Unified Modeling Language (UML)

Pendahuluan

UML (Unified Modeling Language) adalah sebuah bahasa yang berdasarkan grafik/gambar untuk memvisualisasi, menspesifikasikan, membangun, dan pendokumentasian dari sebuah sistem pengembangan software berbasis OO (Object-Oriented).

UML sendiri juga memberikan standar penulisan sebuah sistem blue print, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam bahasa program yang spesifik, skema database, dan komponen-komponen yang diperlukan dalam sistem software

UML adalah salah satu tool / model untuk merancang pengembangan software yang berbasis object oriented

UML sebagai sebuah bahasa yang memberikan vocabulary dan tatanan penulisan kata-kata dalam 'MS Word' untuk kegunaan komunikasi. Sebuah bahasa model adalah sebuah bahasa yang mempunyai vocabulary dan konsep tatanan / aturan penulisan serta secara fisik mempresentasikan dari sebuah sistem.

UML adalah sebuah bahasa standard untuk pengembangan sebuah software yang dapat menyampaikan bagaimana membuat dan membentuk model-model, tetapi tidak menyampaikan apa dan kapan model yang seharusnya dibuat yang merupakan salah satu proses implementasi pengembangan software.

UML tidak hanya merupakan sebuah bahasa pemograman visual saja, namun juga dapat secara langsung dihubungkan ke berbagai bahasa pemograman, seperti JAVA, C++, Visual Basic, atau bahkan dihubungkan secara langsung ke dalam sebuah object-oriented database. Begitu juga mengenai pendokumentasian dapat dilakukan seperti; requirements, arsitektur, design, source code, project plan, tests, dan prototypes.

Untuk dapat memahami UML membutuhkan bentuk konsep dari sebuah bahasa model, dan mempelajari 3 (tiga) elemen utama dari UML seperti building block, aturan-aturan yang menyatakan bagaimana building block diletakkan secara bersamaan, dan beberapa mekanisme umum (common).

Building blocks

Tiga macam yang terdapat dalam building block adalah :

Benda/Things, adalah abstraksi yang pertama dalam sebuah model

Hubungan/Relationships, sebagai alat komunikasi dari benda-benda,

Bagan/Diagrams. sebagai kumpulan / group dari benda-benda/things.

Benda/Things

Adalah hal yang sangat mendasar dalam model UML, juga merupakan bagian paling statik dari sebuah model, serta menjelaskan elemen-elemen lainnya dari sebuah konsep dan atau fisik.
Bentuk dari beberapa benda/thing adalah sebagai berikut:

- ***Classes,*** yang diuraikan sebagai sekelompok dari object yang mempunyai atribute, operasi, hubungan yang semantik. Sebuah kelas mengimplementasikan 1 atau lebih interfaces. Sebuah kelas dapat digambarkan sebagai sebuah persegi panjang, yang mempunyai sebuah nama, atribute, dan metoda pengoperasiannya.

- ***Interfaces,*** merupakan sebuah antar-muka yang menghubungkan dan melayani antar kelas dan atau elemen. ‘Interface’ / antar-muka mendefinisikan sebuah set / kelompok dari spesifikasi pengoperasian, umumnya digambarkan dengan sebuah lingkaran yang disertai dengan namanya. Sebuah antar-muka berdiri sendiri dan umumnya merupakan pelengkap dari kelas atau komponen.

- ***Collaboration,*** yang didefinisikan dengan interaksi dan sebuah kumpulan / kelompok dari kelas-kelas/elemen-elemen yang bekerja secara bersama-sama. Collaborations mempunyai struktur dan dimensi. Pemberian sebuah kelas memungkinkan berpartisipasi didalam beberapa collaborations dan digambarkan dengan sebuah ‘elips’ dengan garis terpotong-potong.

- ***Use cases,*** adalah rangkaian/uraian sekelompok yang saling terkait dan membentuk sistem secara teratur yang dilakukan atau diawasi oleh sebuah aktor. ‘use case’ digunakan untuk membentuk tingkah-laku benda/ things dalam sebuah model serta di realisasikan oleh sebuah collaboration. Umumnya ‘use case’

digambarkan dengan sebuah ‘elips’ dengan garis yang solid, biasanya mengandung nama.

- ***Nodes,*** merupakan fisik dari elemen-elemen yang ada pada saat dijalankannya sebuah sistem, contohnya adalaha sebuah komputer, umumnya mempunyai sedikitnya memory dan processor. Sekelompok komponen mungkin terletak pada sebuah node dan juga mungkin akan berpindah dari node satu ke node lainnya. Umumnya node ini digambarkan seperti kubus serta hanya mengandung namanya.

Hubungan / Relationship

Ada 4 macam hubungan didalam penggunaan UML, yaitu;

- ***Dependency,*** adalah hubungan semantik antara dua benda/things yang mana sebuah benda berubah mengakibatkan benda satunya akan berubah pula. Umumnya sebuah dependency digambarkan sebuah panah dengan garis terputus-putus.

- **Association,** hubungan antar benda struktural yang terhubung diantara obyek. Kesatuan obyek yang terhubung merupakan hubungan khusus, yang menggambarkan sebuah hubungan struktural diantara seluruh atau sebagian. Umumnya assosiation digambarkan dengan sebuah garis yang dilengkapi dengan sebuah label, nama, dan status hubungannya.

- **Generalizations,** adalah menggambarkan hubungan khusus dalam obyek anak/child yang menggantikan obyek parent / induk . Dalam hal ini, obyek anak memberikan pengaruhnya dalam hal struktur dan tingkah lakunya kepada obyek induk. Digambarkan dengan garis panah.

- **Realizations,** merupakan hubungan semantik antara pengelompokkan yang menjamin adanya ikatan diantaranya. Hubungan ini dapat diwujudkan diantara interface dan kelas atau elements, serta antara use cases dan collaborations. Model dari sebuah hubungan realization.

Bagan/Diagram

UML sendiri terdiri atas pengelompokkan diagram-diagram sistem menurut aspek atau sudut pandang tertentu. Diagram adalah yang menggambarkan permasalahan maupun solusi dari permasalahan suatu model. UML mempunyai 9 diagram, yaitu; use-case, class, object, state, sequence, collaboration, activity, component, dan deployment diagram.

- **Use Case Diagram**, menggambarkan sekelompok use cases dan aktor yang disertai dengan hubungan diantaranya. Diagram use cases ini menjelaskan dan menerangkan kebutuhan / requirement yang diinginkan/ dikehendaki user/pengguna, serta sangat berguna dalam menentukan struktur organisasi dan model dari pada sebuah sistem.

- **Class Diagram,** yang memperlihatkan struktur statis dari kelas actual didalam sistem.

- **Object Diagram,** yang merupakan varian dari kelas diagram yang memperlihatkan lebih detail banyaknya obyek yang mengintantiasi (instances) kelas.

- **State Diagram,** yang memperliatkan semua keadaan (state) yang dapat dimiliki oleh kelas dan event yang dapat merubah keadaan tersebut.
- **Sequence Diagram,** yang memperlihatkan kolaborasi dinamik antara objek-objek dengan suatu urutan pesan (a sequence of message) antar objek tersebut.

- **Collaboration Diagram,** yang memperlihatkan kolaborasi dinamik antar objek tanpa memperhatikan aspek waktu.

- **Activity Diagram,** yang memperlihatkan aliran urutan aktifitas.

- **Component Diagram,** yang memperlihatkan struktur fisik dari source code dalam terminology code components. Komponen berisi informasi tentang logical class dapat berupa komponen source code, komponen biner atau komponen yang dapat dieksekusi.

- **Deployment Diagram,** yang memperlihatkan arsitektur fisik dari hardware dan software pada sistem.

TEORI UML

2.1. Penjelasan UML

Pemecahan masalah utama dari Object Oriented biasanya dengan penggambaran dalam bentuk model. Model abstrak (semu) merupakan gambaran detail dari inti masalah yang ada, umumnya sama seperti refleksi dari problem yang ada pada kenyataan. Beberapa modeling tool yang dipakai adalah bagian dari dasar UML, kependekan dari *United Modeling Language*.

UML terdiri atas beberapa diagram, yaitu :

- ☒ Diagram Use Case
- ☒ Diagram Class
- ☒ Diagram Package
- ☒ Diagram Sequence
- ☒ Diagram Collaboration
- ☒ Diagram StateChart
- ☒ Diagram Activity
- ☒ Diagram Deployment

Semakin kompleks bentukan sistem yang akan dibuat, maka semakin sulit komunikasi antara orang-orang yang saling terkait dalam pembuatan dan pengembangan software yang akan dibuat. Pada masa lalu, UML mempunyai peranan sebagai software blueprint (gambaran) language untuk analisis sistem, designer, dan programmer. Sedangkan pada saat ini, merupakan bagian dari software trade (bisnis software). UML memberikan jalur komunikasi dari sistem analis kemudian designer, lalu programmer mengenai rancangan software yang akan dikerjakan.

Salah satu pemecahan masalah Object Oriented adalah dengan menggunakan UML. Oleh karena itu orang-orang yang berminat dalam mempelajari UML harus mengetahui dasar-dasar mengenai Object Oriented Solving (pemecahan masalah OO). Tahap pertama, pembentukan model. ***Model*** adalah gambaran abstrak dari suatu dasar masalah. Dan dunia nyata atau tempat dimana masalah itu timbul bisa disebut dengan ***domain.*** Model mengandung obyek-obyek yang beraktifitas dengan saling mengirimkan messages (pesan-pesan). Obyek mempunyai sesuatu yang diketahui

(atribut /attributes) dan sesuatu yang dilakukan (behaviors atau operations). Attributes hanya berlaku dalam ruang lingkup obyek itu sendiri (state). Lalu “blue print” dari suatu obyek adalah Classes (kelas). Obyek merupakan bagian-bagian dari kelas.

2.1.1. Diagram Use Case

Diagram Use Case menggambarkan apa saja aktifitas yang dilakukan oleh suatu sistem dari sudut pandang pengamatan luar. yang menjadi persoalan itu *apa yang dilakukan* bukan *bagaimana melakukannya*.

Diagram Use Case dekat kaitannya dengan kejadian-kejadian. kejadian (scenario) merupakan contoh apa yang terjadi ketika seseorang berinteraksi dengan sistem. untuk lebih memperjelas lihat gambaran suatu peristiwa untuk sebuah klinik kesehatan di bawah ini :

> “Pasien menghubungi klinik untuk membuat janji (appointment) dalam pemeriksaan tahunan. Receptionist mendapatkan waktu yang luang pada buku jadwal dan memasukkan janji tersebut ke dalam waktu luang itu.”

contoh kegiatan pasien yang membuat janji.

Gambar 2. 1

Diagram Use Case berguna dalam tiga hal :

- ☒ Menjelaskan fasilitas yang ada (requirements) Use Case baru selalu menghasilkan fasilitas baru ketika sistem di analisa, dan design menjadi lebih jelas.
- ☒ Komunikas dengan klien

  Penggunaan notasi dan simbol dalam diagram Use Case membuat pengembang lebih mudah berkomunikasi dengan klien-kliennya.
- ☒ Membuat test dari kasus-kasus secara umum

  Kumpulan dari kejadian-kejadian untuk Use Case bisa dilakukan test kasus layak untuk kejadian-kejadian tersebut.

2.1.2. Diagram Class

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*.

Diagram Class mempunyai 3 macam relationalships (hubungan), sebagai berikut :

- 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* (kelas super) dari kelas yang lain. *Generalization* memiliki tingkatan yang berpusat pada *superClass*. Contoh : Payment adalah *superClass* dari Cash, Check, dan Credit.

Untuk tambahan bahwa *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).

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’.

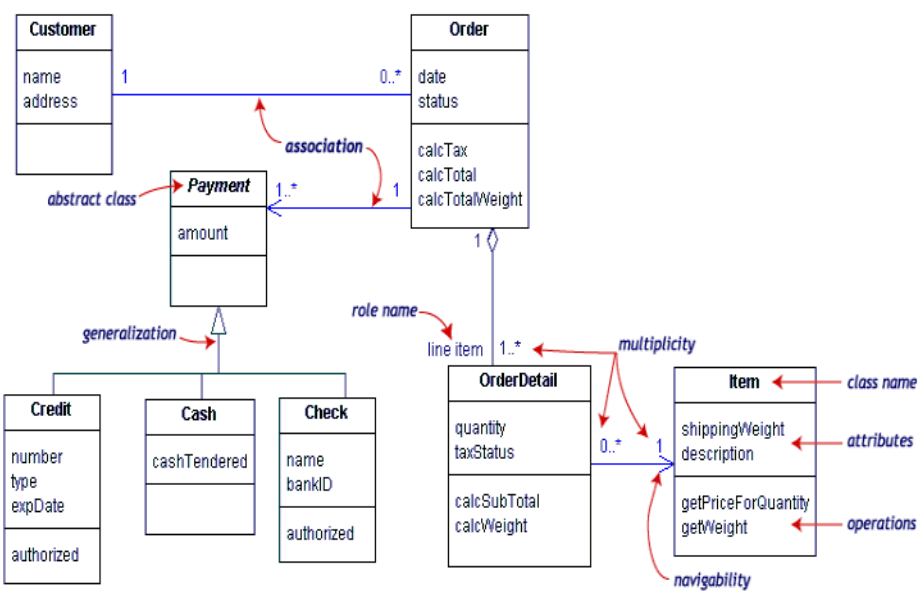
Tabel di bawah mengenai *multiplicity* yang sering digunakan :

Tabel 2. 1

Tabel *Multiplicity*.

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

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



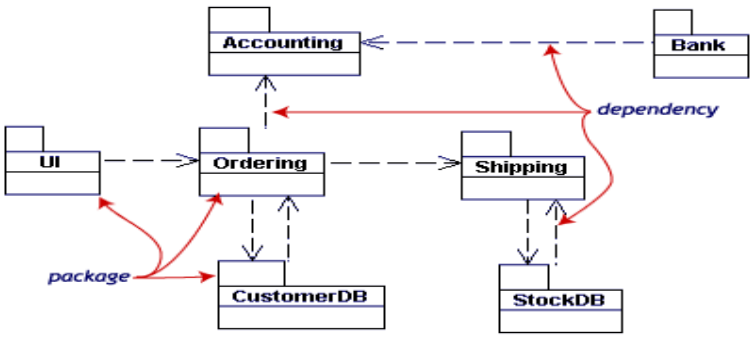
Contoh Diagram Class transaksi Pembelian barang.

Gambar 2. 2

2.1.3. Package dan Object

Untuk mengatur pengorganisasian diagram Class yang *kompleks*, dapat dilakukan pengelompokan kelas-kelas berupa *package* (paket-paket). *Package* adalah

kumpulan elemen-elemen logika UML. Gambar di bawah ini mengenai model bisnis dengan pengelompokan kelas-kelas dalam bentuk paket-paket :

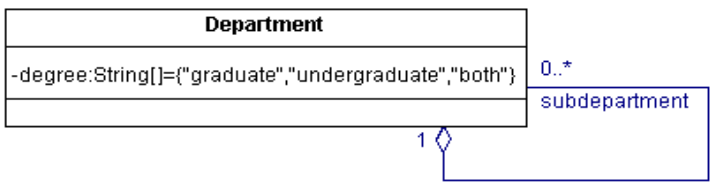


Contoh Diagram Package.

Gambar 2. 3

Ada jenis khusus dari diagram Class yaitu diagram Object. Kegunaannya untuk penjelasan yang sedikit dengan relasi yang sulit, khususnya relasi rekursif.

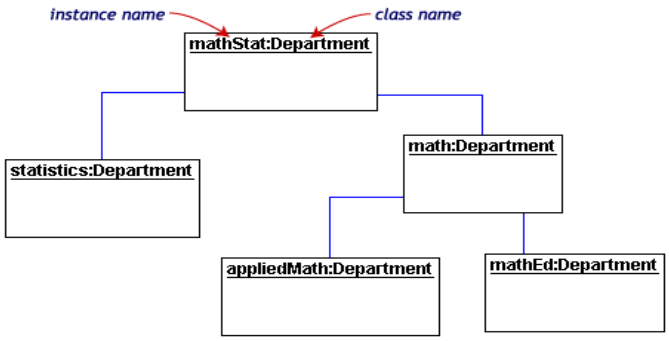
Lihat gambar dibawah, diagram Class kecil menunjukkan bahwa ‘department’ dapat mengandung banyak ‘department’ yang lain.



Class yang relasinya rekursif.

Gambar 2. 4

Setiap tingkatan pada diagram berpengaruh pada *single instance* (bagian tunggal). Nama bagian digarisbawahi dalam diagram UML. Untuk *Class name* (nama kelas) maupun *instance name* (nama bagian) bisa mengambil dari diagram Object selama arti diagram tersebut masih jelas.



Instance name memiliki huruf yang digarisbawahi.

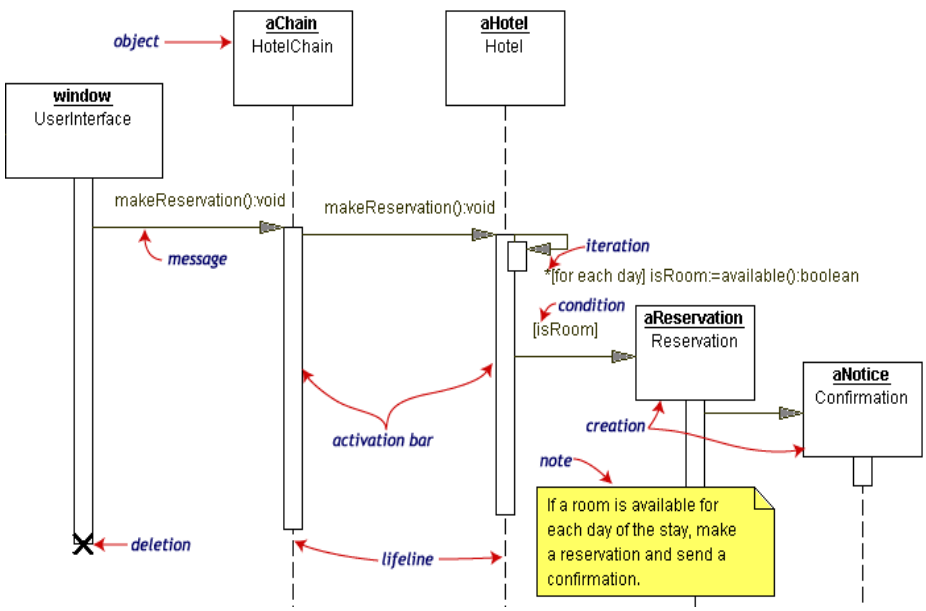
Gambar 2. 5

2.1.4. Diagram Sequence

Diagram Class dan diagram Object merupakan suatu gambaran *model statis*. Namun ada juga yang bersifat *dinamis*, seperti **Diagram Interaction**.

Diagram sequence merupakan salah satu diagram Interaction yang menjelaskan bagaimana suatu operasi itu dilakukan; *message* (pesan) apa yang dikirim dan kapan pelaksanaannya. Diagram ini diatur berdasarkan waktu. Obyek-obyek yang berkaitan dengan proses berjalannya operasi diurutkan dari kiri ke kanan berdasarkan waktu terjadinya dalam pesan yang terurut.

Di bawah ini adalah diagram Sequence untuk pembuatan Hotel Reservation. Obyek yang mengawali urutan *message* adalah ‘aReservation Window’.



Contoh Diagram Sequence ‘Pemesanan kamar di Hotel’.

Gambar 2. 6

'Reservation window' mengirim pesan `makeReservation()` ke 'HotelChain'. Kemudian 'HotelChain' mengirim pesan yang sama ke 'Hotel'. Bila 'Hotel' punya kamar kosong, maka dibuat 'Reservation' dan 'Confirmation'.

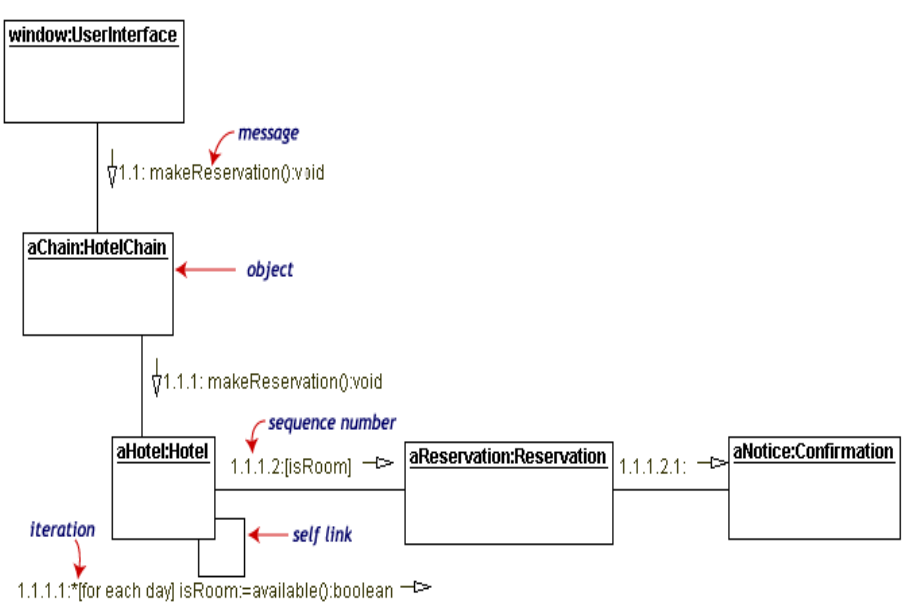
Lifeline adalah garis dot (putus-putus) vertikal pada gambar, menerangkan waktu terjadinya suatu obyek. Setiap panah yang ada adalah pemanggilan suatu pesan. Panah berasal dari pengirim ke bagian paling atas dari batang kegiatan (*activation bar*) dari suatu pesan pada *lifeline* penerima. *Activation bar* menerangkan lamanya suatu pesan diproses.

Pada gambar diagram , terlihat bahwa 'Hotel' telah melakukan pemanggilan diri sendiri untuk pemeriksaan jika ada kamar kosong. Bila benar, maka 'Hotel' membuat 'Reservation' dan 'Confirmation'. Pemanggilan diri sendiri disebut dengan *iterasi*. *Expression* yeng dikurung dengan "[ ]", adalah *condition* (keadaan kondisi).

Pada diagram dapat dibuat *note* (catatan). Pada gambar, terlihat seperti selembar kertas yang berisikan teks. *Note* bisa diletakan dimana saja pada diagram UML.

2.1.5. Diagram Collaboration

Diagram Collaboration juga merupakan *diagram interaction*. Diagram membawa informasi yang sama dengan diagram Sequence, tetapi lebih memusatkan atau memfokuskan pada kegiatan obyek dari waktu pesan itu dikirimkan.



Contoh Diagram Collaboration 'Pemesanan kamar di Hotel'.

Gambar 2. 7

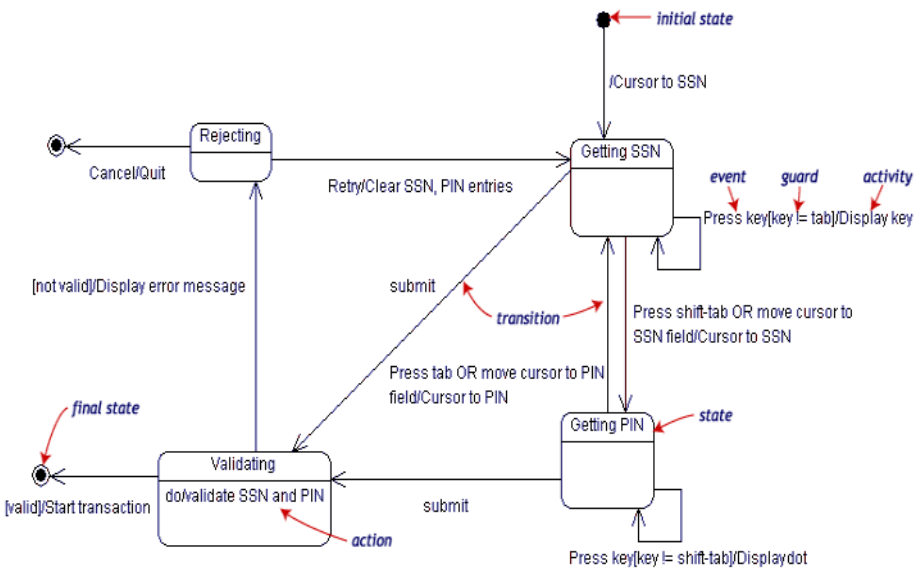
Kotak kegiatan obyek diberi label dengan nama kelas atau obyek (atau keduanya). Nama kelas dibatasi dengan *colons /titik dua* (:).

Setiap pesan pada diagram Collaboration mempunyai angka yang terurut. Pesan yang tingkatannya tertinggi adalah angka 1. Pesan yang berada pada tingkat yang sama memiliki *prefix* yang sama, namun *suffix* berbeda bergantung pada posisinya; hanya untuk angka 1, 2, dan seterusnya.

2.1.6. Diagram StateChart

Behaviors dan *state* dimiliki oleh obyek. Keadaan dari suatu obyek bergantung pada kegiatan dan keadaan yang berlaku pada saat itu. Diagram StateChart menunjukan kemungkinan dari keadaan obyek dan proses yang menyebabkan perubahan pada keadaannya.

Untuk lebih jelas, contoh yang digunakan model diagram untuk login yang merupakan bagian dari Online Banking System. Logging in terdiri atas masukan input Social Security Number dan Personal Id Number yang berlaku, lalu memutuskan kesahan dari informasi tersebut.



Contoh Diagram StateChart ‘Sistem Perbankkan secara Online’.

Gambar 2. 8

Logging in dapat dibagi menjadi empat tahapan proses, yaitu :

- ☒ Getting SSN (masukkan SSN),
- ☒ Getting PIN (masukkan PIN),
- ☒ Validating (periksa kesahannya), dan
- ☒ Rejecting (keluar).

Proses peralihan digambarkan dengan panah dari satu state ke yang lainnya. *Event* (peristiwa) atau *condition* (keadaan) yang menyebabkan perubahan dituliskan pada samping panah. Diagram ini mengandung dua *self-transition* (transisi sendiri), satu pada getting SSN dan lainnya pada getting PIN.

Keadaan awal *Start* (*black circle* /lingkar hitam) adalah *dummy* (model) untuk memulai action (kegiatan). Keadaan akhir juga keadaan model yang menghentikan kegiatan.

Aksi yang terjadi sebagai hasil dari suatu peristiwa atau keadaan ditandai dalam bentuk `/action`. Pada Validating State, obyek tidak menunggu peristiwa dari luar untuk menyebabkan suatu perubahan. Sebagai gantinya melakukan suatu *activity* (aktifitas). Hasil dari aktifitas tersebut menentukan keadaan berikutnya dari obyek tersebut.

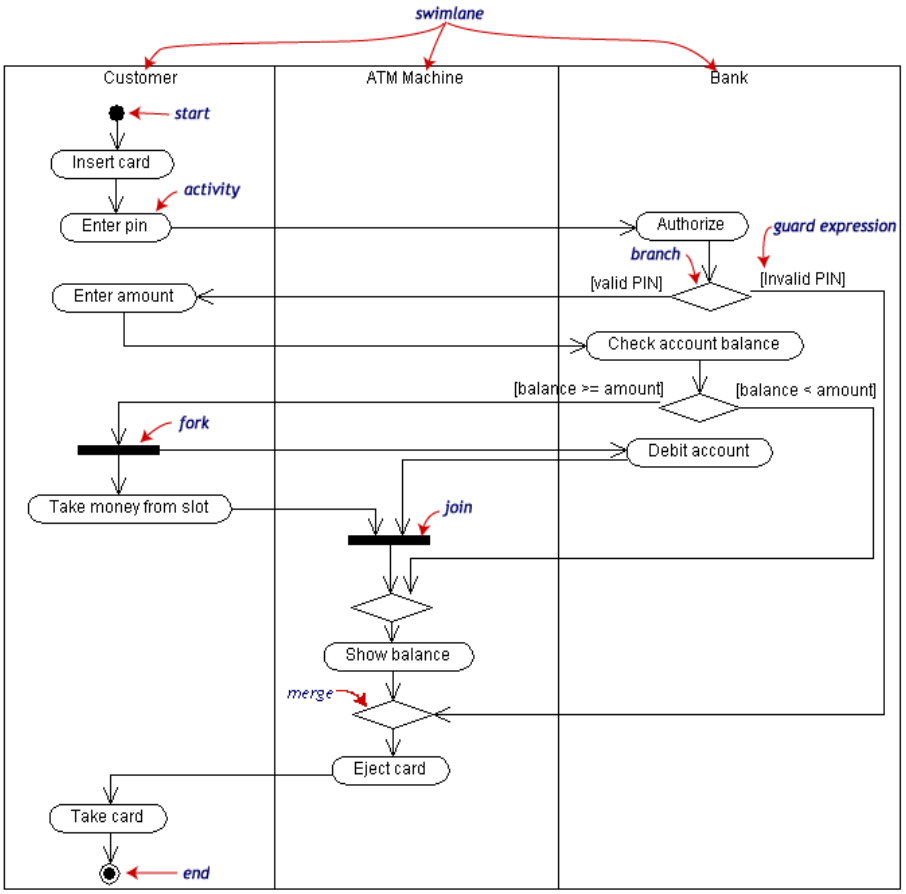
2.1.7. Diagram Activity

Pada dasarnya diagram Activity sering digunakan oleh *flowchart*. Diagram ini berhubungan dengan diagram Statechart. Diagram Statechart berfokus pada *obyek yang dalam suatu proses* (atau proses menjadi suatu obyek), diagram Activity berfokus pada *aktifitas-aktifitas yang terjadi yang terkait dalam suatu proses tunggal*. Jadi dengan kata lain, diagram ini menunjukkan bagaimana aktifitas-aktifitas tersebut bergantung satu sama lain.

Sebagai contoh, perhatikan proses yang terjadi.

“Pengambilan uang dari bank melalui ATM.”

Ada tiga aktifitas kelas (orang, dan lainnya) yang terkait yaitu : Customer, ATM, and Bank. Proses berawal dari lingkaran start hitam pada bagian atas dan berakhir di pusat lingkaran stop hitam/putih pada bagian bawah. Aktivitas digambarkan dalam bentuk kotak persegi. Lihat gambar di bawah ini, agar lebih jelas :



Contoh Diagram Activity 'Pengambilan Uang melalui ATM'.

Gambar 2. 9

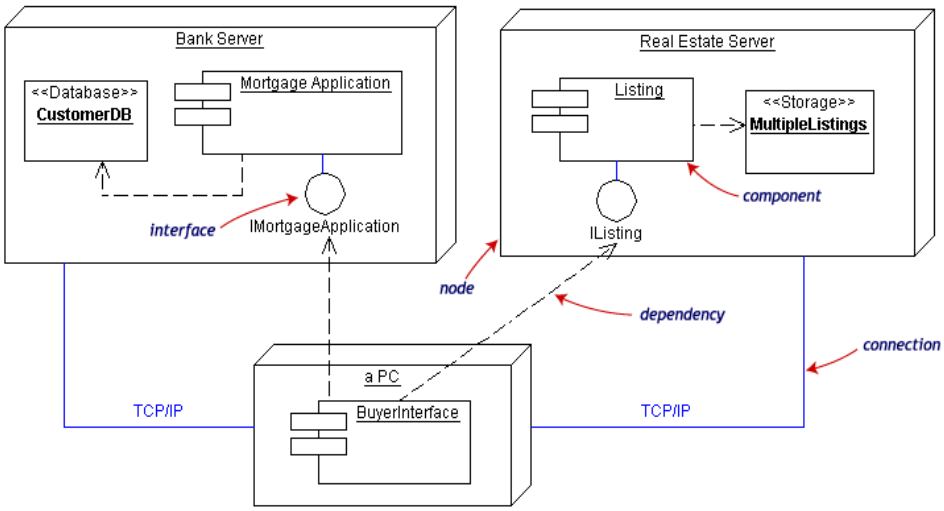
Diagram Activity dapat dibagi menjadi beberapa jalur kelompok yang menunjukkan obyek yang mana yang bertanggung jawab untuk suatu aktifitas. Peralihan tunggal (*single transition*) timbul dari setiap adanya *activity* (aktifitas), yang saling menghubungi pada aktifitas berikutnya.

Sebuah *transition* (transisi) dapat membuat cabang ke dua atau lebih percabangan *exclusive transition* (transisi eksklusif). Label *Guard Expression* (ada di dalam [ ]) yang menerangkan output (keluaran) dari percabangan. percabangan akan menghasilkan bentuk menyerupai bentuk intan. *transition* bisa bercabang menjadi beberapa aktifitas paralel yang disebut **Fork.** *Fork* beserta *join* (gabungan dari hasil output *fork*) dalam diagram berbentuk *solid bar* (batang penuh).

2.1.8. Diagram Component dan Deployment

Component adalah sebuah *code module* (kode-kode module). Diagram Component merupakan *fisik sebenarnya dari diagram Class*. Diagram Deployment menerangkan bahwa *konfigurasi fisik software* dan *hardware*.

Gambar 2.10 menerangkan hubungan sekitar komponen software dan hardware yang berperan dalam ruang lingkup real estate.



Contoh Diagram Deployment 'Sistem Real Estate'.

Gambar 2. 10

Fisik hardware berbentuk seperti *node-node*. Setiap komponen merupakan bagian dari *node*. Pada gambar komponen berbentuk dua kotak tersusun yang terletak di sebelah kiri atas.