2 UNIFIED MODELLING LANGUAGE

Obyektif :

- **Mengenal komponen diagram-diagram yang ada pada UML**
- **Memahami penggunaan diagram dengan baik sesuai dengan kondisi dan sudut pandang.**
- **Memahami konsep relasi antar class**

2.1. Model-model diagram pada UML.

UML terdiri atas banyak elemen-elemen grafis yang digabungkan membentuk diagram. Tujuan representasi elemen-elemen grafis ke dalam diagram adalah untuk menyajikan beragam sudut pandang dari sebuah sistem berdasarkan fungsi masing-masing diagram tersebut. Kumpulan dari beragam sudut pandang inilah yang kita sebut sebuah **model**.

Diagram-diagram yang tersedia pada UML versi 1.4 ada 9 diagram yang terbagi atas 2 kategori, yaitu :

Structural Diagram :

- *Class diagram*
- *Object diagram*
- *Component diagram*
- *Deployment diagram*

Behavioral Diagram :

- *Use case diagram*
- *Sequence diagram*
- *Collaboration diagram*
- *Statechart diagram*
- *Activity diagram*

2.1.1. Class Diagram

Class Diagram adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah obyek dan merupakan inti dari pengembangan dan desain berorientasi obyek. *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 *object* beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lainlain.

Sebuah *Class* memiliki tiga area pokok :

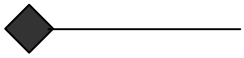
1. Nama, merupakan nama dari sebuah kelas
2. Atribut, merupakan peroperti dari sebuah kelas. Atribut melambangkan batas nilai yang mungkin ada pada obyek dari class
3. Operasi, adalah sesuatu yang bisa dilakukan oleh sebuah *class* atau yang dapat dilakukan oleh *class* lain terhadap sebuah *class*.

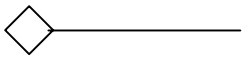
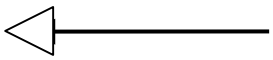
Atribut dan metoda dapat memiliki salah satu sifat berikut :

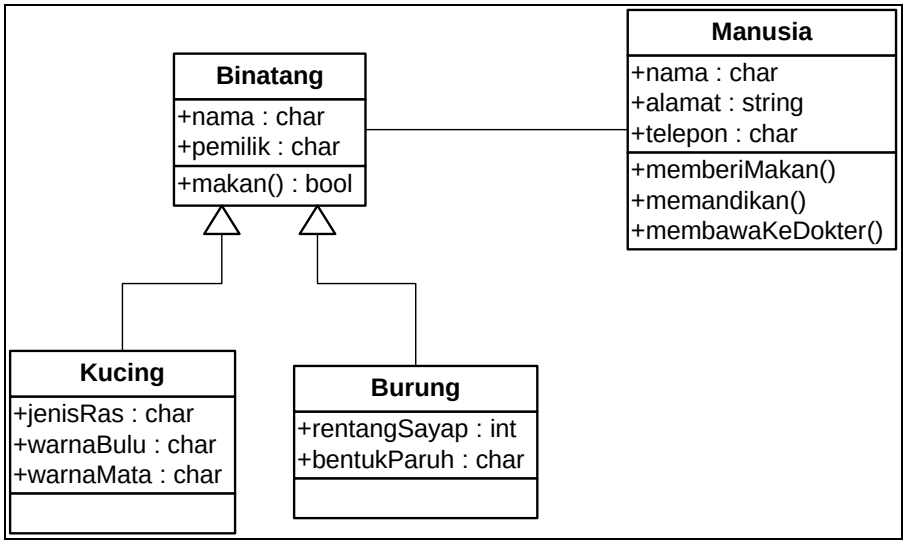
- *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
- *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
- *Public*, dapat dipanggil oleh siapa saja.
- *Package*, hanya dapat dipanggil oleh instance sebuah class pada paket yang sama.

Berikut adalah notasi – notasi yang ada pada class diagram :

Tabel 2.1. Notasi pada Class Diagram

Class	*Class* adalah blok - blok pembangun pada pemrograman berorientasi obyek. Sebuah class digambarkan sebagai sebuah kotak yang terbagi atas 3 bagian. Bagian atas adalah bagian nama dari *class*. Bagian tengah mendefinisikan property/atribut *class*. Bagian akhir mendefinisikan method-method dari sebuah *class*.	Site Config +sqlDNS:string +Adminemail:String
Assosiation	Sebuah asosiasi merupakan sebuah *relationship* paling umum antara 2 class, dan dilambangkan oleh sebuah garis yang menghubungkan antara 2 *class*. Garis ini bisa melambangkan tipe-tipe *relationship* dan juga dapat menampilkan hukum-hukum multiplisitas pada sebuah *relationship* (Contoh: One-to-one, one-to-many, many-to-many).	1..n Owned by 1
Composition	Jika sebuah *class* tidak bisa berdiri sendiri dan harus merupakan bagian dari *class* yang lain, maka *class* tersebut memiliki relasi *Composition* terhadap *class* tempat dia bergantung tersebut. Sebuah *relationship composition* digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi/solid.	

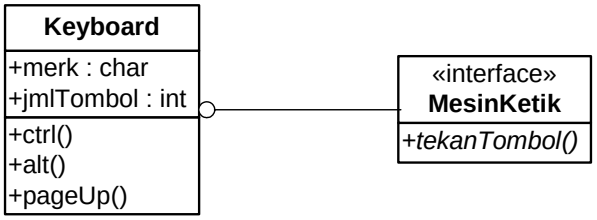
Dependency	Kadangkala sebuah *class* menggunakan *class* yang lain. Hal ini disebut *dependency*. Umumnya penggunaan *dependency* digunakan untuk menunjukkan operasi pada suatu *class* yang menggunakan *class* yang lain. Sebuah *dependency* dilambangkan sebagai sebuah panah bertitik-titik.	
Aggregation	*Aggregation* mengindikasikan keseluruhan bagian *relationship* dan biasanya disebut sebagai relasi "mempunyai sebuah" atau "bagian dari". Sebuah *aggregation* digambarkan sebagai sebuah garis dengan sebuah jajaran genjang yang tidak berisi/tidak solid.	
Generalization	Sebuah relasi *generalization* sepadan dengan sebuah relasi *inheritance* pada konsep berorientasi obyek. Sebuah *generalization* dilambangkan dengan sebuah panah dengan kepala panah yang tidak solid yang mengarah ke kelas "*paren*t"-nya/induknya.	



Gambar 2.1. Contoh *Class Diagram*

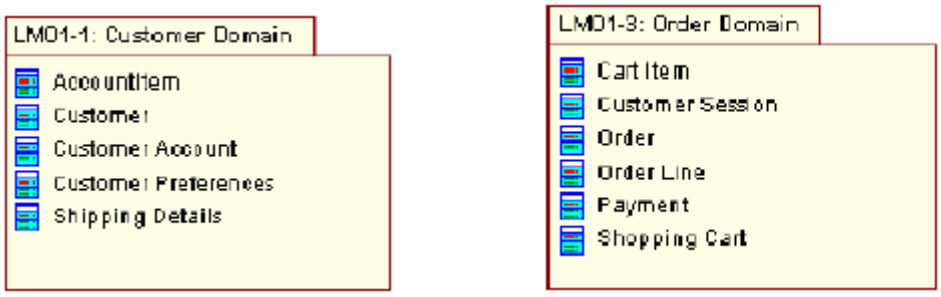
Perhatikan contoh *class diagram* pada gambar 2.1. Diagram ini mempunyai 2 kelas, yaitu kelas burung dan kelas manusia. Masing-masing kelas memiliki atribut dan operasi. Selain itu pada gambar 2.1 juga terdapat konsep inheritansi kelas, yaitu pada kelas binatang, yang inheritansinya diterapkan kepada kucing dan burung.

Class juga dapat merupakan implementasi dari sebuah *interface,* yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Dengan demikian *interface* mendukung resolusi metoda pada saat *runtime*.



Gambar 2.2. Contoh sebuah *Interface*

Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package* (paket).



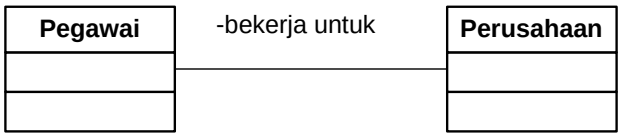
Gambar 2.3. Contoh *Package*

2.1.1.1. Relasi pada Class Diagram

Relasi atau *relationship* merupakan keterhubungan antar kelas yang muncul pada saat sebuah kelas berinteraksi dengan kelas-kelas lainnya. Didalam *class diagram*, setiap kelas pasti akan berinteraksi dengan baik satu maupun lebih kelas. Relasi yang muncul pada setiap keterhubungan antar kelas juga akan memiliki atribut-atribut yang akan lebih menjelaskan definisi dari sebuah relasi yang terjadi.

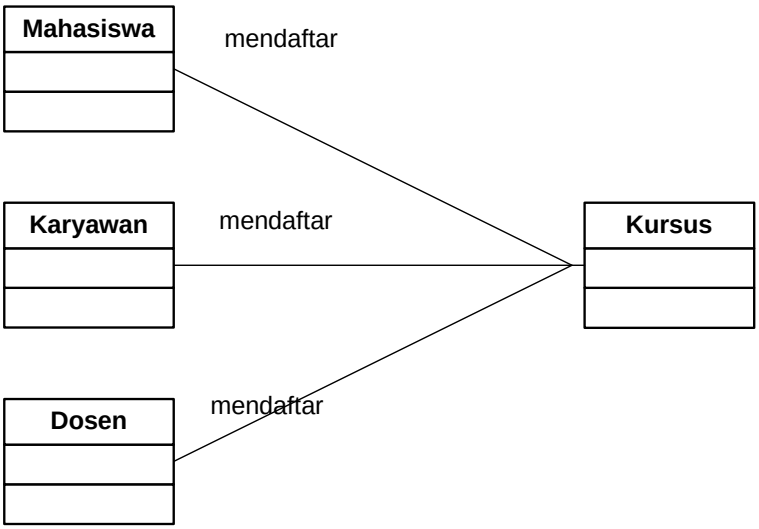
Association (asosiasi).

Kita menggunakan pemahaman asosiasi adalah pada saat beberapa kelas saling terhubung satu sama lain secara konseptual. Sebagai contoh, misalkan seorang pegawai bekerja pada sebuah perusahaan. Maka “bekerja” merupakan sebuah asosiasi antara kelas pegawai dan kelas perusahaan. Contoh yang lain kita misalkan seorang mahasiswa mendaftar sebuah kursus, maka jelas sekali disini bahwa asosiasi yang muncul adalah “mendaftar”. Selanjutnya bisa kita simpulkan bahwa sebuah asosiasi bisa merupakan sebuah bentuk kata kerja yang merelasikan kelas yang satu dengan kelas yang lainnya. Untuk menggambarkan sebuah asosiasi anda dapat kembali merujuk ke tabel 2.1. Gambar 2.4 berikut menunjukkan bagaimana visualisasi sebuah asosiasi.



Gambar 2.4. Contoh sebuah asosiasi

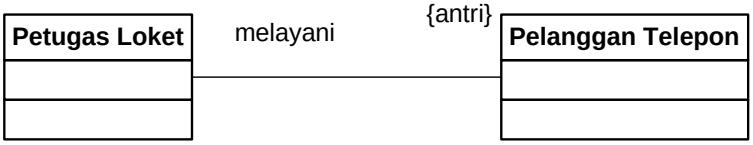
Asosiasi juga dapat menjadi lebih kompleks pada saat beberapa *class* terhubung ke satu *class*, seperti yang terlihat pada gambar 2.5 berikut.



Gambar 2.5. Contoh asosiasi lebih dari 2 *class*

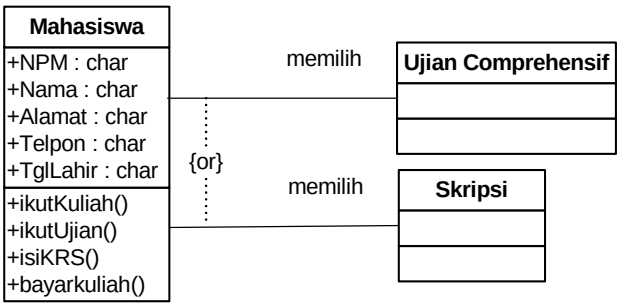
Constraint pada asosiasi.

Kadangkala sebuah asosiasi diantara dua *class* harus mengikuti sebuah aturan dan aturan ini bisa diletakkan dalam sebuah *constraint* pada garis asosiasi dan diletakkan dalam kurung kurawal. Berikut (gambar 2.6) adalah contoh *constraint* dimana petugas loket akan melayani para pelanggan telepon yang ingin melakukan segala urusan yang berhubungan dengan masalah telepon, tapi untuk dapat dilayani maka para pelanggan harus antri, maka “antri” kita jadikan *constraint* pada asosiasi tersebut.



Gambar 2.6. Contoh constraint pada sebuah asosiasi

Bentuk lain dari tipe *constraint* adalah relasi OR yang ditulis dengan {or} dalam garis putus-putus yang menghubungkan 2 garis asosiasi. Kondisi OR ini menghadapkan kepada keadaan bahwa sebuah kelas terhubung dengan dua kelas tetapi pada saat mendefinisikan relasinya kelas harus memilih salah satu dari kedua kelas tersebut, sedangkan kondisi untuk memilih keduanya adalah invalid. Untuk lebih jelasnya anda dapat melihat gambar 2.7 berikut.

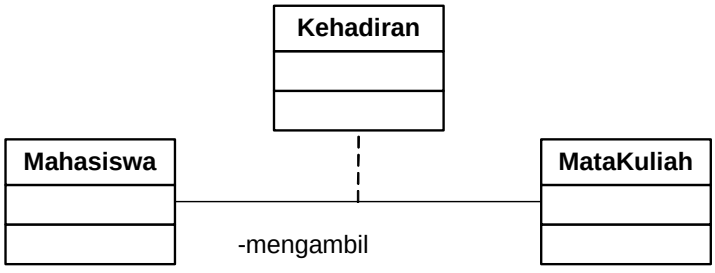


Gambar 2.7. Contoh *constraint* OR pada sebuah asosiasi

Jika kita lihat contoh gambar 2.7, dapat kita nyatakan bahwa seorang mahasiswa yang ingin lulus harus memilih salah satu metode ujian akhir, ujian comprehensif atau skripsi, dan tidak boleh memilih keduanya.

Associations Class.

Sebuah asosiasi dapat memiliki atribut dan operasi seperti halnya sebuah *class*. Sebuah *association class* sebenarnya diperlukan apabila salah satu dari kelas yang terhubung mempunyai sebuah atau beberapa atribut yang tidak layak dimiliki oleh kelas tersebut, karena secara logis atribut tersebut lebih layak dimiliki oleh asosiasi yang menghubungkan kedua kelas tersebut. Akan lebih mudah dipahami jika kita menganalogikan hal ini dengan diagram ERD, dimana sesuai dengan hukum-hukum tertentu maka jika ada sebuah relasi *binary* atau *ternary* maka harus dibuatkan sebuah entitas tambahan yang merupakan entitas transaksi untuk menampung *record-record* transaksi yang terjadi antar entitas murni. Entitas transaksi yang tercipta tersebut mirip sekali dengan *association class*. Berikut adalah contoh sebuah *association class*.



Gambar 2.8. Contoh *Association Class*

Seperti yang dilihat pada gambar 2.8 diatas, *association class* divisualisasikan sama halnya seperti *class* biasa, hanya saja untuk menghubungkan ke garis asosiasi digunakan garis putus-putus.

Multiplicity.

Multiplicity atau multiplisitas adalah jumlah banyaknya obyek sebuah *class* yang berelasi dengan sebuah obyek lain pada *class* lain yang berasosiasi dengan *class* tersebut. Untuk menyatakan multiplisitas anda dapat meletakkannya diatas garis asosiasi berdekatan dengan *class* yang sesuai. Anda dapat melihat contohnya pada gambar 2.9.

Ada banyak multiplisitas yang mungkin untuk dipakai. Tabel berikut menjabarkan multiplisitas yang dapat digunakan.

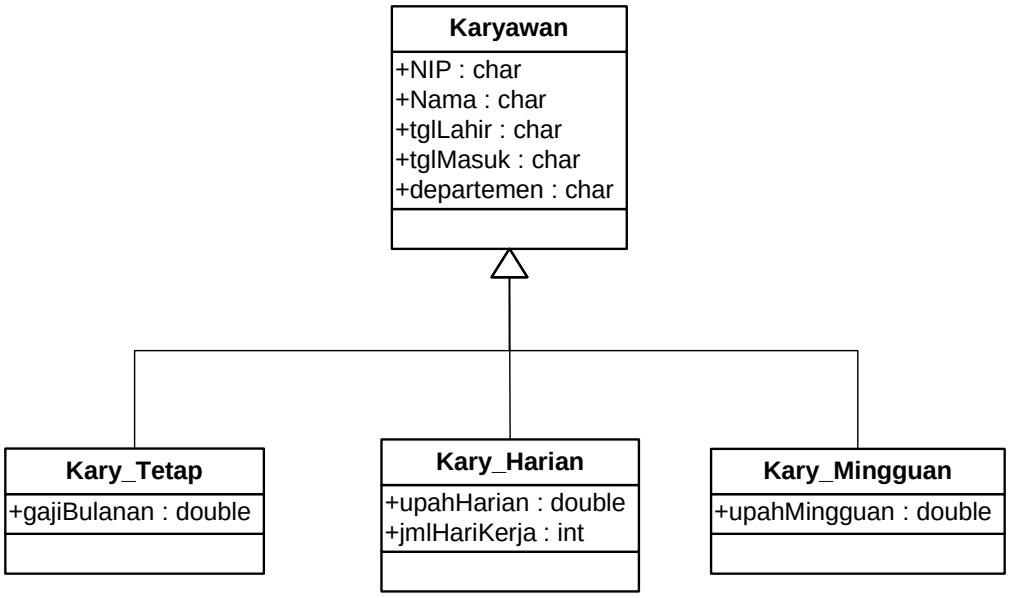
Tabel 2.2. Nilai multiplisitas

Potencial Multiplicity Values	
Indicator	**Meaning**
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
N	Only *n* (where *n* > 1)
0..n	Zero to *n* (where *n* > 1)
1..n	One to *n* (where *n* > 1)

Gambar 2.9. Asosiasi dengan multiplisitas

2.1.1.2. Generalization & Inheritance.

Sebuah *class* (*child class* atau *subclass*) dapat mewarisi atribut-atribut dan operasi-operasi dari *class* lainnya (*parent class* atau *super class*) dimana *parent class* bersifat lebih umum daripada *child class*. Generalisasi pada konsep *Object Oriented* digunakan untuk menjelaskan hubungan kesamaan diantara *class*. Dengan menggunakan generalisasi bisa dibangun struktur logis yang bisa menampilkan derajat kesamaan atau perbedaan diantara *class-class*. Manfaat lain dari struktur hirarkis juga memungkinkan untuk penambahan *subclass* (*child class*) baru tanpa harus merubah struktur yang sudah ada.



Gambar 2.10. Generalisasi dengan 3 *subclass*

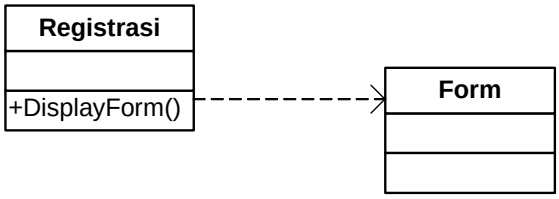
Inheritance adalah sebuah mekanisme pengimplementasian generalisasi dan spesialisasi. Aturan *inheritance* dapat secara umum bisa diklasifikasikan sebagai berikut :

- *Subclass* selalu mewarisi semua sifat dari *superclass*-nya.
- Definisi *subclass* selalu mencakup paling tidak satu detil yang tidak diturunkan dari *superclass*-nya.

Inheritance sangat dekat dengan asosiasinya dengan generalisasi. Generalisasi menjelaskan hubungan logis antar elemen-elemen yang mempunyai karakteristik yang sama. Sedangkan *inheritance* menerangkan mekanisme agar bagi pakai (*sharing*) bisa terjadi.

2.1.1.3. Dependant Class

Pada penggunaan relasi kadangkala satu *class* menggunakan *class* yang lain, hal ini disebut *dependency*. Umumnya penggunaan *dependency* digunakan untuk menunjukkan operasi pada suatu *class* yang menggunakan *class* yang lain. Notasi untuk *dependency* pada UML dapat menggunakan garis putus-putus dan tanda panah pada ujungnya.

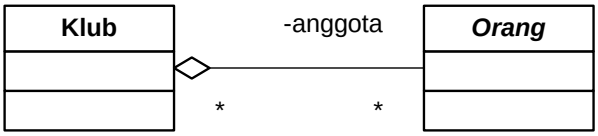


Gambar 2.11. *Dependency*

2.1.1.4. Agregasi

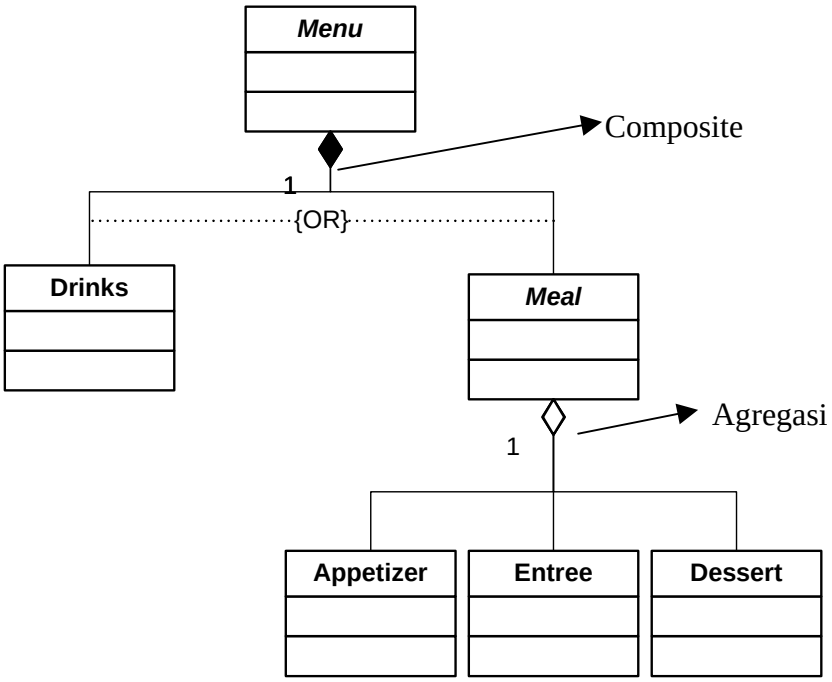
Didalam UML, ada relasi dengan perlakukan khusus yang disebut dengan "bagian dari (*part of*)" yang menangani antar obyek-obyek dimana salah satunya adalah bagian dari yang lain. Dengan kata lain sebuah obyek terdiri atas obyek-obyek yang lain. Hal seperti ini di dalam UML disebut Agregasi.

Sebuah agregasi adalah kasus khusus dari asosiasi. Agregasi disimbolkan dengan jajaran genjang yang diletakkan pada *class* yang mengandung obyek.



Gambar 2.12. Agregasi

Kadangkala relasi OR harus digunakan pada agregasi. Untuk memodelkan hal tersebut, sebuah *constraint* – dengan kalimat atau dalam tanda kurung kurawal pada garis putus-putus harus dibuat untuk menghubungkan 2 obyek. Perhatikan gambar berikut.



Gambar 2.13. Agregasi dengan constraint dan komposit

Pada gambar diatas dapat kita lihat bahwa jika seseorang akan memilih sesuatu di menu maka dia boleh memilih salah satu, yaitu boleh minuman saja atau makanan saja. Hal inilah yang diwakili oleh constraint OR.

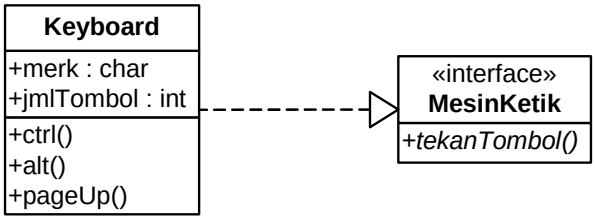
Komposit (*composite*) adalah sebuah tipe agregasi yang kuat dimana bagian dari obyek tergantung penuh/secara keseluruhan terhadap obyeknya sehingga bila sebuah obyek komposit dibuang maka bagian yang tergantung pada komponen tersebut akan terbuang juga pada saat yang bersamaan. Notasi komposit sama seperti agregasi hanya saja jajaran genjangnya terisi (solid) seperti yang bisa anda lihat pada gambar 2.13.

2.1.1.5. Interface & Realisasi.

Interface adalah satu set *operation* yang memberikan spesifikasi beberapa aspek dari perilaku dan operasi disuatu *class* ke *class* yang lain. Contohnya, *keyboard* pada komputer sebenarnya merupakan *interface* yang bisa dipakai ulang karena tombol-tombol *keyboard* sebenarnya berasal dari mesin ketik, hanya saja mungkin ada beberapa operasionalisasi tombol-tombol yang berbeda yang sudah ditransfer ke sistem yang lain. Misalnya tombol *control, page up, page down* dll. Pemodelan *interface* sama dengan pemodelan *class* hanya saja pada *interface*

tidak mempunya atribut dan pada penamaannya perlu ditambahkan *stereotype* <<interface>> atau ditambahkan huruf "I" di depan nama interface seperti yang ditunjukkan pada gambar 2.14.

Relasi antara *class* dan *interface* disebut *Realization*. Realisasi dituliskan dengan garis putus-putus dengan segitiga yang mengarah ke *interface*, seperti yang ditunjukkan pada gambar 2.14.



Gambar 2.14. *Interface* dan *Realization*

2.1.2. Object Diagram.

Object diagram adalah diagram yang memberikan gambaran model *instance-instance* dari sebuah *class*. Diagram ini digunakan untuk menggambarkan sebuah sistem pada sebuah sudut pandang waktu tertentu. Dengan menggunakan diagram ini anda dapat memeriksa keabsahan kelas-kelas diagram berikut aturan-aturan multiplisitasnya dengan "*real data*" dan mengujinya dengan scenario-skenario tertentu. Notasi diagramnya dapat anda lihat pada tabel 2.3.

Tabel 2.3. Notasi Object Diagram

Obyek	Obyek-obyek diidentifikasikan dengan cara meletakkan nama instance-nya kemudian diikuti oleh tanda titik dua didepan nama class-nya. Nilai property/atribut dituliskan berpasangan seperti “nama_atribut=nilai”. Sedangkan notasi sebuah obyek digambarkan segi empat yang terbagi atas 2 bagian.	John Doe : Kary_Harian NIP : char = 50499333 Nama : char = John Doe tglLahir : char = 23-12-70 tglMasuk : char = 05-05-03 departemen : char = Finance upahHarian : double = 120000 jmlHariKerja : int = 25
Association	Object diagram juga dapat mengandung asosiasi. Biasanya constraint, detil relationship, multiplisitas yang ada di class diagram tidak disertakan dalam object diagram sebagai upaya memfokuskan perhatian hanya terhadap obyek dan property/atributnya. Asosiasi antar 2 obyek biasanya dinotasikan dengan sebuah garis yang menghubungkan kedua obyek.	

Berikut adalah contoh sebuah *object diagram* dari relasi antar *class*

Gambar 2.15. *Object Diagram*

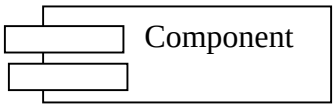
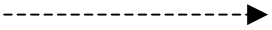
2.1.3. Component Diagram

Komponen perangkat lunak adalah bagian fisik dari sebuah sistem yang menetap di komputer. komponen merupakan implementasi software dari sebuah class. Komponen bisa berupa tabel, *file* data, *file* exe, file DLL, dokumen dan lain-lain.

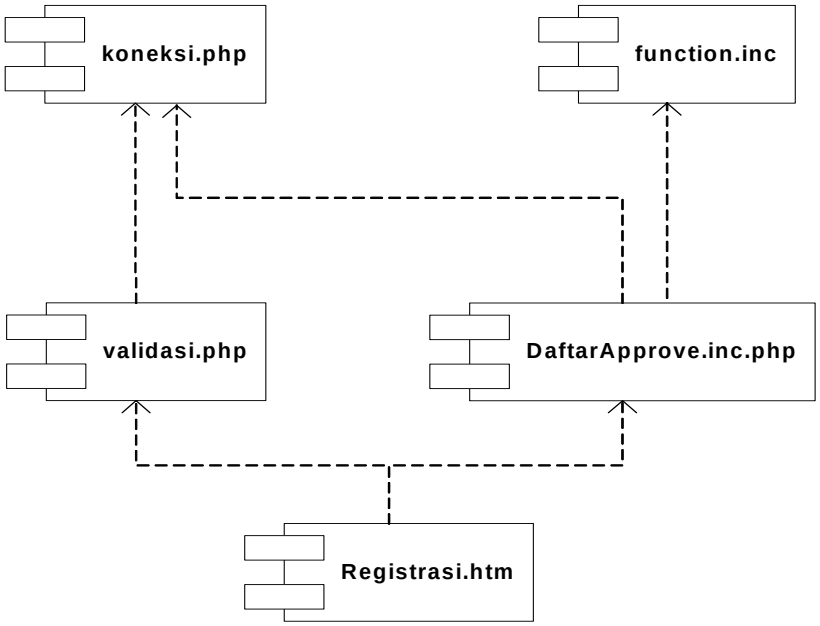
Component diagram mengandung komponen, *interface* dan *relationship*. Komponen diagram ini digunakan pada saat anda ingin memecah sistem menjadi komponen-komponen dan ingin menampilkan hubungan-hubungan mereka dengan antarmuka atau pemecahan komponen menjadi struktur yang lebih rendah. Secara umum dapat kita katakan bahwa component diagram kita gunakan untuk menjelaskan kebergantungan antar beragam komponen-komponen *software* seperti misalnya kebergantungan antara *file-file executable* dengan *file-file* sumbernya (*source file*) dll.

Berikut adalah notasi dari *component diagram* :

Tabel 2.4. Notasi *Component Diagram*

Component	Sebuah komponen melambangkan sebuah entitas software dalam sebuah sistem. Sebuah komponen dinotasikan sebagai sebuah kotak segiempat dengan dua kotak kecil tambahan yang menempel disebelah kirinya.	
Depedency	Sebuah Dependency digunakan untuk menotasikan relasi antara dua komponen. Notasinya adalah tanda panah putus-putus yang diarahkan kepada komponen tempat sebuah komponen itu bergantung.	

Berikut adalah contoh sebuah *component diagram* :

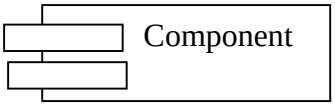
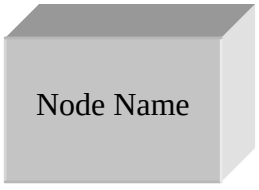


Gambar 2.16. *Component Diagram*

2.1.4. Deployment Diagram

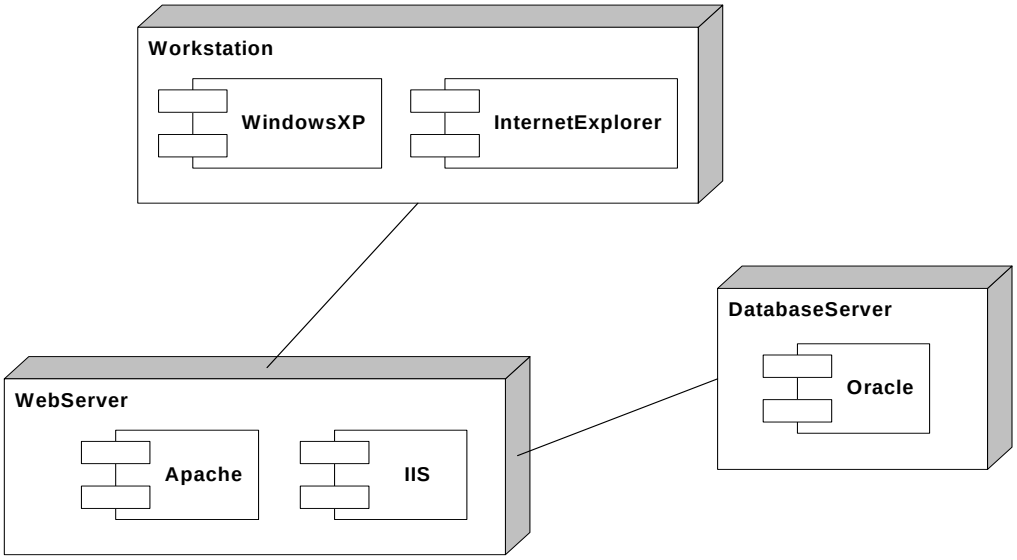
Deployment diagram menunjukkan tata letak sebuah sistem secara fisik, menampakkan bagian-bagian *software* yang berjalan pada bagian-bagian *hardware* yang digunakan untuk mengimplementasikan sebuah sistem dan keterhubungan antara komponen-komponen *hardware* tersebut. *Deployment diagram* dapat digunakan pada bagian-bagian awal proses perancangan sistem untuk mendokumentasikan arsitektur fisik sebuah sistem. Berikut adalah notasi-notasi yang digunakan pada *deployment diagram* :

Tabel 2.5. Notasi *Deployment Diagram*

Component	Pada deployment diagram, komponen-komponen yang ada diletakkan didalam node untuk memastikan keberadaan posisi mereka.	
Node	Node menggambarkan bagian-bagian hardware dalam sebuah sistem. Notasi untuk node digambarkan sebagai sebuah kubus 3 dimensi.	

Association	Sebuah association digambarkan sebagai sebuah garis yang menghubungkan dua node yang mengindikasikan jalur komunikasi antara element-elemen hardware.	———

Berikut adalah contoh sebuah *deployment diagram* :



Gambar 2.17. Contoh *Deployment Diagram*

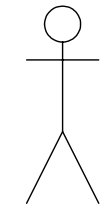
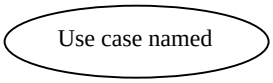
2.1.5. Use Case Diagram

Use case adalah deskripsi fungsi dari sebuah sistem dari perspektif/sudut pandang para pengguna sistem. Use case mendefinisikan *"apa"* yang dilakukan oleh sistem dan elemen-elemennya, bukan *"bagaimana"* sistem dan elemen-elemennya saling berinteraksi. *Use case* bekerja dengan menggunakan "*scenario*", yaitu deskripsi urutan-urutan langkah yang menerangkan apa yang dilakukan penggunan terhadap sistem maupun sebaliknya.

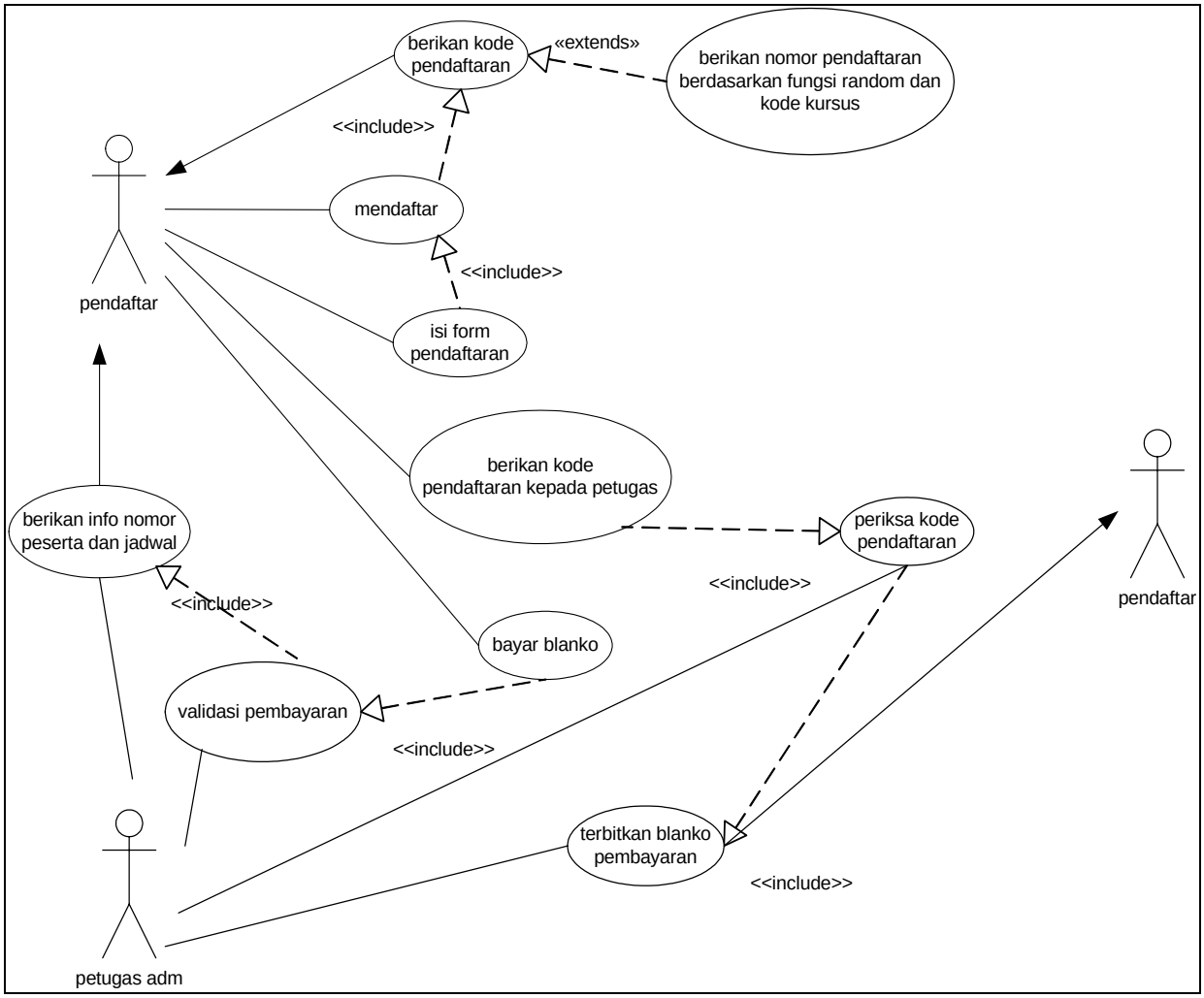
Use case diagram mengidentifikasikan fungsionalitas yang dipunyai oleh sistem (*use-case*), user yang berinteraksi dengan sistem (actor) dan asosiasi/keterhubungan antara *user* dengan fungsionalitas sistem.

Komponen notasi dasar yang dipunyai oleh *use-case diagram* adalah *actor*, *use-case*, dan *association*. Berikut adalah notasi yang terdapat pada use-case diagram :

Tabel 2.6. Notasi *Use Case Diagram*

Actor	Actor adalah pengguna sistem. Actor tidak terbatas hanya manusia saja, jika sebuah sistem berkomunikasi dengan aplikasi lain dan membutuhkan input atau memberikan output, maka aplikasi tersebut juga bisa dianggap sebagai actor.	 Mahasiswa
Use Case	Use case digambarkan sebagai lingkaran elips dengan nama use case dituliskan didalam elips tersebut.	 Use case named
Association	Asosiasi digunakan untuk menghubungkan actor dengan use case. Asosiasi digambarkan dengan sebuah garis yang menghubungkan antara Actor dengan Use Case.	

Berbicara mengenai *use case diagram* tidak akan terlepas dengan hal yang disebut *stereotype*. *Stereotype* adalah sebuah model khusus yang terbatas untuk kondisi tertentu. Untuk menunjukkan *stereotype* digunakan symbol “<<” diawalnya dan ditutup dengan “>>” diakhirnya. Terdapat 2 *stereotype* paling sering digunakan dalam *use case diagram* yaitu <<extend>> dan <<include>>. <<extend>> digunakan untuk menunjukkan bahwa satu *use case* merupakan tambahan fungsional dari *use case* yang lain jika kondisi atau syarat tertentu dipenuhi. Sedangkan <<include>> digunakan untuk menggambarkan bahwa suatu *use case* seluruhnya merupakan fungsionalitas dari *use case* lainnya. Berikut adalah contoh *use case diagram* :



Gambar 2.18. Contoh Use Case Diagram

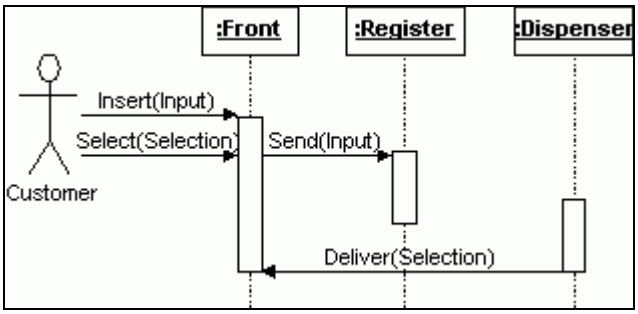
2.1.6. Sequence Diagram

Sequence diagram mendokumentasikan komunikasi/interaksi antar kelas-kelas. Diagram ini menunjukkan sejumlah obyek dan *message* (pesan) – yang diletakkan diantara obyek-obyek didalam *use case*. Perlu diingat bahwa di dalam diagram ini, kelas-kelas dan aktor-aktor diletakkan dibagian atas diagram dengan urutan dari kiri ke kanan dengan garis *lifeline* yang diletakkan secara vertikal terhadap kelas dan aktor. Berikut adalah notasi-notasinya :

Tabel 2.7. Notasi *Sequence Diagram*

Object	Object merupakan instance dari sebuah class dan dituliskan tersusun secara horizontal. Digambarkan sebagai sebuah class (kotak) dengan nama obyek didalamnya yang diawali dengan sebuah titik koma.	: Object1
Actor	Actor juga dapat berkomunikasi dengan object, maka actor juga dapat diurutkan sebagai kolom. Simbol Actor sama dengan simbol pada Actor Use Case Diagram.	
Lifeline	Lifeline mengindikasikan keberadaan sebuah object dalam basis waktu. Notasi untuk Lifeline adalah garis putus-putus vertikal yang ditarik dari sebuah obyek.	
Activation	Activation dinotasikan sebagai sebuah kotak segi empat yang digambar pada sebuah lifeline. Activation mengindikasikan sebuah obyek yang akan melakukan sebuah aksi.	
Massage	Message, digambarkan dengan anak panah horizontal antara Activation.Message mengindikasikan komunikasi antara object-object.	message

Berikut adalah contoh sebuah sequence diagram yang menggambarkan sebuah sistem mesin minuman otomatis :

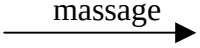


Gambar 2.19. Contoh Sequence Diagram

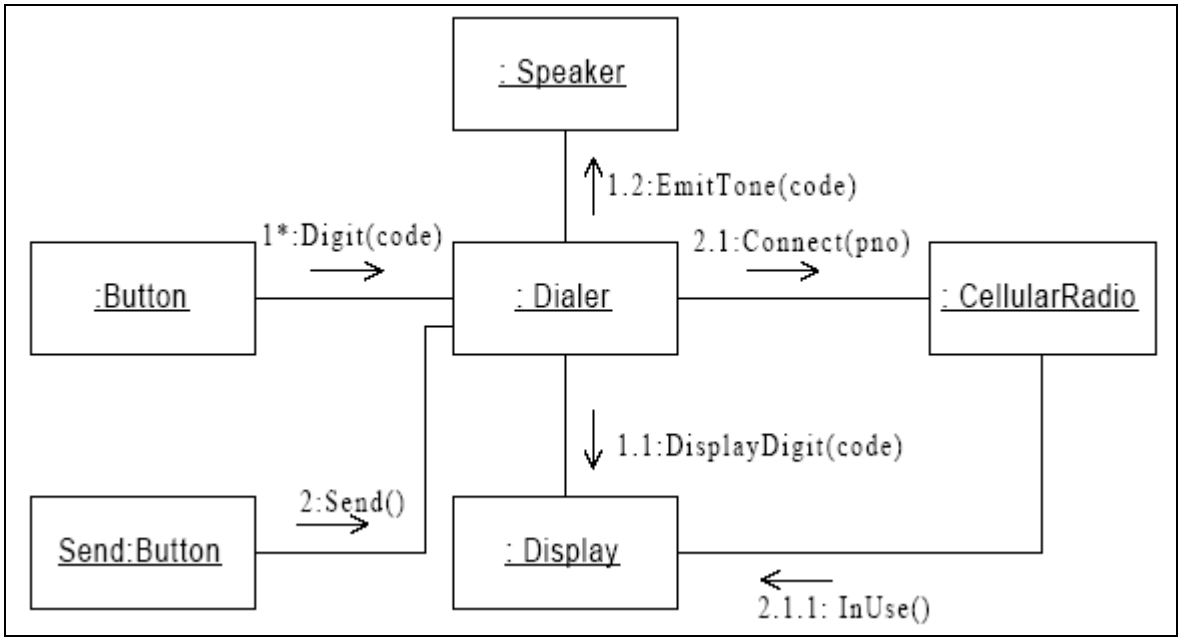
2.1.7. Collaboration Diagram

Collaboration diagram menggunakan prinsip yang sama dengan *sequence diagram*, sama-sama memodelkan interaksi antar obyek-obyek, yang membedakannya hanya cara penggambarannya saja. Pada collaboration diagram ini, obyek-obyek dan *message* (pesan) yang ada digambarkan mirip seperti flowchart, hanya saja, untuk menjaga urutan pesan yang diterima oleh masing-masing obyek, pesan-pesan tersebut diberi nomor urutan pesan. Berikut adalah notasi untuk *collaboration diagram* :

Tabel 2.8. Notasi Collaboration Diagram

Object	Object merupakan instance dari sebuah class. Digambarkan sebagai sebuah class (kotak) dengan nama obyek didalamnya yang diawali dengan sebuah titik koma.	: Object1
Actor	Actor juga dapat berkomunikasi dengan object, maka actor juga dapat disertakan ke dalam collaboration diagram. Simbol Actor sama dengan simbol pada Actor Use Case Diagram.	
Massage	Message, digambarkan dengan anak panah yang mengarah antar obyek dan diberi label urutan nomor yang mengindikasikan urutan komunikasi yang terjadi antar obyek.	massage 

Berikut adalah sebuah contoh *collaboration diagram* yang mengilustrasikan sebuah sistem telepon genggam (*handphone*) :

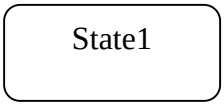
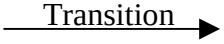


Gambar 2.20. Contoh *Collaboration Diagram*

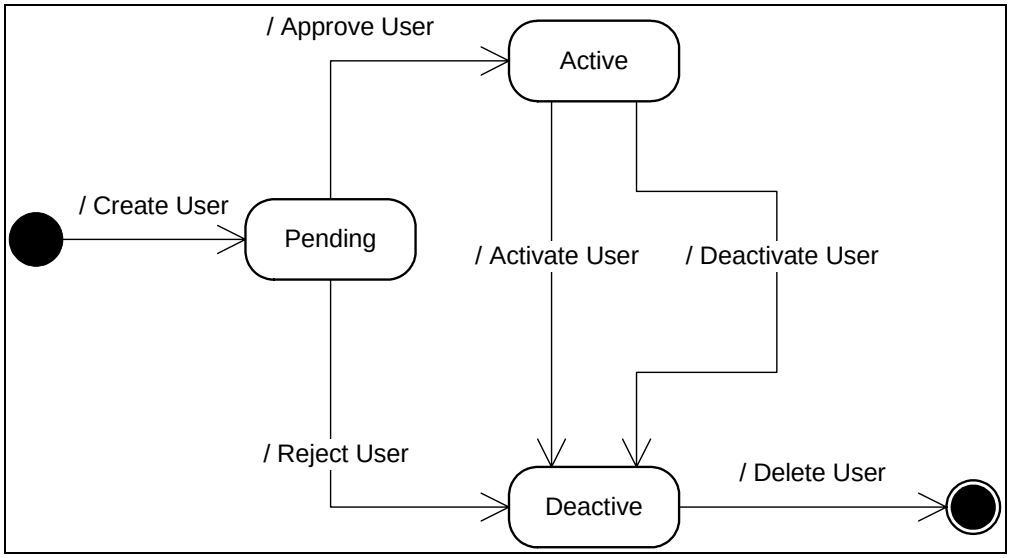
2.1.8. Statechart Diagram.

Statechart diagram, atau yang biasa juga disebut *state diagram* digunakan untuk mendokumentasikan beragam kondisi/keadaan yang bisa terjadi terhadap sebuah *class* dan kegiatan apa saja yang dapat merubah kondisi/keadaan tersebut. Contohnya sebuah televisi yang dapat berada dalam kondisi menyala atau mati, jika tombol “power” ditekan maka televisi akan menyala, begitu juga sebaliknya akan mati jika tombol “power” ditekan kembali. Maka disini kita mempunyai sebuah kelas yaitu televisi, 2 *state* yaitu *menyala* dan *mati* dan 2 *transition* yaitu *menyalakan TV* dan *mematikan TV*. Tidak seperti diagram-diagram *behavioural* lainnya yang memodelkan interaksi diantara beberapa *class*, *state diagram* justru biasanya hanya memodelkan transisi yang terjadi hanya pada sebuah *class*. Berikut adalah notasi *state diagram* :

Tabel 2.9. Notasi *Statechart Diagram*

State	Notasi State menggambarkan kondisi sebuah entitas, dan digambarkan dengan segiempat yang pinggirnya tumpul dengan nama state didalamnya.	
Transition	Sebuah Transition menggambarkan sebuah perubahan kondisi objek yang disebabkan oleh sebuah event. Transition digambarkan dengan sebuah anak panah dengan nama event yang ditulis diatasnya, dibawahnya atau sepanjang anak panah tersebut.	
Initial state	Initial State adalah sebuah kondisi awal sebuah object sebelum ada perubahan keadaan. Initial State digambarkan dengan sebuah lingkaran solid. Hanya satu Initial State yang diizinkan dalam sebuah diagram	
Final State	Final State menggambarkan ketika objek berhenti memberi respon terhadap sebuah event. Final State digambarkan dengan lingkaran solid didalam sebuah lingkaran kosong.	

Berikut adalah contoh sebuah *statechart diagram* yang menggambarkan sebuah *class* pembuatan sebuah *account user* baru pada sebuah sistem *electronic mail* (*e-mail*) :

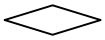


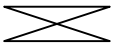
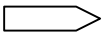
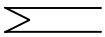
Gambar 2.21. Contoh *Statechart Diagram*

2.1.9. Activity Diagram

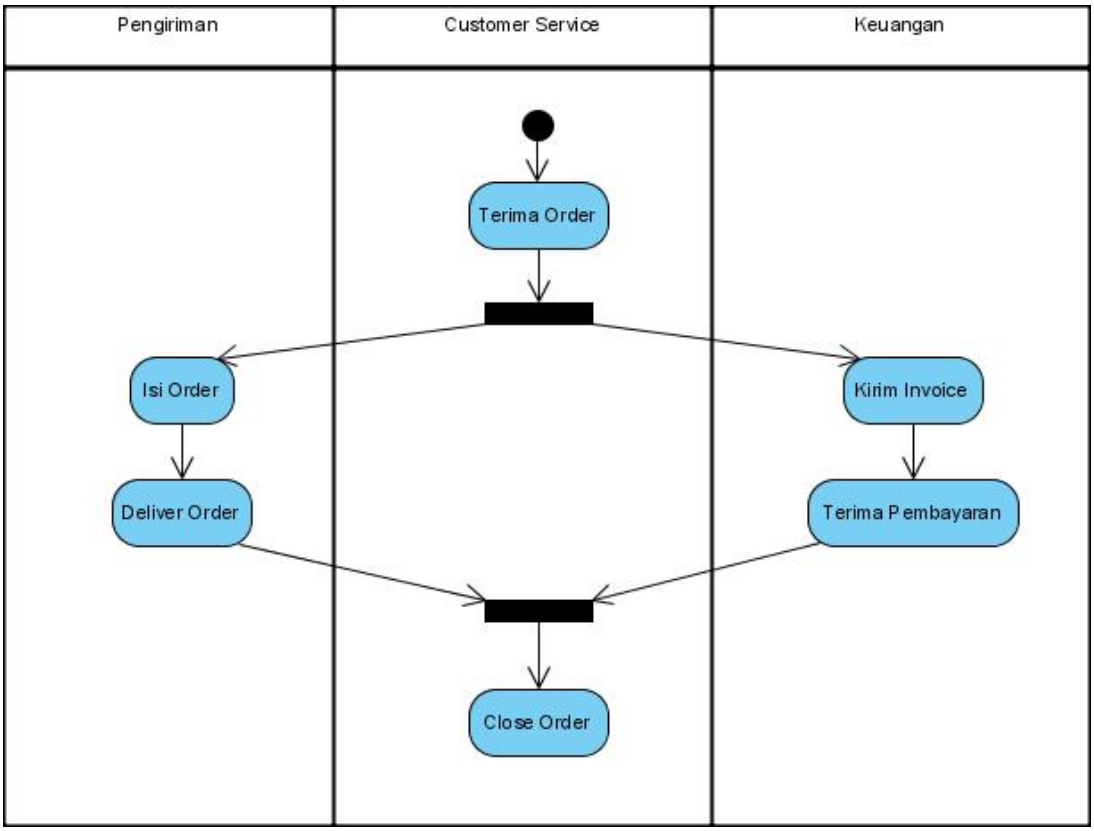
Activity diagram digunakan untuk mendokumentasikan alur kerja pada sebuah sistem, yang dimulai dari pandangan *business level* hingga ke *operational level*. Pada dasarnya, *activity diagram* merupakan variasi dari *statechart diagram*. *Activity diagram* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah *activity diagram* bisa mendukung perilaku parallel sedangkan *flowchart* tidak bisa. Berikut adalah notasi *activity diagram* :

Tabel 2.10. Notasi Activity Diagram

Simbol	**Keterangan**
	Titik Awal
	Titik Akhir
	Activity
	Pilihan Untuk mengambil Keputusan
	Fork; Digunakan untuk menunjukkan kegiatan yang dilakukan secara parallel atau untuk menggabungkan dua kegiatan peralel menjadi satu.

	Rake; Menunjukkan adanya dekomposisi
	Tanda Waktu
	Tanda pengiriman
	Tanda penerimaan
	Aliran akhir (Flow Final)

Berikut adalah sebuah contoh *activity diagram* yang menggambarkan sebuah sistem *Purchasing* :



Gambar 2.22. Contoh *Activity Diagram*