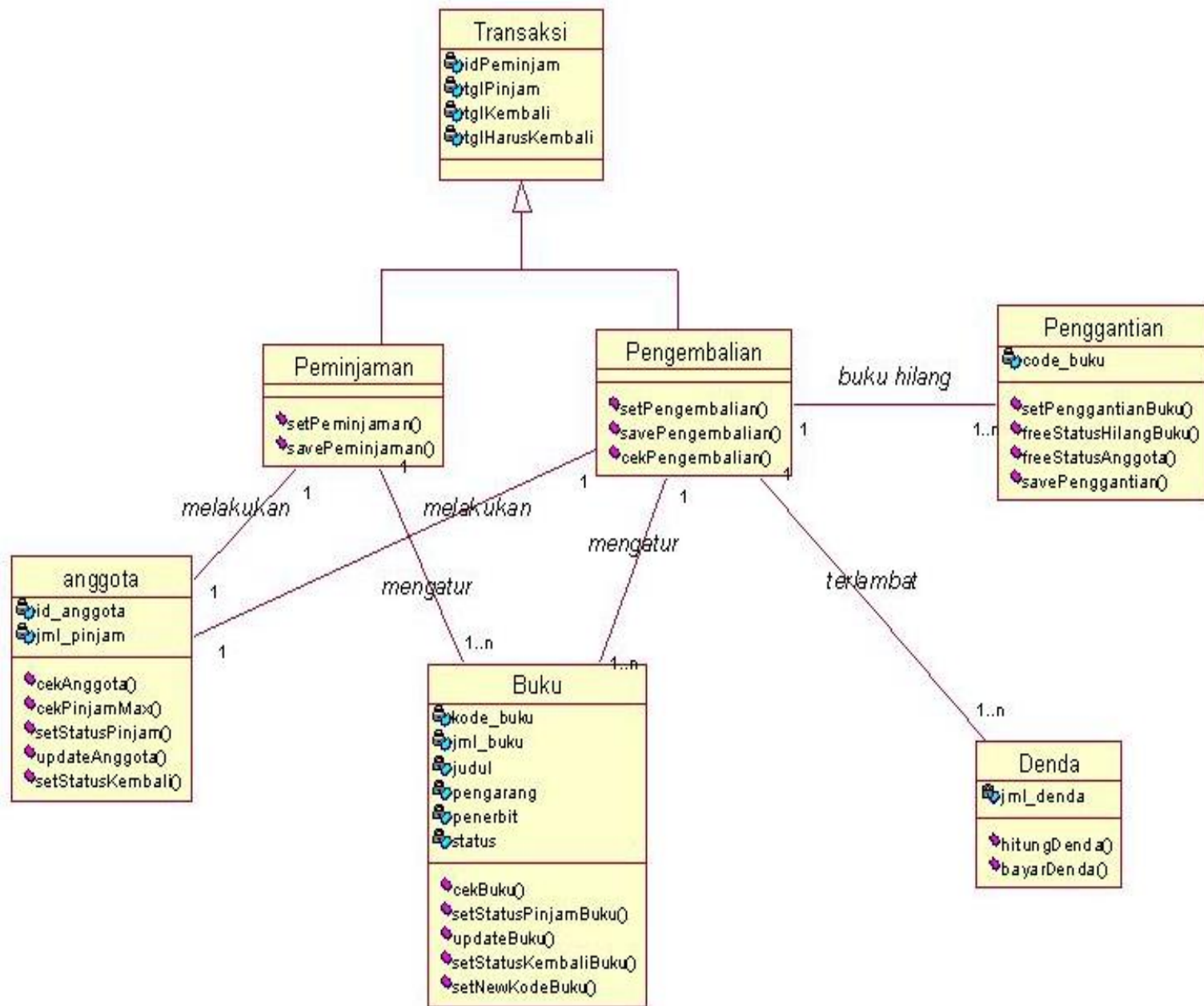
- Untuk mengatur pengorganisasian diagram Class yang *kompleks*, dapat dilakukan pengelompokan kelas-kelas berupa *package* (paket).
- *Package* adalah kumpulan elemen-elemen logika UML.
- Gambar di bawah ini mengenai model bisnis dengan pengelompokan kelas-kelas dalam bentuk paket-paket :



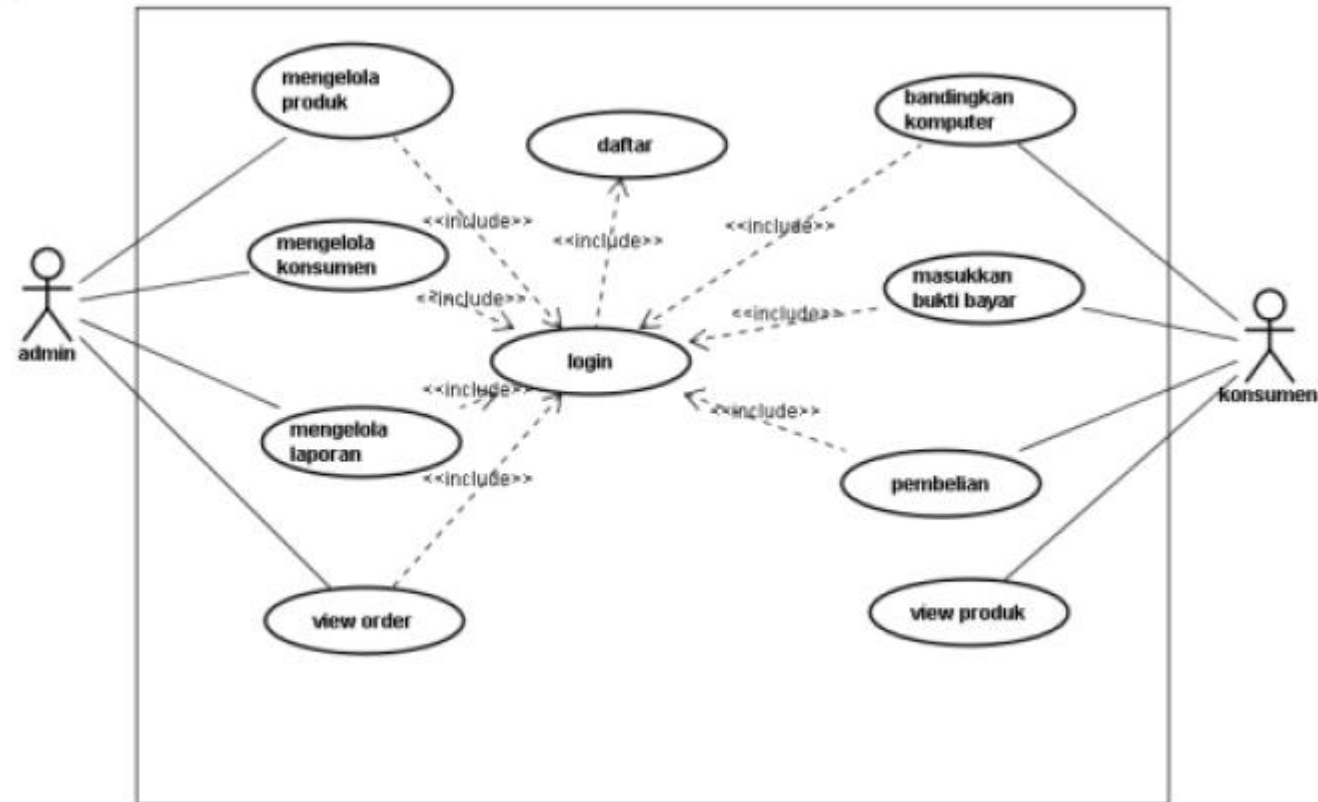
Gambar 3. Contoh Diagram Package

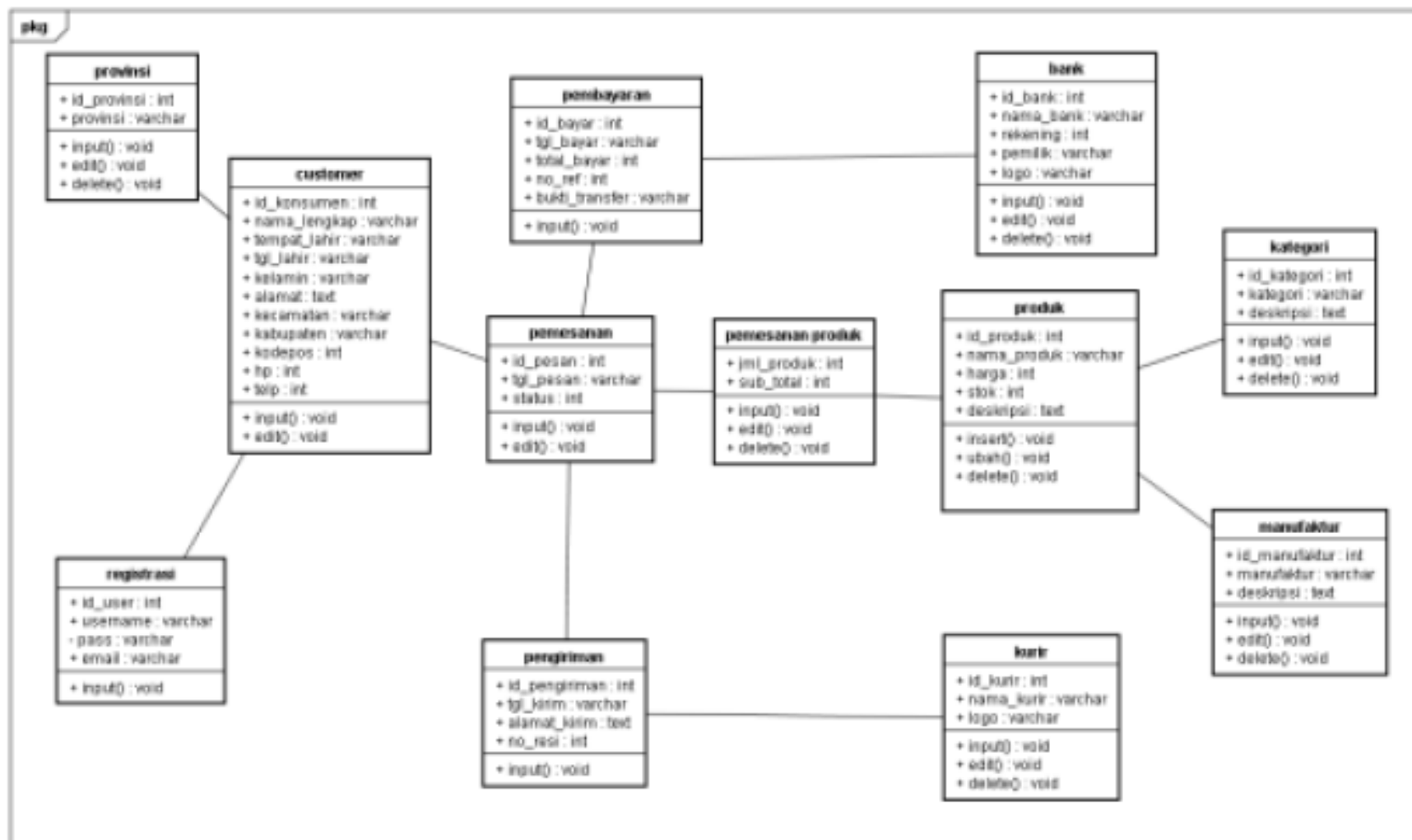
Case (1) : Perpustakaan



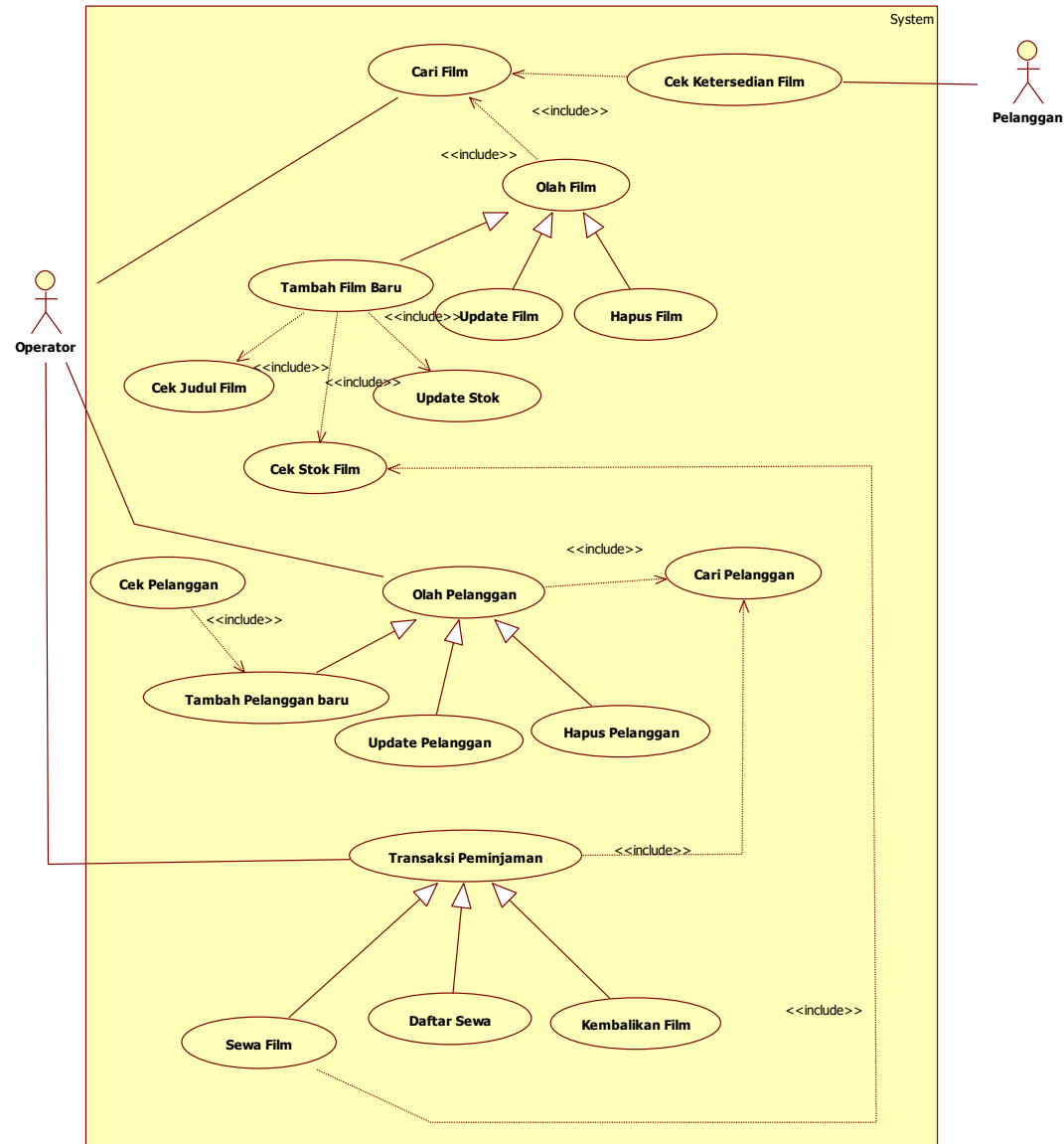


Case (2) : Penjualan Online

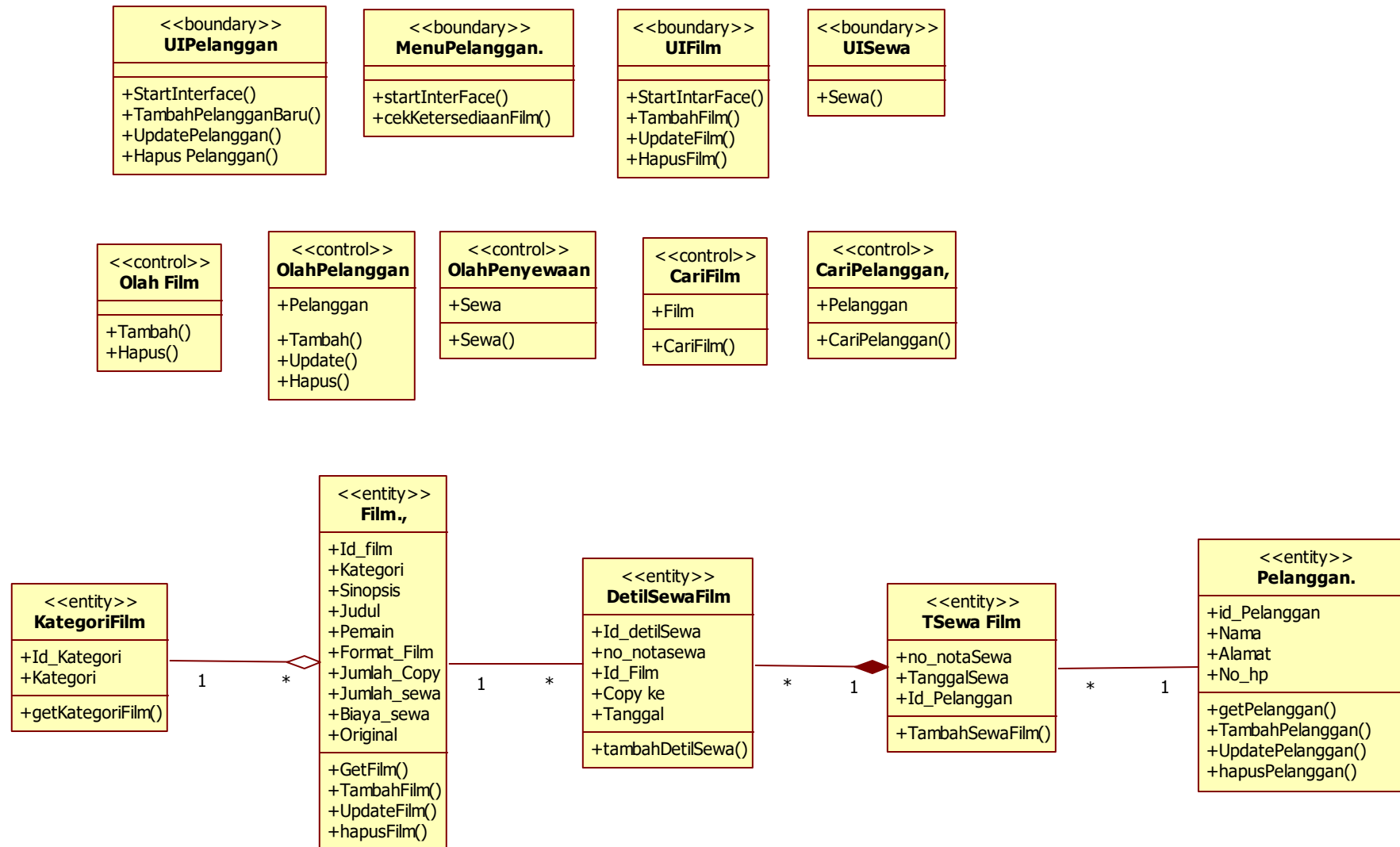




Case (3) : Rental Film



-



Referensi

- Catur Iswahyudi + Edhy Sutanta, UML, Class Diagrams
- <http://www.agilemodeling.com/>
- <http://www.visual-parActivityDiagramigm.com/VPGallery/diagrams/index.html>