MODUL I

DASAR UNIFIED MODELING LANGUAGE (UML)

Tujuan:

- Mahasiswa mengetahui konsep Objec Oriented
- Mengenalkan konsep dan bagian-bagian UML

Teori

UML adalah bahasa untuk menspesifikasi, memvisualisasi, membangun dan mendokumentasikan *artifacts* (bagian dari informasi yang digunakan atau dihasilkan oleh proses pembuatan perangkat lunak, *artifact* tersebut dapat berupa model, deskripsi atau perangkat lunak) dari sistem perangkat lunak, seperti pada pemodelan bisnis dan sistem non perangkat lunak lainnya [HAN98]. Selain itu UML adalah bahasa pemodelan yang menggunakan konsep orientasi *object*. UML dibuat oleh *Grady Booch* , *James Rumbaugh* , dan *Ivar Jacobson* di bawah bendera *Rational Software Corp* [HAN98]. UML menyediakan notasi-notasi yang membantu memodelkan sistem dari berbagai perspektif. UML tidak hanya digunakan dalam pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan..

1. Bagian-bagian Dari UML

Bagian-bagian utama dari UML adalah *view, diagram, model element, dan general mechanism.*

1. View

View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. *View* bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis *view* dalam UML antara lain: *use case view, logical view, component view, concurrency view,* dan *deployment view.*

- **Use case view**

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai yang diinginkan *external actors*. *Actor* yang berinteraksi dengan sistem dapat berupa user atau sistem lainnya. *View* ini digambarkan dalam *use case diagrams* dan kadang-kadang dengan *activity diagrams*. *View* ini digunakan terutama untuk pelanggan, perancang (*designer*), pengembang (*developer*), dan penguji sistem (*tester*).

- **Logical view**

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (*class, object,*

dan *relationship*) dan kolaborasi dinamis yang terjadi ketika *object* mengirim pesan ke *object* lain dalam suatu fungsi tertentu. *View* ini digambarkan dalam *class diagrams* untuk struktur statis dan dalam *state, sequence, collaboration, dan activity diagram* untuk model dinamisnya. *View* ini digunakan untuk perancang (designer) dan pengembang (developer).

- **Component view**

Mendeskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari *code module* diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi administrative lainnya. *View* ini digambarkan dalam *component view* dan digunakan untuk pengembang (developer).

- **Concurrency view**

Membagi sistem ke dalam proses dan prosesor. *View* ini digambarkan dalam diagram dinamis (*state, sequence, collaboration*, dan *activity diagrams*) dan diagram implementasi (*component* dan *deployment diagrams*) serta digunakan untuk pengembang (developer), pengintegrasi (integrator), dan penguji (tester).

- **Deployment view**

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat (*nodes*) dan bagaimana hubungannya dengan lainnya. *View* ini digambarkan dalam *deployment diagrams* dan digunakan untuk pengembang (developer), pengintegrasi (integrator), dan penguji (tester).

2. Diagram

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya dialokasikan untuk view tertentu. Adapun jenis diagram antara lain :

- **Use Case Diagram**

Menggambarkan sejumlah *external actors* dan hubungannya ke *use case* yang diberikan oleh sistem. *Use case* adalah deskripsi fungsi yang disediakan oleh sistem dalam bentuk teks sebagai dokumentasi dari *use case symbol* namun dapat juga dilakukan dalam *activity diagrams*. *Use case* digambarkan hanya yang dilihat dari luar oleh *actor* (keadaan lingkungan sistem yang dilihat user) dan bukan bagaimana fungsi yang ada di dalam sistem.

- **Class Diagram**

Menggambarkan struktur statis *class* di dalam sistem. *Clas*s merepresentasikan sesuatu yang ditangani oleh sistem. *Class* dapat berhubungan dengan yang lain melalui berbagai cara: *associated* (terhubung satu sama lain), *dependent* (satu *class* tergantung/menggunakan *class* yang lain), *specialed* (satu *class* merupakan spesialisasi dari *class* lainnya), atau *package* (grup bersama sebagai satu unit). Sebuah sistem biasanya mempunyai beberapa *class diagram*.

- **State Diagram**

Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu *object* dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa *object* lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh st*ate* yang berbeda.

- **Sequence Diagram**

Menggambarkan kolaborasi dinamis antara sejumlah *object*. Kegunaanya untuk menunjukkan rangkaian pesan yang dikirim antara *object* juga interaksi antara *object*, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

- **Collaboration Diagram**

Menggambarkan kolaborasi dinamis seperti *sequence diagrams*. Dalam menunjukkan pertukaran pesan, *collaboration diagrams* menggambarkan *object* dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence diagrams,* tapi jika penekanannya pada konteks gunakan *collaboration diagram*.

- **Activity Diagram**

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi.

- **Component Diagram**

Menggambarkan struktur fisik kode dari komponent. Komponent dapat berupa *source code*, komponent biner, atau *executable component*. Sebuah komponent berisi informasi tentang logic class atau class yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view*.

- **Deployment Diagram**

Menggambarkan arsitektur fisik dari perangkat keras dan perangkat lunak sistem, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis

hubungannya. Di dalam *nodes*, *executeable component* dan *object* yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh *node* tertentu dan ketergantungan komponen.

Area Penggunaan UML

UML digunakan paling efektif pada domain seperti :

- Sistem Informasi Perusahaan
- Sistem Perbankan dan Perekonomian
- Bidang Telekomunikasi
- Bidang Transportasi
- Bidang Penerbangan
- Bidang Perdagangan
- Bidang Pelayanan Elekronik
- Bidang Pengetahuan
- Bidang Pelayanan Berbasis Web Terdistribusi

Namun UML tidak terbatas untuk pemodelan software. Pada faktanya UML banyak untuk memodelkan sistem non software seperti:

- Aliran kerja pada sistem perundangan.
- Struktur dan kelakuan dari Sistem Kepedulian Kesehatan Pasien
- Desain hardware dll.

Tujuan Penggunaan UML

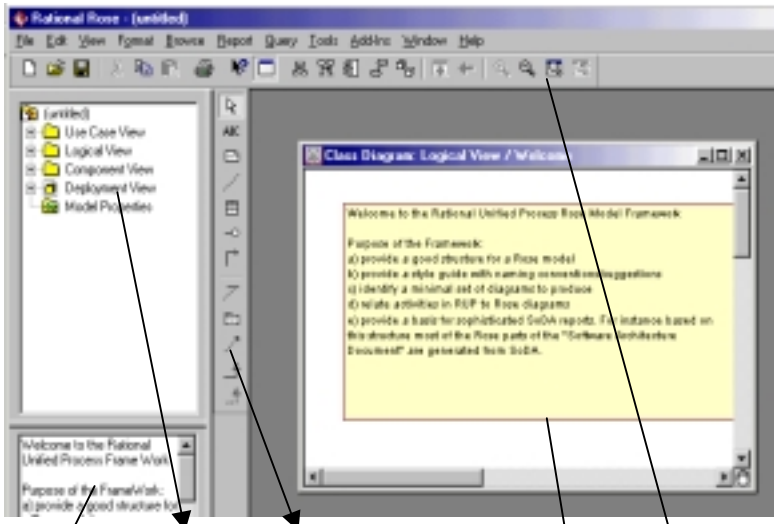
1. Memodelkan suatu sistem (bukan hanya perangkat lunak) yang menggunakan konsep berorientasi object.
2. Menciptakan suatu bahasa pemodelan yang dapat digunakan baik oleh manusia maupun mesin.

Praktek

1. Cara menjalankan Rational Rose:
   - Pilih start → program → Rational Rose 2000 Enterprise Edition

2. Bekerja dalam framework Rational Unified Process

Dalam sub menu New, pilih ikon rational unified process dan tekan OK, maka muncul tampilan seperti dibawah ini:



Bagian-bagian inilah yang akan kita gunakan dalam membuat pemodelan UML

Tugas:

1. Buatlah contoh keadaan nyata dengan menggunakan pendekatan bagian-bagian dari UML view
2. Dalam permasalahan system yang komplek, apakah pemodelan UML dapat digunakan dengan baik. Jelaskan jawaban anda

MODUL II
USE CASE DIAGRAM

Tujuan:

- Mahasiswa mampu membuat sebuah skenario sistem yang nantinya dapat diimplementasikan menjadi sebuah perangkat lunak.
- Mahasiswa bisa memahami alur dari setiap tahap yang digunakan dalam perancangan perangkat lunak menggunakan UML.
- Praktikan dapat memahami hubungan atara *actor* dengan *use case diagram*.
- Praktikan mampu membuat *use case diagram* dari skenario yang telah ada.

Teori

Use-case merupakan gambaran fungsionalitas dari suatu system, sehingga custumer atau pengguna system mengerti kegunaan system yang akan dibangun.

Use case diagram adalah penggambaran system dari sudut user, sehingga pembuatan use case lebih dititik beratkan pada fungsionalitas yang ada pada system, bukan berdasarkan alur kegiatan system.

Komponen-komponen yang terlibat dalam use case diagram :

1. Actor

Pada dasarnya *actor* bukanlah bagian dari *use case diagram*, namun untuk dapat terciptanya suatu *use case diagram* diperlukan beberapa *actor* dimana *actor* tersebut mempresentasikan seseorang atau sesuatu (seperti perangkat, sistem lain) yang berinteraksi dengan sistem. *Actor* digambarkan dengan *stick man* . *Actor* dapat digambarkan secara secara umum atau spesifik, dimana untuk membedakannya kita dapat menggunakan *relationship*

Notasi UML untuk actor

Membuat actor pada Rational Rose:

- Klik pada use case view package di browser
- Pilih New → actor, maka sebuah aktor baru bernama new class ditempatkan di browser.
- Tempatkan kursor pada documentation window, lalu ketikkan dokumentasi yang diinginkan.

2. Use Case

Use case ini merupakan bentuk fungsionalitas dari suatu system. Use case juga merupakan dialog antara actor dan system.

Cara menentukan use case pada system:

- Pola perilaku perangkat lunak apliaksi
- Gambaran tugas dari sebuah actor
- System atau benda yang memberikan sesuatu yang bernilai kepada actor
- Apa yang dikerjakan oleh suatu perangkat lunak

Membuat use Case

1. Klik kanan use case pada browser
2. Pilih New→ use case. Sebuah use case ditempatkan pada browser
3. Klik use case tersebut dan beri nama yang diinginkan

Membuat use case diagram utama

1. Klik kanan Main diagram pada Use Case View di browser, untuk membuka diagram
2. Klik actor dibrowser dan tarik actor ke dalam diagram
3. Ulangi langkah 2 untuk menambah actor yang diperlukan pada diagram
4. Klik untuk memilih sebuah use case di browser dan tarik use case kedalam diagram
5. Ulangi langkah 4 untuk menambah use case pada diagram

Notasi UML untuk Use Case

catatan: use case dan actor dapat diciptakan lewat toolbar

3. Relasi dalam use case

Ada beberapa relasi yang terdapat pada use case diagram :

1. Association, menghubungkan link antar elemen
2. Generalization, disebut juga inheritance(pewarisan) artinya sebuah elemen merupakan spesialisasi dari elemen lain.
3. Dependency, elemen bergantung dalam beberapa cara pada elemen lain
4. Aggregation, merupakan bentuk asosiasi, dimana elemen dapat berisi elemen lain

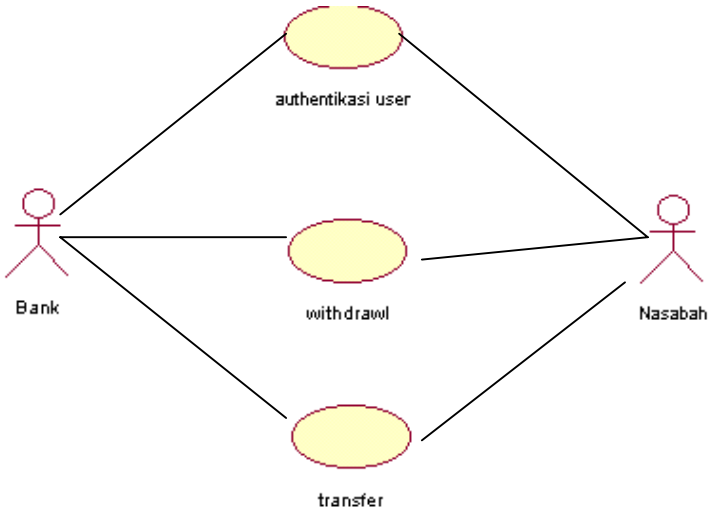
Tipe relasi/stereotype antara lain:

1. <<include>> yaitu perilaku yang harus terpenuhi agar sebuah event dapat terjadi, pada kondisi ini use case menjadi bagian dari use case lainnya
2. <<extends>> yaitu perilaku yang hanya berjalan pada kondisi tertentu, misal : jam alarm
3. <<communicates>> biasanya ditambahkan untuk asosiasi yang menunjukkan asosiasinya yaitu *communicates association*.

4. Use Case Diagram

Merupakan gambaran graphical dari beberapa atau semua *actor, use case* dan interaksi diantaranya yang memperkenalkan suatu system.

Contoh ATM dibawah ini menunjukkan use case diagram



Skenario use case diatas dapat dideskripsikan sebagai berikut:

1. Nama use case : Authentikasi nasabah
   Actor : Nasabah, Bank
   Type : Priamry
   Tujuan : Verifikasi nasabah (user)

- Langkah-langkah yang dilakukan **Actor** dan **Sistem**
  1. User Memasukkan kartu ATM (**Actor**)
  2. Mesin ATM meminta no PIN dari user (**Sistem**)
  3. User memasukkan PIN dan menekan OK (**Actor**)
  4. Mesin ATM memverifikasi no PIN dengan Bank (**Sistem**)
  5. Mesin ATM meminta jenis transaksi (**Sistem**)

2. Nama Use Case : Withdrawal

Actor : Bank, Nasabah(user)

Type : Primary

Tujuan : Penarikan uang cash

- Langkah-langkah yang dilakukan Actor dan Sistem

1. User memilih menu withdrawal (**Actor**)
2. ATM meminta jumlah uang yang akan ditarik (**Sistem**)
3. User memasukkan jumlah uang yang akan ditarik (**Actor**)
4. ATM melakukan validasi dengan saldo minimal dari rekening user (**Sistem**)
5. Update Saldo (**Sistem**)
6. ATM mengeluarkan uang(**Sistem**)
7. ATM mencetak nota dan mengeluarkan kartu (**Sistem**)

Tugas :

1. Evaluasi Use Case yang ada pada kasus ATM diatas dengan melakukan modifikasi diagram Use Casenya
2. Buatlah Use Case diagram sederhana system pembelian dengan kartu kredit di Supermarket dan deskripsikan masing-masing Actor dan Use Casenya.

MODUL III

OBJECT DAN CANDIDATE CLASS

Tujuan:

- Mahasiswa dapat mengetahui obyek dalam suatu system
- Mahasiswa dapat mengetahui candidate class dalam suatu system

Teori

Definisi Object dan Class

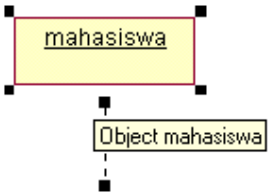
1. Object adalah gambaran dari entity, baik dunia nyata atau konsep dengan batasan-batasan dan pengertian yang tepat. *Object* bisa mewakili sesuatu yang nyata seperti komputer, mobil atau dapat berupa konsep seperti proses kimia, transaksi bank, permintaan pembelian, dll. Setiap *object* dalam sistem memiliki tiga karakteristik yaitu *State* (status), *Behaviour* (sifat) dan *Indentity* (identitas).

Cara mengidentifikasi *object*:

1. pengelompokan berdasarkan kata/frase benda pada skenario.
2. berdasarkan daftar kategori object, antara lain:

· object fisik, contoh:pesawat telepon
· spesifikasi/rancangan/deskripsi, contoh: deskripsi pesawat
· tempat, contoh:gudang
· transaksi, contoh: penjualan
· butir yang terlibat pada transaksi, contoh: barang jualan
· peran, contoh :pelanggan
· wadah, contoh : pesawat terbang
· piranti, contoh:PABX
· kata benda abstrak, contoh: kecanduan
· kejadian, contoh:pendaratan
· aturan atau kebijakan, contoh:aturan diskon
· catalog atau rujukan, contoh: daftar pelanggan

Notasi Object dalam UML



2. Class adalah deskripsi sekelompok *object* dari property (atribut), sifat (operasi), relasi antar *object* dan sematik yang umum. *Class* merupakan *template* untuk membentuk *object*. Setiap *object* merupakan contoh dari beberapa *class* dan *object* tidak dapat menjadi contoh lebih dari satu *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).

Dari skenario pada modul II untuk studi kasus pada ATM, kita dapat mendefenisikan *candidate class*, dimana *candidate class* secara kasar dapat diambil dari kata benda yang ada, atau sesuai dengan apa yang telah dijelaskan diatas.

Candidate class

No	Kategori obyek	Nama obyek
1	Obyek fisik	ATM, ATM Card
2	Transaksi	Withdrawal, Transfer
3	Hal-hal yangterlibat dlm transaksi	……..
4	Peran/actor	Nasabah, Bank
5	Piranti	ATM komputer
6	Proses	Withdrawal update
7	Katalog	Daftar Account

Dari beberapa candidate class diatas, dapat kita tetapkan beberapa class yang nantinya akan berpengaruh pada system antara lain : ATM Card, Withdrawal, Nasabah, Bank, Transfer dan mesin ATM.

Membuat *Class* dalam Rational Rose :

- Klik kanan *Logical View* pada browser
- Pilih new→ *Class*, maka sebuah *Class* akan muncul pada *browser*
- Untuk memberi nama, atribut dan operasi *Class*, klik kanan *Class* yang baru dibuat → *Spesification*, dan anda dapat memberi identitas yang berkaitan dengan *Class* yang telah anda buat.

Notasi class dalam UML:

Bagian dari Class:

a. Nama class

b. **Atribut** adalah salah satu *property* yang dimiliki oleh Class yang menggambarkan dari nilai yang dapat dimiliki *property* tersebut. Sebuah Class dapat memiliki beberapa atribut atau tidak memiliki sama sekali. Sebuah atribut mempresentasikan beberapa property dari sesuatu yang kita modelkan.

Cara membuat atribut pada *Class*:

- Klik kanan *Class* yang telah dibuat pada *browser*
- Pilih *New*→ *attribute,* ketikkan nama atribut yang anda inginkan.

c. Operasi

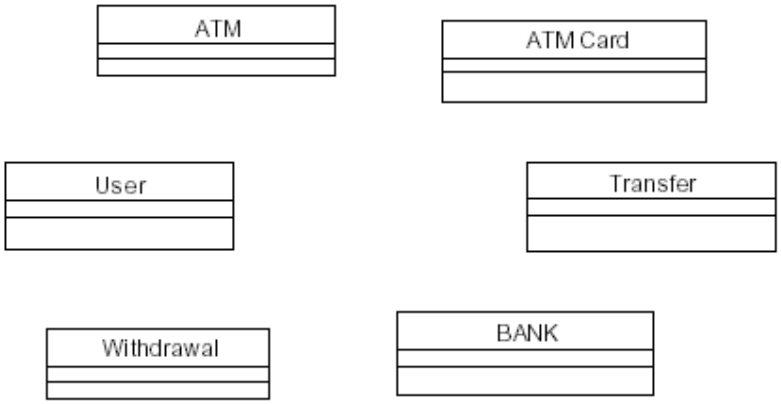
Operasi merupakan implementasi layanan dari beberapa *object* dari *class* yang mempengaruhi *behaviour* (sifat). Class dapat memiliki beberapa operasi atau sama sekali tidak ada operasinya.

Cara membuat operasi pada Class:

- Klik kanan class yang ada pada *browser*

Pilih *New* → *operation* , ketikkan operasi yang anda kehendaki dari *class* yang ada.

Notasi class untuk candidate class yang ada dalam system diatas: (untuk pemahaman class lebih lanjut ada pada modul V)



Tugas:

Lihat tugas no 2 pada Modul II, buatlah :

1. obyek-obyeknya dan gambarkan hubungan antar obyek
2. candidate class dari system tersebut
3. *class-class* beserta atribut dan operasinya
4. Gambarkan class-class tersebut pada Rational Rose.

MODUL IV

INTERACTION DIAGRAM

(Sequence dan Collaboration Diagram)

Tujuan

- Mahasiswa mengetahui interaksi diagram yang ada pada UML
- Mahasiswa dapat menggunakan interaksi diagram untuk system nyata

Teori

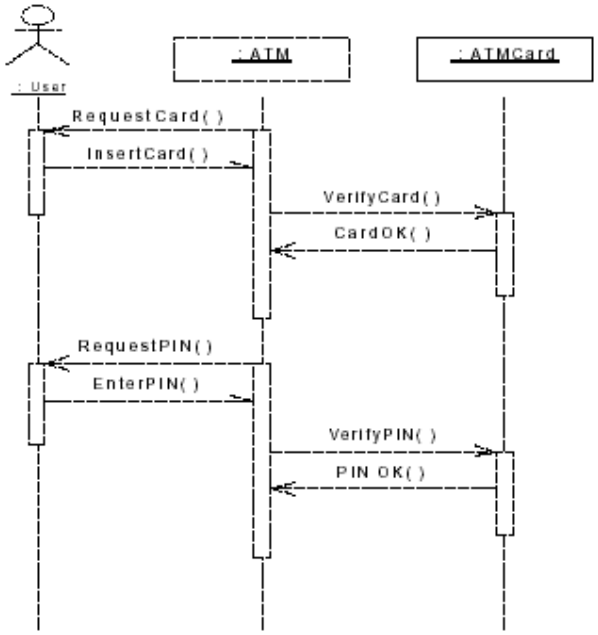
1. Use Case Realization

Fungsionalitas use case direpresentasikan dengan aliran peristiwa-peristiwa. Skenario digunakan untuk menggambarkan bagaimana use case-use case direalisasikan sebagai interaksi antara object-object. Interaction diagram merupakan model yang menjelaskan bagaimana sejumlah object bekerja sama dalam beberapa kelakuan. Interaksi diagram digunakan ketika kita ingin melihat kelakuan dari beberapa object dalam use case tunggal. Diagram ini baik saat menunjukkan kolaborasi diantara object-object, namun kurang baik dalam mendefinisikan behaviour(sifat). Ada dua macam bentuk interaksi diagram yaitu: *Sequence Diagram* dan ***Collaboration Diagram*.**

2. Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa message yang digambarkan terhadap waktu. Sequence diagram terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan output tertentu. Diawali dari apa yang memicu aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan output apa yang dihasilkan.

Contoh Sequence diagram (untuk authentikasi user):



Cara membuat Sequence Diagram:

- Klik kanan use case pada browser
- Pilih New→ sequence pada menu bar
- Ketika sequence diagram masih disorot, masukkan nama untuk sequence diagram tersebut

Cara membuat Object dan Message dalam Sequence Diagram:

- Klik ganda *sequence diagram* pada *browser*.
- Klik *actor* pada *browser*.
- Tarik *actor* ke dalam *sequence diagram.*
- Klik *object icon* pada *toolbar*
- Klik *sequence diagram window* untuk menempatkan *object*
- Ketika *object* masih disorot, masukkan nama *object*
- Ulangi langkah selanjutnya jika masih ingin memasukkan *object* dan *actor*
- Klik *object message icon* dari *toolbar*
- Klik *actor* atau *object sending message* lalu tarik garis *message* ke *actor* atau *object* yang menerima *message*

- Ketika *message* masih disorot,masukkan nama ke dalam *message* tersebut

Cara memasukkan objects ke dalam sebuah sequence diagram ke dalam classes

- Klik *class* ke browser
- Tarik class ke dalam object pada *sequence diagram*. Rose akan menambahkan nama class diawali dengan a : ke dalam nama *object*. Jika *object* belum mempunyai nama, maka nama diset menjadi : *class-name.*

3. Collaboration Diagram

Collaboration diagram merupakan cara alternatif untuk menggambarkan skenario dari system. Diagram ini menggambarkan interaksi object yang diatur object sekelilingnya dan hubungan antara setiap object dengan object yang lainnya.

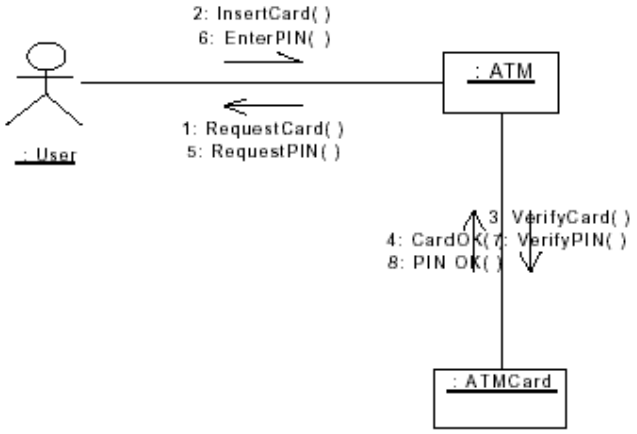
Collaboration diagram berisi:

- Object yang digambarkan dengan segiempat
- Hubungan antara object yang digambarkan dengan garis penghubung
- Pesan yang digambarkan dengan teks dan panah dari object yang mengirim pesan ke penerima pesan

Cara membuat Collaboration diagram dari sequence diagram

- Klik ganda sequencediagram pada browser
- Pilih browser, create collaboration diagram atau tekan F5
- Atur object dan message pada diagram seperlunya.

Gambar collaboration diagram (authentikasi user)



4. Perbedaan Sequence dan Collaboration Diagram

Sequence diagram memberikan cara untuk melihat scenario dari system berdasarkan waktu (apa yang terjadi pertama kali, apa yang terjadi selanjutnya). User akan lebih mudah membaca dan mengerti tipe diagram ini. Karenanya sangat berguna pada fase analisis awal. Sedangkan Collaboration diagram cenderung untuk memberikan gambaran besar dari scenario selama kolaborasi disusun dari object sekelilingnya dan hubungan antar object yang satu dengan lainnya.

Tugas

1. Buatlah Sequence diagram dan Collaboration diagram dengan menggunakan tugas no 2 pada Modul II?

MODUL V

RELASI DALAM OBJECT

Tujuan:

- Mahasiswa memahami relasi yang ada pada UML
- Mahasiswa dapat melakukan relasi antar class

Teori

Semua system terdiri dari *class-class* dan *object*. Kelakuan system dicapai melalui kerjasama antar *object*, misalkan seorang mahasiswa ditambahkan dalam daftar *class*, jika daftar class memperoleh message untuk menambahkan mahasiswa. Interaksi antar *object* disebut *object relationship*. Dua tipe yang ada pada saat analisa adalah *association* dan *aggregation* .

1. Relasi asosiasi (association relation)

Asosiasi adalah hubungan semantic bi-directional diantara class-class. Ini bukan aliran data seperti pada pemodelan desain dan analisa terstruktur, data diperbolehkan mengalir dari kedua arah. Asosiasi diantara class-class artinya ada hubungan antara object-object pada class-class yang berhubungan. Banyaknya object yang terhubung tergantung dengan beragamnya object (multiplicity) yang ada asosiasi

Cara membuat association relationship:

- Klik association icon dari toolbar
- Klik satu dari class association pada class diagram
- Tarik garis association kepada class yang ingin dihubungkan

Contoh relasi assosiasi :

2. Relasi pengumpulan (aggregation relation)

Aggregation relationship adalah bentuk khusus dari asosiasi dimana induk terhubung dengan bagian-bagiannya. Notasi UML untuk relasi ini adalah sebuah asosiasi dengan diamond putih melekat pada class yangmenyatkan induk.

Contoh relasi aggregation

Cara membuat relasi aggregation :

- Klik aggregation icon pada toolbar
- Klik class yang nertindak sebagai part (bagian) dan yang sebagai whole(induk).

3. Penamaan Relasi

Sebuah asosiasi dapat diberi nama, biasanya digunakan kata kerja aktif atau klausa kata kerja dengan cara pembacaan dari kiri ke kanan atau atas ke bawah. Agregasi tidak diberi nama karena agregasi menggunakan kata "mempunyai" atau "terdiri".

Cara memberi nama pada relasi:

- Klik garis relationship pada class diagram
- Masukkan nama relationship.

4. Indikator Multiplicity

Walaupun multiplicity ditentukan untuk class, multiplicity menentukan banyaknya object yang terlibat dalam relasi. Multiplicity menentukan banyaknya object yang terhubung satu dengan yang lainnya. Indikator multiplicity terdapat pada masing-masing akhir garis relasi, baik pada asosiasi ataupun agregasi.

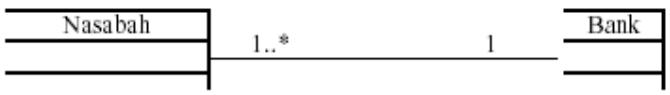
Contoh multiplicity adalah:

1 tepat Satu
0..* nol atau lebih
1..* satu atau lebih
0..1 nol atau Satu
5..8 range 5 s.d.8
4..6,9 range 4 s.d. 6 dan 9

Cara membuat multiplicity

- Klik ganda garis relationship untuk membuat specification
- Pilih tab detail untuk role yang akan dimodifikasi (Role A Detail atau Role B Detail)
- Masukkan multiplicity yang diinginkan

Contoh relasi class dengan multiplicity



- Sebuah *object* Nasabah berelasi dengan tepat satu *object* Bank, misal : Irma berelasi dengan Bank Dana.
- Sebuah *object* Bank berelasi dengan satu atau tak hingga nasabah. Misal : Bank Dana berelasi dengan Irma, Ilham, Norman, dsb.
-

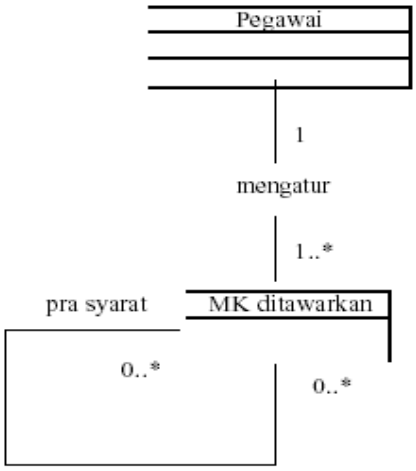
5. Refleksif Relationships

Adanya banyaknya obyek yang ada pada class dapat saling berkomunikasi satu dengan yang lainnya. Hal ini ditunjukkan pada class diagram sebagai reflexive association atau aggregation. Penamaan Role lebih disukai untuk digunakan pada reflexive relationship daripada penamaan association relationship.

Cara membuat Reflexive Relationship

- Klik association (aggreagation) icon di toolbar
- Klik class dan tarik garis association keluar class
- Lepaskan tombol mouse
- Klik dan tarik garis association kembali ke class
- Masukkan nama role dan multiplicity untuk tiap akhir dari Reflexive Association

Contoh relasi Refleksif



6. Menemukan Relationship

Untuk menemukan Relationship class-class yang ada dapat dilakukan dengan memeriksa scenario dan memeriksa

Tugas :

Lihat tugas no 2 pada Modul II, buatlah

1. Relasi refleksif
2. Relasi Aggregation DAN Relasi Assosiasi

MODUL VI

CLASS DIAGRAM dan *PACKAGE*

Tujuan:

- Mahasiswa dapat membuat *class* diagram dari suatu system
- Mahasiswa mampu mendefinisikan paket dalam UML
- Mahasiswa dapat menggambarkan relasi antar *class* dan typenya

Teori

1. *Class Diagram*

Class diagram adalah gambaran struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti containment, pewarisan, asosiasi, dan lain-lain. Untuk merancang *Class diagram*, *Rational Unified Process* menggunakan *use case realization* yang menggambarkan bagaimana realisasi dari setiap *use ca*se yang ada pada *use case* model. Untuk menggambarkan *use case realization* disini digunakan *class diagram owned by use case realization.* Setiap *use case* yang ada di*breakdown* sehingga akan terlihat jelas entiats-entitas apa saja yang terlibat dalam merealisasikan sebuah *use case*. Entitas-entitas ini akan menjadi *Candidate Class* dalam *Class diagram*.

Cara membuat *class* diagram

- Klik kanan salah satu paket di *browser* (misalnya *package*: *logical view*)
- Pilih *new*→*class diagram,* dan ketikan nama pada *class* diagram yang muncul

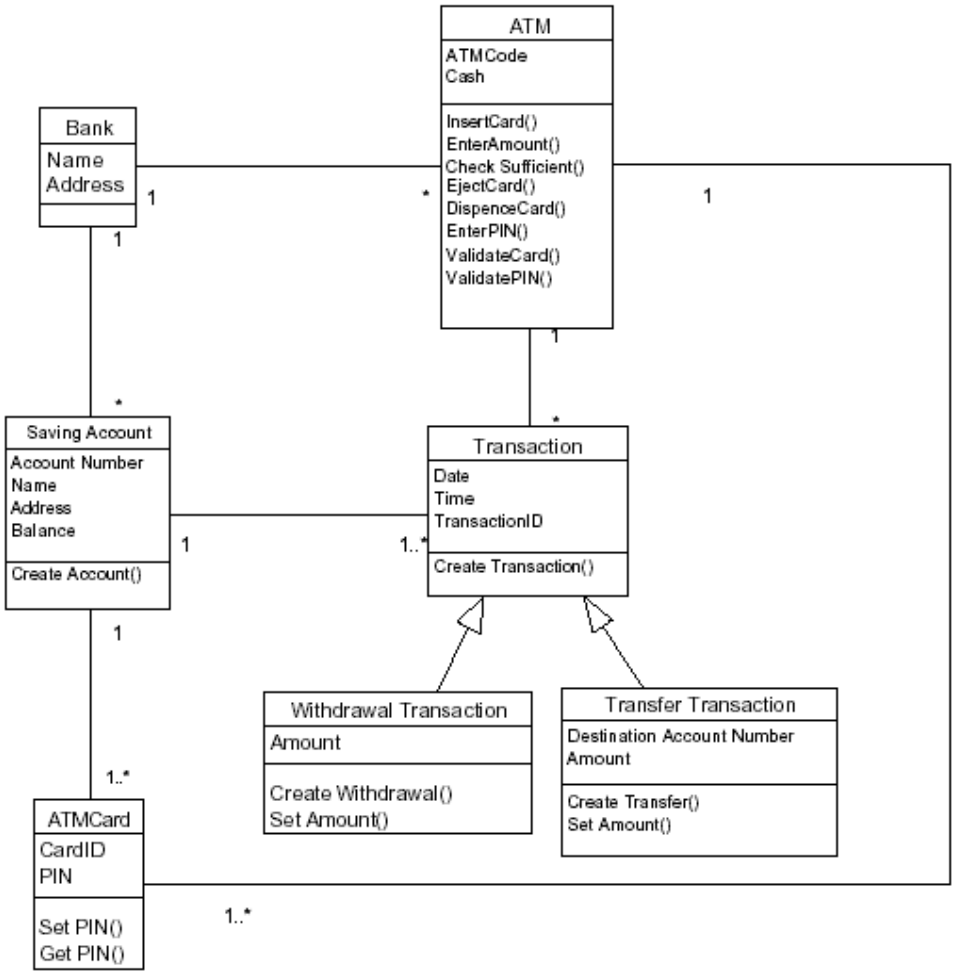
Cara membuat *class* diagram untuk menunjukkan atribut dan operasi dari sebuah paket:

- Klik kanan untuk *package* dibrowser
- Pilih new , *Class* diagram dan ketikkan nama yang anda inginkan.

Cara menambahkan *class* ke dalam sebuah diagram menggunakan menu query

- Klik ganda diagram pada browser.
- Pilih *Query : Add Classes*.
- Pilih *package* yang diinginkan
- Klik untuk memilih *classes* yang diinginkan dan klik tombol ">>>>>" untuk menambahkan semua *class*es ke dalam diagram.

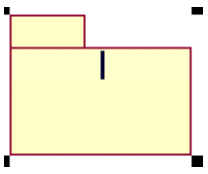
Contoh Class Diagram untuk system ATM



2. *Package*

Merupakan tinjauan logika dari suatu model yang berupa kumpulan clas/*package* itu sendiri. Tiap paket berisi interface yang direalisasikan oleh publik klas yaitu clas yang berkomunikasi dengan clas atau *package* lain. Misalnya data service, user service, bussines service, dst.

Notasi UML untuk *package*



Dengan mengelompokkan *Class* kedalam *package*, kita bisa melihat level yang lebih tinggi dari model kita atau kita bisa menggali model dengan lebih dalam dengan

melihat apa yang ada didalam *package*. Jika system yang akan dibangun komplek, *package* sebaiknya dibuat ditahap awal sebagai fasilitator komunikasi. Untuk system yang lebih sederhana, *class-class* yang didapat pada tahap analisa dapat dikelompokkan ke dalam *package-package*.

Cara membuat *package*:

- Klik kanan Logical View pada browser
- Pilih New → *Package*
- Dan beri nama *Package* yang anda kehendaki

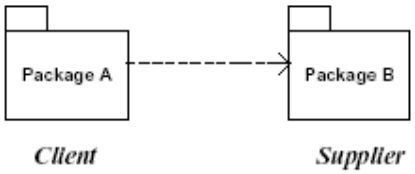
Relasi antar paket (*package*)

Relasi yang digunakan pada *package* relationship adalah jenis dependency relationship. Jika sebuah paket A tergantung pada paket B, hal ini berakibat satu atau lebih *class-class* dipaket A memulai berkomunikasi dengan satu atau lebih public *class* dipaket B. Paket A disebut *client package* dan paket B disebut *supplier package*.

Cara membuat relasi antar paket:

- Pilih dependency relationship icon dari toolbar.
- Klik dependent package dan tarik panah ke package yang berhubungan.

Contoh relasi antar paket (package):



Tugas:

1. Perbaikilah class diagram untuk system ATM diatas
2. Buatlah Class Diagram untuk tugas no 2 pada Modul II

MODUL VII

StateChart Diagram dan Activity Diagram

Tujuan:

- Mahasiswa dapat menentukan obyek dinamis dalam class
- Mahasiswa dapat menggambar *statechart diagram*
- Mahasiswa dapat menggambar *activity diagram*

Teori

1. Statechart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu state ke state lainnya) suatu objek pada sistem sebagai akibat dari stimuli yang diterima. Pada umumnya statechart diagram menggambarkan class tertentu (satu class dapat memiliki lebih dari satu statechart diagram).

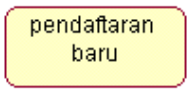
Elemen yang muncul pada statechart: State (keadaan sesaat), start, end, transition, action entry, do dan exit.

Cara membuat state (state transition) :

- Munculkan icon state dari toolbar
- Browse → State machine diagram → maka state muncul browser
- State yang muncul pada browser → New → State
- Beri nama state yang dikehendaki.

Notasi state pada UML

1. State



2. Start state dan Stop state

dan

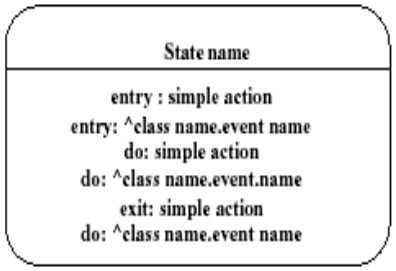
a. State Details

Action-action yang mengiringi seluruh *state transition* ke sebuah *state* mungkin ditempatkan sebagai sebuah *entry action* dalam *state*. Demikian juga, *action-action* yang mengiringi seluruh *state transition* keluar dari sebuah *state* mungkin ditempatkan sebagai sebuah aksi keluar dalam *state*. Kelakuan yang terjadi dalam *state* disebut *activity*. Sebuah *activity* memulai ketika *state* dimasukkan dan salah satu dari melengkapi atau diinterupsi oleh sebuah *state transition* yang keluar. Kelakuan mungkin sebuah *action* yang sederhana, atau kelakuan merupakan sebuah *event* yang terkirim ke *object* lain. Sesuai dengan *action-action* dan *guard -guard*, kelakuan ini secara tipikal dipetakan ke operasi-operasi dalam *object*.

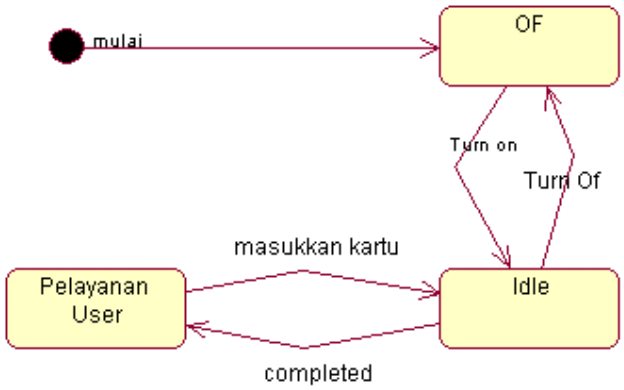
Membuat Entry Actions, Exit Actions dan Activities

1. Klik kanan pada *state* untuk menampilkan *shortcut*.
2. Pilih *menu specification*.
3. Pilih *General*.
4. Klik kanan pada *field Action* untuk menampilkan *shortcut*.
5. Pilih *menu Insert* untuk aksi yang disebut *entry*.
6. Double klik pada *entry* untuk menampilkan *Action Specification*.
7. Pilih tipe *action*: *action atau send event*.
8. Masukkan informasi *action* atau *send event*.
9. Pilih kapan *action* seharusnya terjadi: *on entry, on exit , on event dan do*.
10. Klik tombol OK untuk menutup *Action Specification* .
11. Klik tombol OK untuk menutup *State Specification*.

Contoh state detail



Contoh Statechart Diagram untuk ATM:



2. Activity Diagram

Activity diagram memodelkan *workflow* proses bisnis dan urutan aktivitas dalam sebuah proses. Diagram ini sangat mirip dengan *flowchart* karena memodelkan *workflow* dari satu aktivitas ke aktivitas lainnya atau dari aktivitas ke status. Menguntungkan untuk membuat *activity diagram* pada awal pemodelan proses untuk membantu memahami keseluruhan proses. *Activity diagram* juga bermanfaat untuk menggambarkan *parallel behaviour* atau menggambarkan interaksi antara beberapa *use case*.

Elemen-eleman activity diagram :

1. Status *start* (mulai) dan *end* (akhir)
2. Aktifitas yang merepresentasikan sebuah langkah dalam *workflow*.
3. *Transition* menunjukkan terjadinya perubahan status aktivitas (*Transitions show what state follows another*).
4. Keputusan yang menunjukkan alternatif dalam *workflow*.
5. *Synchronization bars* yang menunjukkan *subflow parallel*. *Synchronization bars* dapat digunakan untuk menunjukkan *concurent threads* pada *workflow* proses bisnis.
6. *Swimlanes* yang merepresentasikan *role* bisnis yang bertanggung jawab pada aktivitas yang berjalan.

Membuat Swimlanes

1. Klik kanan pada *use case* yang akan dibuat *activity diagram*, kemudian pilih *Select in Browser*. *Use case* yang dipilih akan tersorot pada *browser*.
2. Klik kanan *use case* yang tersorot di *browser*, kemudian klik *New, Activity Diagram*.
3. Beri nama *activity diagram*.
4. Buka *activity diagram* dengan double klik
5. Pilih *icon swimlane* dari *toolbar* dan klik ke dalam *activity diagram*.

6. Buka *Specification* dari *swimlane* dengan cara double klik *header swimlane* (*NewSwimlane*) pada diagram.
7. Beri nama *swimlane* dengan nama sesuai dengan *role* bisnis yang menjalankan aktivitas -aktivitas.
8. Klik OK.

Membuat status Aktifitas (Aktifitas)

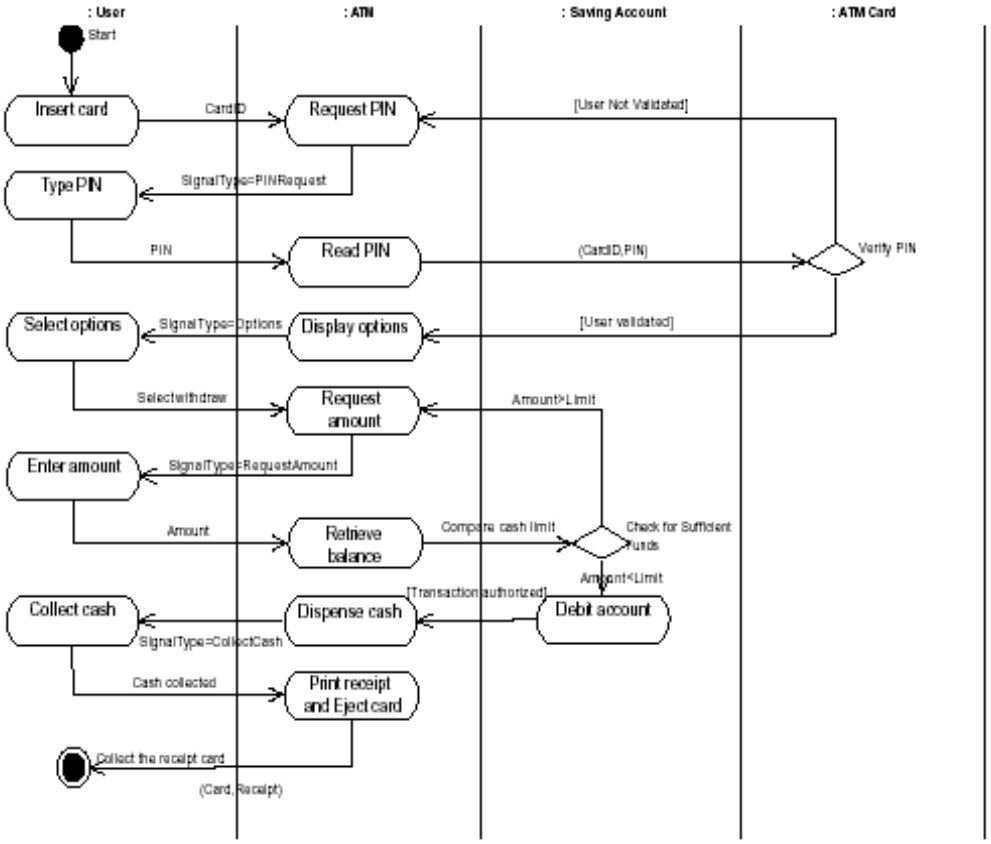
1. Klik *icon* status mulai di *toolbar* dan kemudian klik di *swimlane*.
2. Klik *icon* aktifitas di *toolbar* dan kemudian klik di *swimlane*.
3. Ganti nama *NewActivity* sesuai dengan aktiftas yang dilakukan
4. Untuk menunjukkan aktifitas pada nomor tiga berhubungan dengan status mulai , klik *icon state transition* di *toolbar*..
5. Klik dan *drag transition* dari status mulai menuju ke aktifitas nomor tiga.

Catatan: untuk membuat aktifitas dan *transition* lainnya dapat dilakukan dengan mengulang langkah 2 sampai 5.

Membuat Decision point

1. Klik icon *decision point* di *toolbar* dan kemudian sambungka n *transition* menuju dan dari *decision point* ke aktifitas-aktifitas yang berhubungan.
2. Buka *decision specification* dengan cara double klik *decision point*.
3. Masukkan nama *decision point* sesuai dengan fungsinya.
4. Untuk setiap *transition* yang keluar dari *decision point*, double klik untuk membuka *specification*-nya.
5. Pada *tab Detail*, masukkan label *guard condition* dengan fungsi yang sesuai di kotak *Guard Condition.* Arti *Guard Condition* adalah transition yang keluar dari *decision point* di-*triger* oleh *guard condition* pada *decision point*-nya.
6. Klik OK

Contoh activity diagram untuk system ATM



Tugas

Lihat tugas no 2 pada Modul II buatlah:

1. Statechart Diagram
2. Activity Diagram