MODUL PRAKTIKUM

TOPIK KHUSUS: SOFTWARE AGENT

Edisi ke – 1

February 2017

Disusun oleh :

Andri Pranolo

Laboratorium Sistem Cerdas

Program Studi Teknik Informatika

Fakultas Teknologi Industri

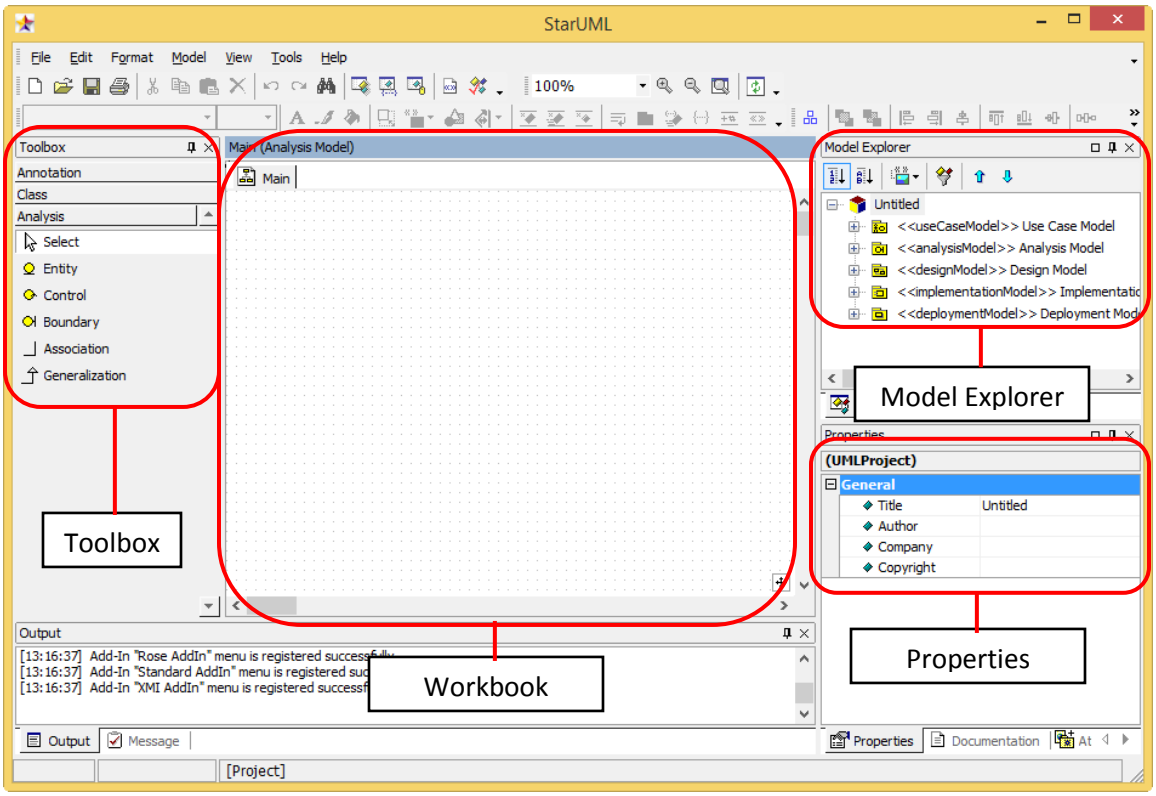
Universitas Ahmad Dahlan

Pertemuan 1

PEMODELAN OBJEK DENGAN STARUML

1.1. Pengantar Pemodelan StarUML

Dalam memodelkan objek menggunakan use case terdapat banyak cara yang dapat dilakukan salah satunya menggunakan sebuah modeling tools StarUML. Tools ini bersifat open-source dan cukup powerful dalam merancang use case diagram. StarUML adalah software pemodelan yang mendukung UML (Unified Modeling Language). Berdasarkan pada UML versi 1.4 dan dilengkapi 11 macam diagram yang berbeda, mendukung notasi UML 2.0 dan juga mendukung pendekatan MDA (Model Driven Architecture) dengan dukungan konsep UML. StarUML dapat memaksimalkan pruduktivitas dan kualitas dari suatu software project. Berikut ini merupakan tampilan utama dari StarUML.



1.2. Konsep Dasar StarUML

StarUML memiliki 3 konsep dasar dalam melakukan pemodelan objek, antara lain:

1. Model, View dan Diagram

   StarUML membuat perbedaan konseptual yang lebih jelas antara model, view dan diagram. Model adalah elemen yang memuat informasi untuk model software. View adalah suatu ekspresi visual dari informasi di dalam model. Sedangkan diagram adalah suatu koleksi dari elemen yang memberikan pemikiran user di dalam mendesain secara spesifik.

2. Project dan Unit

   - Project

     Project adalah unit manajemen dasar di dalam StarUML. Suatu project dapat mengatur satu atau lebih model software. Project merupakan top-level package yang selalu ada di dalam model software. Secara umum, satu project disimpan dalam satu file.

     Struktur Project

     Sebuah project terdiri dari sub-elements berikut ini:

Project Sub-Element	Deskripsi
Model	Elemen yang mengatur suatu model software.
Subsystem	Elemen yang mengatur model-model yang tercangkup dalam satu sub sistem.
Package	Elemen yang paling umum untuk mengatur elemen.

File Project

File project disimpan ke dalam format XML dengan extension ".UML". Semua model, view dan diagram yang dibuat dengan StarUML disimpan dalam satu file project.

File project berisikan informasi sebagai berikut:

1. UML profile yang digunakan dalam project.
2. Unit file yang direferensi oleh project.
3. Informasi untuk semua model yang ada di dalam project.
4. Informasi untuk diagram dan view yang ada di dalam project.

- Unit

  Ada beberapa kasus dimana satu project perlu disimpan di dalam beberapa file-file kecil sehingga para pengembang dapat bekerja di dalam satu project secara bersamaan. Di dalam kasus ini suatu project dapat mengatur bermacam-macam unit. Suatu unit mempunyai struktur hirarki dan berisikan beberapa sub-unit. Unit disimpan sebagai ".UML " file dan beberapa mengacu pada file project (.UML) atau unit file lainnya (.UNT).

  Komposisi Unit

  Hanya package, subsystem dan elemen model yang dapat membentuk satu unit. Semua elemen di bawah jenis elemen package ini disimpan di dalam masing-masing file unit (.UNT).

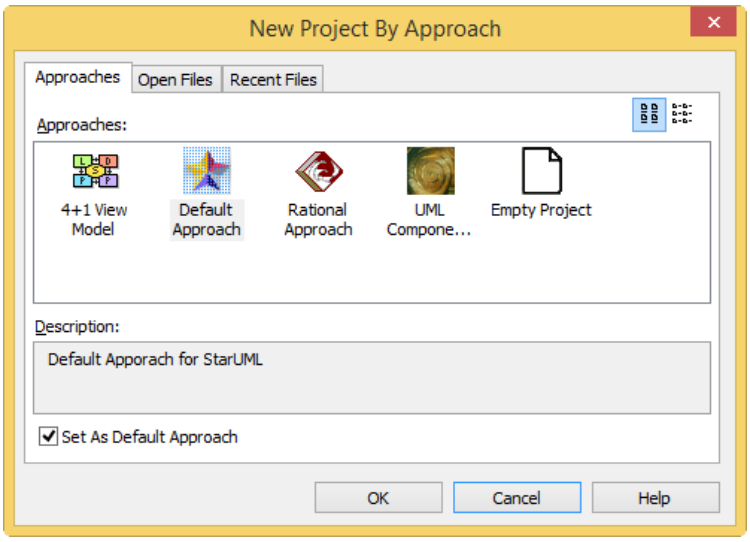
3. Module

Modul adalah suatu package yang menyediakan fungsi-fungsi baru dan fitur sebagai perluasan dari StarUML. Modul dapat dibuat sebagai kombinasi dari beberapa elemen-elemen extension dan juga membuat beberapa jenis elemen-elemen di dalam suatu modul.

1.3. Managing Project

Sekarang, kita coba untuk membuat project baru menggunakan StarUML. Langkah-langkahnya adalah sebagai berikut:

1. Dalam membuat project baru kita dapat menggunakan dua cara, yang pertama pilih menu File – New Project (Ctrl+N), maka project baru akan langsung terbentuk. Cara kedua dengan memilih menu File – New Project By Approach (Ctrl+I), maka akan muncul kotak dialog seperti di bawah ini.



Masing-masing model di atas memiliki fungsinya sendiri sesuai kebutuhan yang kita inginkan.

- 4+1 View Model

  Model UML yang dapat mendukung view 4+1 dari arsitektur software yang akan dibangun. View 4+1 maksudnya adalah jenis-jenis dari view UML tersebut antara lain use case view (scenario), logical view, development view, process view, dan pysical view.

- Default Approach

Merupakan model default yang telah ditentukan oleh StarUML. Dalam pilihan ini, model-model yang telah ditentukan antara lain use case model, analysis model, design model, implementation model, dan deployment model.

- Rational Approach

  Model UML yang digunakan untuk memodelkan software rasional. View-view yang telah disediakan untuk model ini antara lain use case view, logical view, component view, dan deployment view.

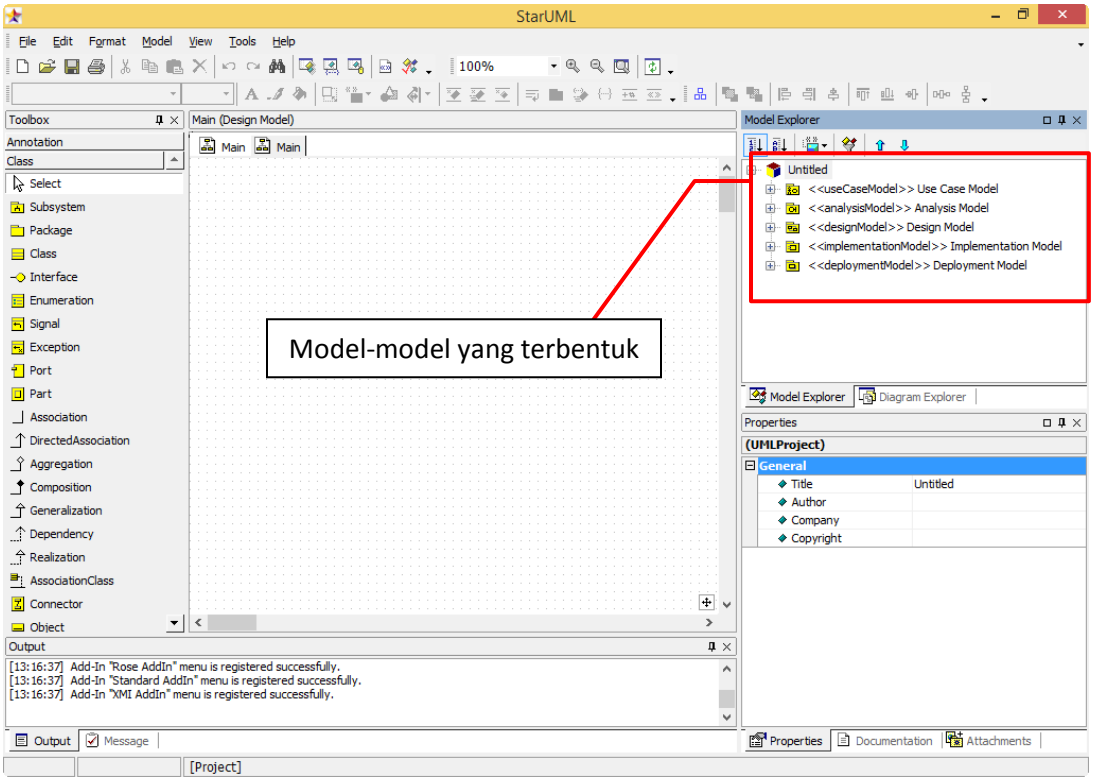
- UML Component Approach

  Model ini digunakan untuk menggambarkan alur proses objek menggunakan komponen UML. Proses-proses yang digambarkan dalam model ini meliputi requirement (business concept model, use case model) dan specification (business type model, interface specification, component specification, component architecture).

- Empty Project

  Berupa project kosong yang akan kita tentukan sendiri model-model apa saja yang akan kita butuhkan.

2. Pada kesempatan ini kita pilih saja New Project. Maka akan terbentuk tampilan seperti ini.


StarUML
File Edit Format Model View Tools Help
Toolbox
Annotation
Class
Select
Subsystem
Package
Class
Interface
Enumeration
Signal
Exception
Port
Part
Association
DirectedAssociation
Aggregation
Composition
Generalization
Dependency
Realization
AssociationClass
Connector
Object
Main (Design Model)
Main
Main
Model-model yang terbentuk
Model Explorer
Untitled
<<useCaseModel>> Use Case Model
<<analysisModel>> Analysis Model
<<designModel>> Design Model
<<implementationModel>> Implementation Model
<<deploymentModel>> Deployment Model
Model Explorer
Diagram Explorer
Properties
(UMLProject)
General
Title
Untitled
Author
Company
Copyright
Output
[13:16:37] Add-In "Rose AddIn" menu is registered successfully.
[13:16:37] Add-In "Standard AddIn" menu is registered successfully.
[13:16:37] Add-In "XMI AddIn" menu is registered successfully.
Output
Message
Properties
Documentation
Attachments
[Project]

Pertemuan 2

UNIFIED MODELING LANGUAGE (UML)

2.1. Pengantar UML

UML (Unified Modeling Language) merupakan pengganti dari metode analisis berorientasi object dan design berorientasi object (OOA&D) yang dimunculkan sekitar akhir tahun 80-an dan awal tahun 90-an.

UML merupakan gabungan dari metode Booch, Rumbaugh (OMT) dan Jacobson. Tetapi UML ini akan mencakup lebih luas daripada OOA&D. Pada pertengahan pengembangan UML dilakukan standarisasi proses dengan OMG (Object Management Group) dengan harapan UML akan menjadi bahasa standar pemodelan pada masa yang akan datang.

UML disebut sebagai bahasa pemodelan bukan metode. Kebanyakan metode terdiri paling sedikit prinsip, bahasa pemodelan dan proses. Bahasa pemodelan (sebagian besar grafik) merupakan notasi dari metode yang digunakan untuk mendesain secara cepat.

Bahasa pemodelan merupakan bagian terpenting dari metode. Ini merupakan bagian kunci tertentu untuk komunikasi. Jika kita ingin berdiskusi tentang desain dengan seseorang, maka kita hanya membutuhkan bahasa pemodelan bukan proses yang digunakan untuk mendapatkan desain.

UML merupakan bahasa standar untuk penulisan blueprint software yang digunakan untuk visualisasi, spesifikasi, pembentukan dan pendokumentasian alat-alat dari sistem perangkat lunak.

2.2. Diagram UML

Dalam UML terdapat berbagai macam diagram untuk memodelkan objek-objek yang akan digunakan dalam merancang suatu sistem berbasis objek. Semisal terdapat suatu kasus seperti ini:

"Suatu sekolah ingin membuat sistem yang dapat mencatat transaksi peminjaman buku yang terdapat pada perpustakaan sekolah tersebut. Pihak sekolah menginginkan agar sistem tersebut dapat mencatat data anggota, data pustakawan, data katalog buku, data transaksi (peminjaman dan pengembalian), menampilkan katalog buku, dan menghasilkan laporan transaksi. Perlu diketahui juga untuk alur prosesnya yaitu pada saat akan melakukan peminjaman buku, anggota terlebih dahulu harus melakukan login untuk memastikan anggota tersebut valid. Setelah login berhasil, anggota dapat melihat katalog buku dan dapat memilih buku yang akan dipinjam. Kemudian anggota dapat melakukan transaksi peminjaman untuk buku-buku yang telah dipilih. Terakhir pustakawan dapat melihat laporan transaksi perperiode tertentu."

Berdasarkan kasus di atas, bagaimana memodelkan objek beserta alur prosesnya menggunakan diagram UML?

Sekarang mari kita coba untuk membuat diagram-diagram UML untuk analisa kasus di atas menggunakan StarUML.

1. Use Case Diagram

Use case adalah gambaran fungsionalitas dari suatu sistem, sehingga pengguna sistem paham dan mengerti mengenai kegunaan sistem yang akan dibangun.

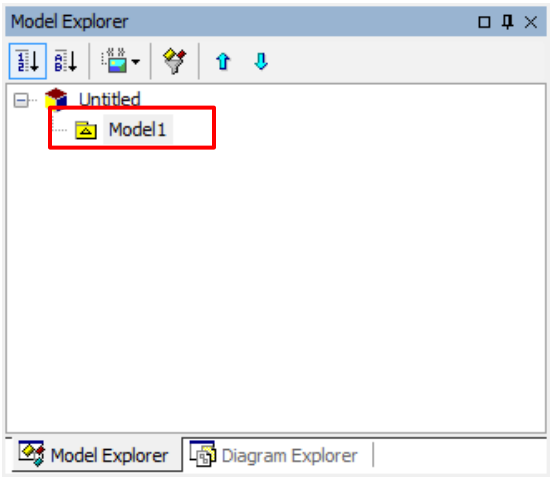
Catatan:

Use case diagram adalah penggambaran sistem dari sudut pandang pengguna sistem tersebut (user), sehingga pembuatan

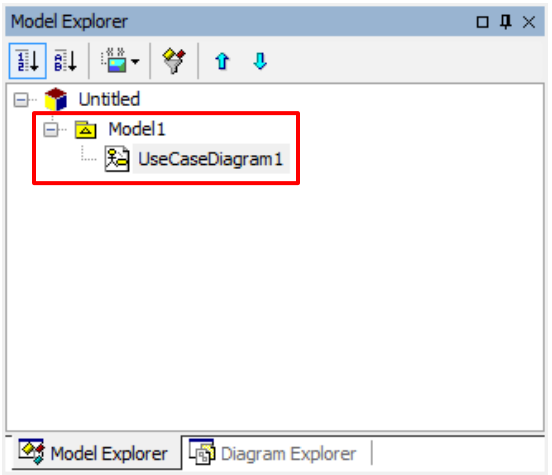
use case lebih dititikberatkan pada fungsionalitas yang ada pada sistem, bukan berdasarkan alur atau urutan kejadian.

Membuat Use Case Diagram

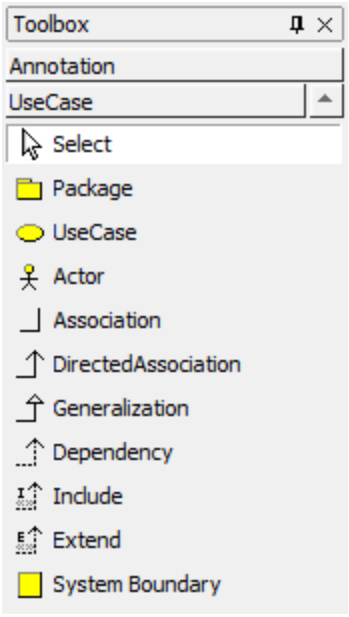
- Sebelum membuat use case diagram, terlebih dahulu kita buat modelnya (apabila model belum terbentuk) dengan cara pilih menu Model – Add, kemudian pilih Model. Maka akan terbentuk model seperti ini.



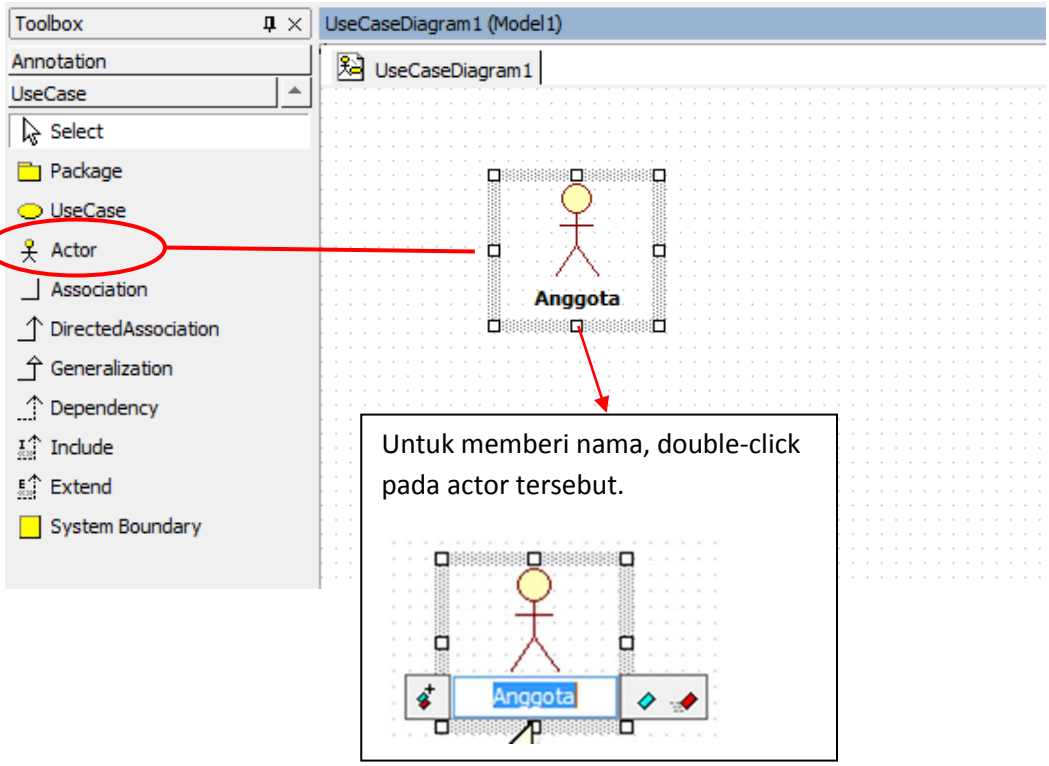
- Sekarang kita buat use case diagramnya dengan cara pilih menu Model – Add Diagram, lalu pilih Use Case Diagram. Sehingga akan menjadi seperti ini.



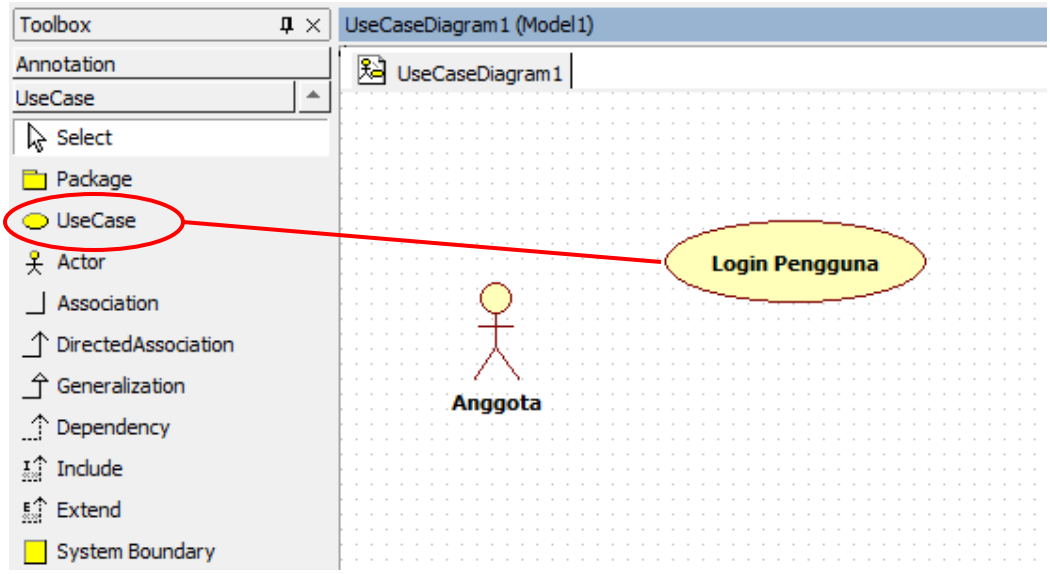
Kemudian kita juga akan mendapatkan tampilan toolbox seperti ini (toolbox untuk membuat use case diagram).



- Untuk merancang use case, klik salah satu notasi pada toolbox kemudian masukkan ke dalam lembar kerja. Semisal kita akan memasukkan actor dengan nama "Anggota".



- Selanjutnya pilih usecase pada toolbox dan masukkan ke dalam lembar kerja, kemudian beri nama sesuai kebutuhan sistem.



- Untuk menghubungkan antara actor dengan usecase ataupun usecase dengan usecase, terdapat beberapa notasi relasi yang dapat digunakan, antara lain:
  - **Association**

    Menghubungkan link antar elemen secara bidirectional.
  - **Directed Association**

    Menghubungkan link antar elemen yang sama secara langsung.
  - **Generalization**

    Dapat juga disebut inheritance (pewarisan), menghubungkan suatu elemen yang merupakan spesialisasi dari elemen lainnya.
  - **Dependency**

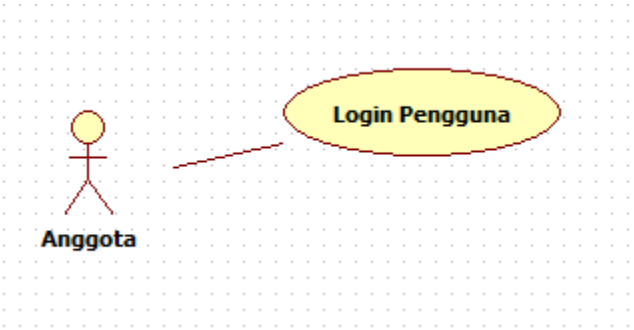
    Menghubungkan suatu elemen tertentu yang memiliki ketergantungan dengan elemen lainnya.
  - **Include**

    Suatu kondisi yang harus terpenuhi agar sebuah usecase dapat dijalankan, di mana pada kondisi ini suatu usecase merupakan bagian dari usecase lainnya.

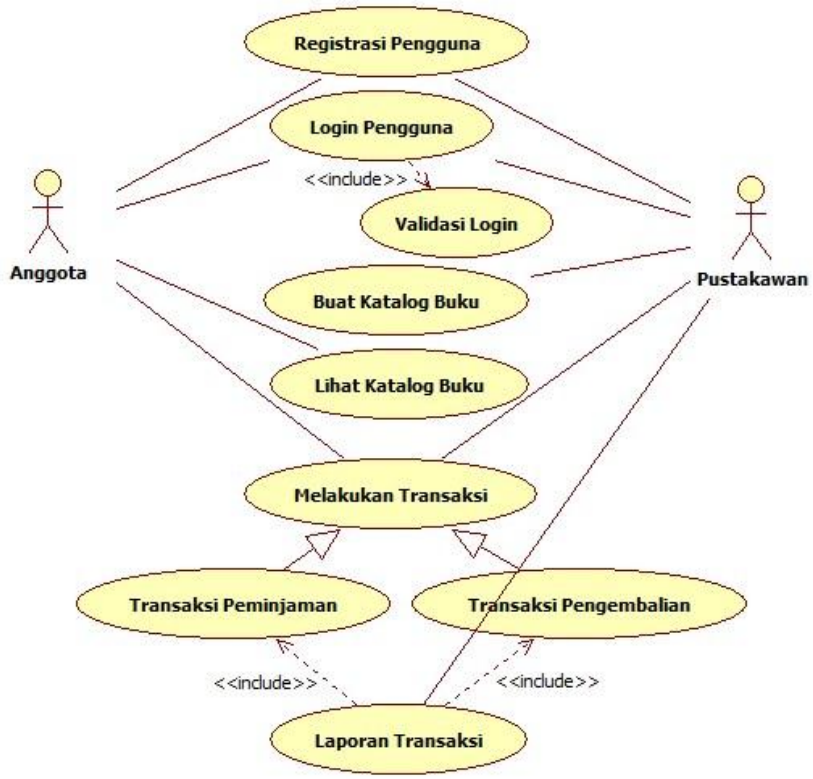
- **Extend**

  Sebuah usecase yang hanya dapat dijalankan di bawah kondisi tertentu.

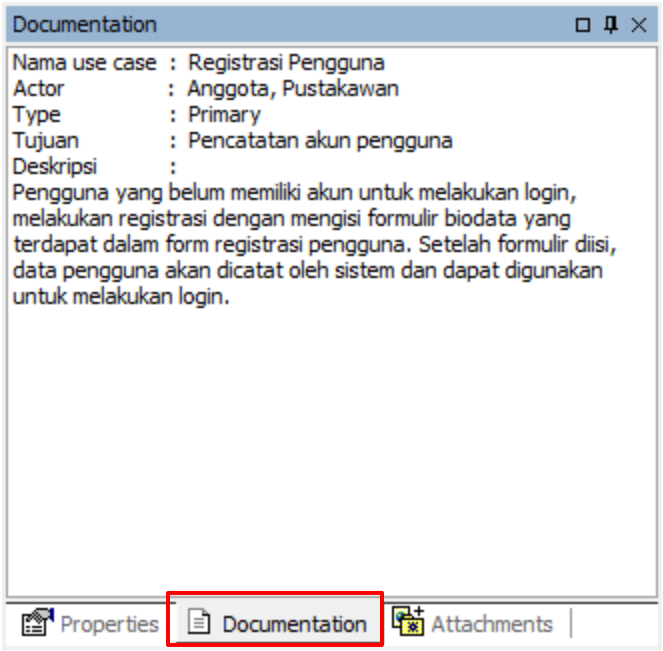
Misalkan kita gunakan notasi Association untuk menghubungkan actor dengan usecasenya, caranya adalah arahkan pada actor kemudian tarik menuju usecasenya dan lepaskan.



- Sama seperti langkah di atas, buatlah rancangan usecasenya untuk kasus yang kita bahas di awal menjadi seperti ini.



- Setelah selesai, berikan juga deskripsi pada setiap usecase agar lebih mudah dipahami oleh pengguna (user) melalui pallete Documentation.



2. Sequence Diagram

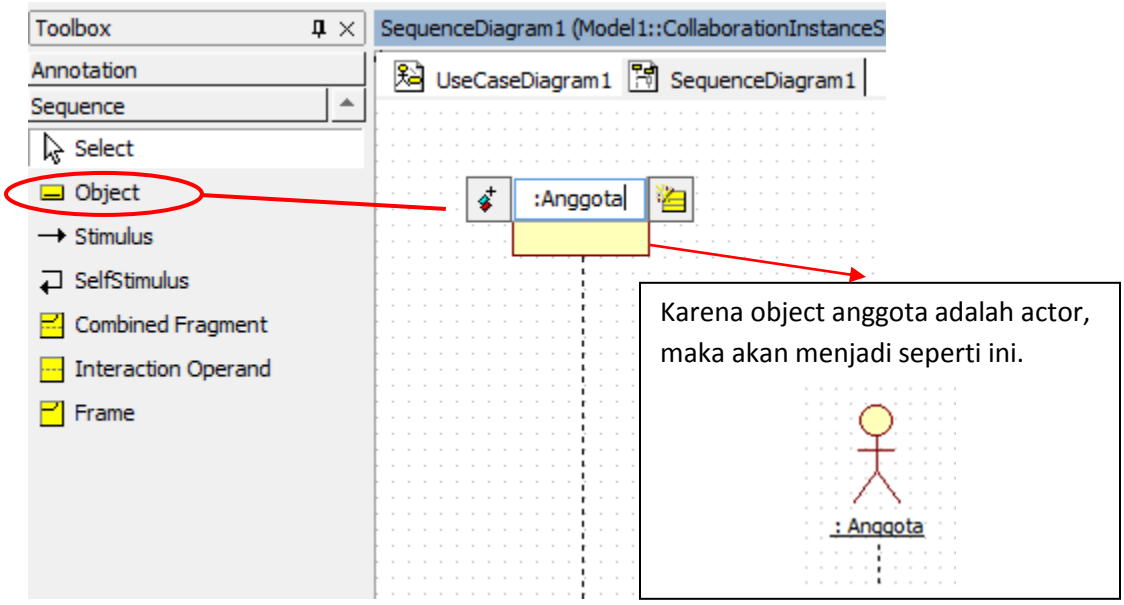
Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa message yang digambarkan terhadap waktu.

Sequence diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). Sequence diagram dapat digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan output tertentu.

Masing-masing objek, termasuk aktor, memiliki lifeline vertikal. Message digambarkan sebagai garis berpanah dari satu objek ke objek lainnya.

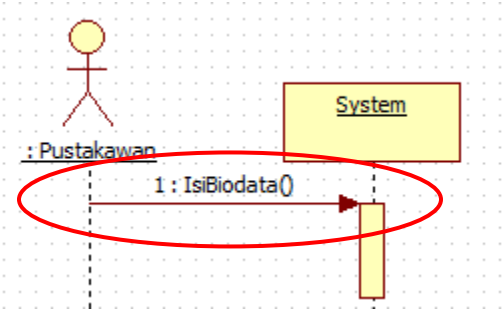
Membuat Sequence Diagram

- Pilih menu Model – Add Diagram, lalu pilih Sequence Diagram.
- Sekarang kita akan mencoba untuk memasukkan object ke dalam lembar kerja, caranya pilih object pada toolbox dan masukkan ke dalam lembar kerja.

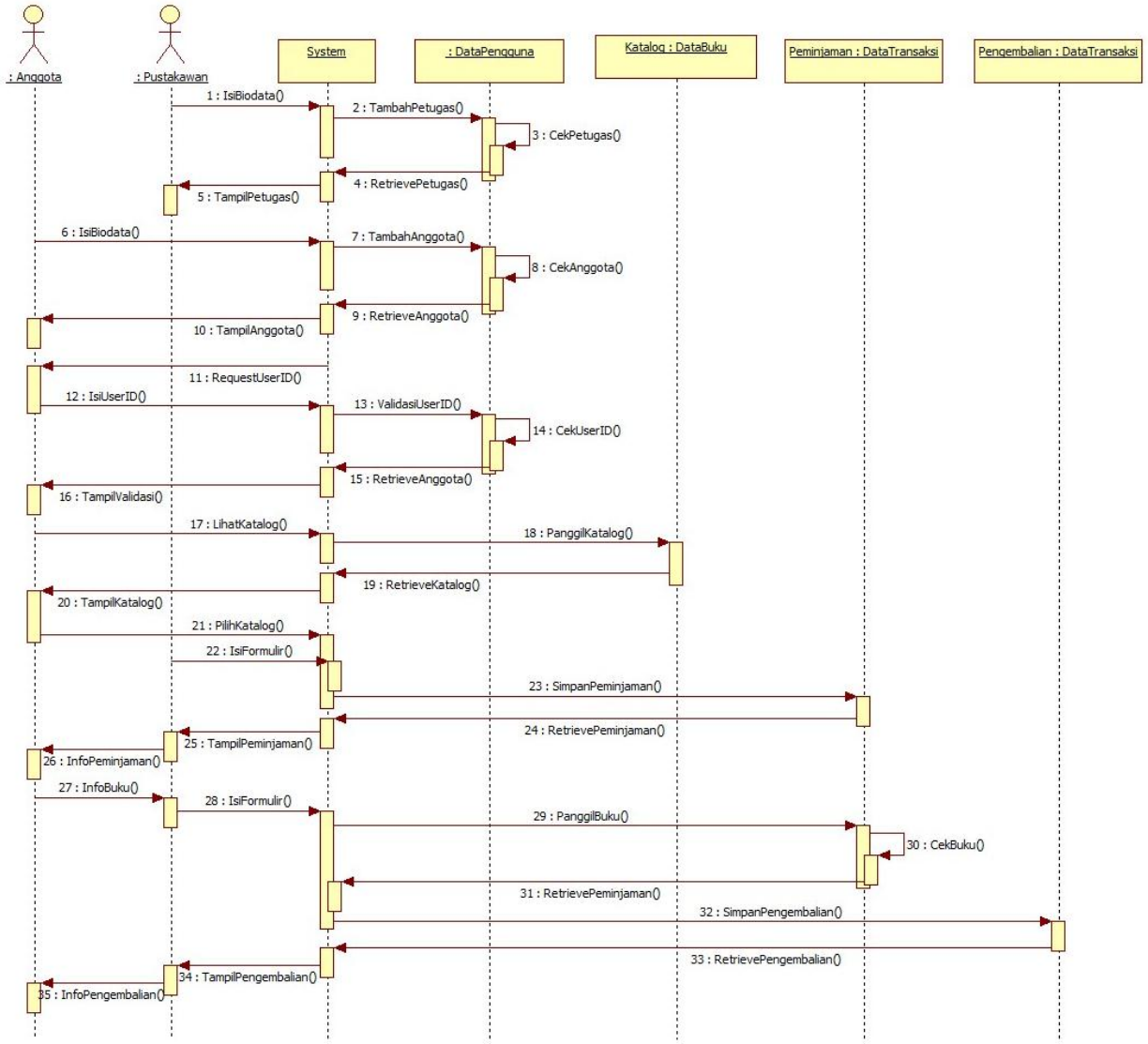


- Untuk memberikan stimulus (pesan) dari satu object ke object lainnya, dapat dilakukan dengan dua cara, yaitu:
  - Stimulus, memberikan pesan dari satu object ke objek lain.
  - SelfStimulus, memberikan pesan ke objek itu sendiri.

  Untuk penggunaannya adalah dengan cara menarik stimulus dari object yang memberi pesan ke object yang akan diberi pesan, kemudian tentukan pesan yang akan diberikan.



- Sama seperti langkah di atas, buatlah rancangan sequence diagram untuk kasus yang kita bahas di awal menjadi seperti ini.



3. Collaboration Diagram

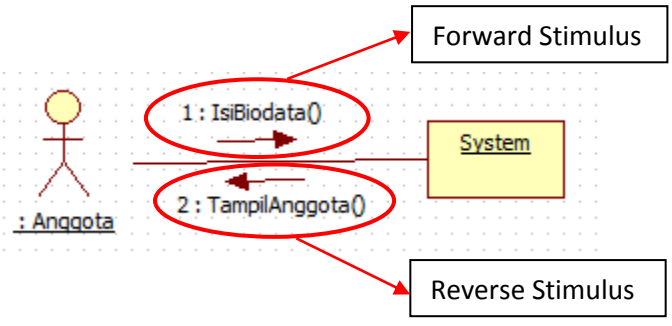
Collaboration diagram merupakan cara alternatif untuk menggambarkan skenario dari sistem. Diagram ini menggambarkan interaksi object yang diatur object sekelilingnya dan hubungan antara setiap object dengan object yang lainnya. Collaboration diagram berisi :

- 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.

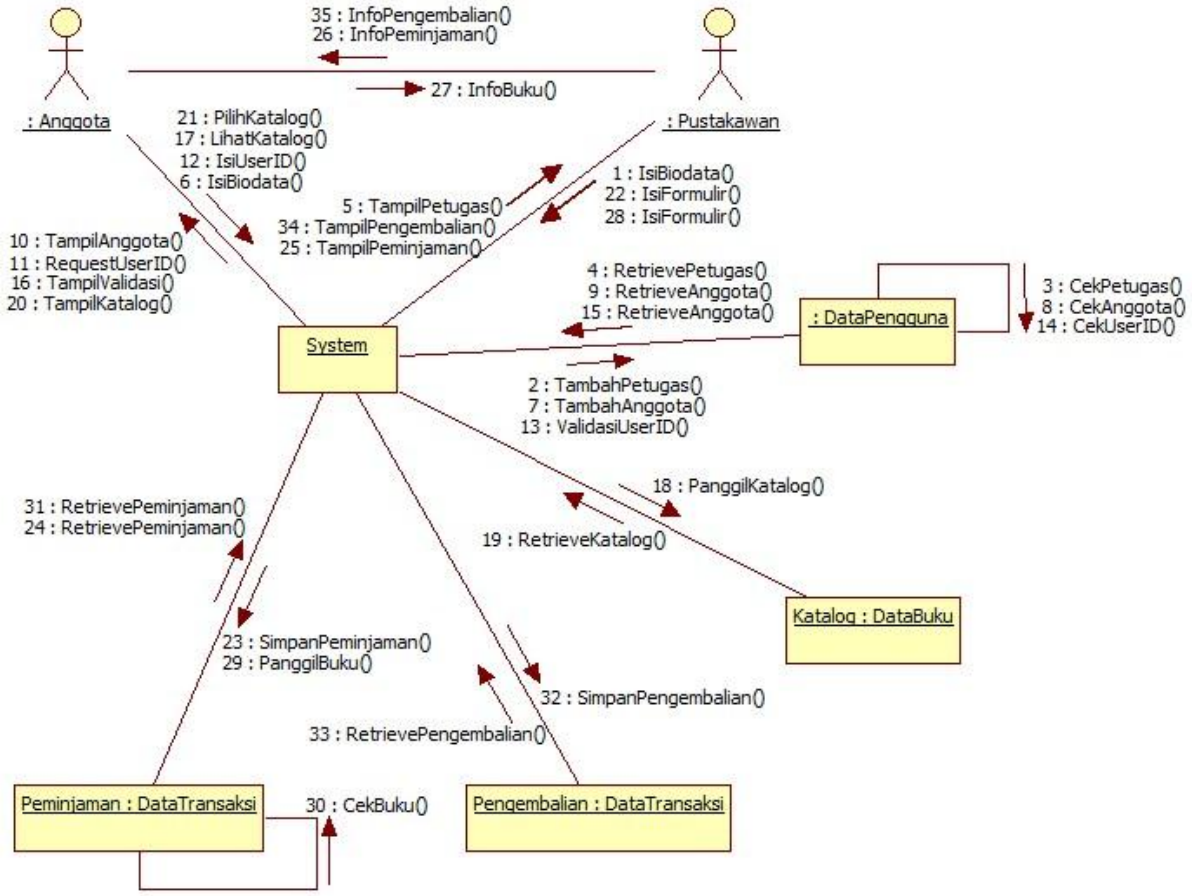
Membuat Collaboration Diagram

- Pilih menu Model – Add Diagram, lalu pilih Collaboration Diagram.
- Untuk memasukkan object ke dalam lembar kerja, caranya sama seperti pada langkah yang telah dibahas sebelumnya.
- Untuk menghubungkan antar object, gunakan link maupun selflink sesuai dengan penerima pesan yang dituju.
- Dalam menentukan arah stimulus (pesan) dari suatu object ke object lain, terdapat dua arah yang dapat diberikan, yaitu:
  - Forward Stimulus (Arah Maju), pesan diberikan dari object pengirim kepada object penerima.
  - Reverse Stimulus (Arah Balik), pesan dikembalikkan dari object penerima kepada object pengirim.

  Cara penggunaannya adalah dengan menempatkan pada link yang menghubungkan antar object, kemudian tentukan pesan yang akan diberikan.



- Sama seperti langkah di atas, buatlah rancangan collaboration diagram untuk kasus yang kita bahas di awal menjadi seperti ini.



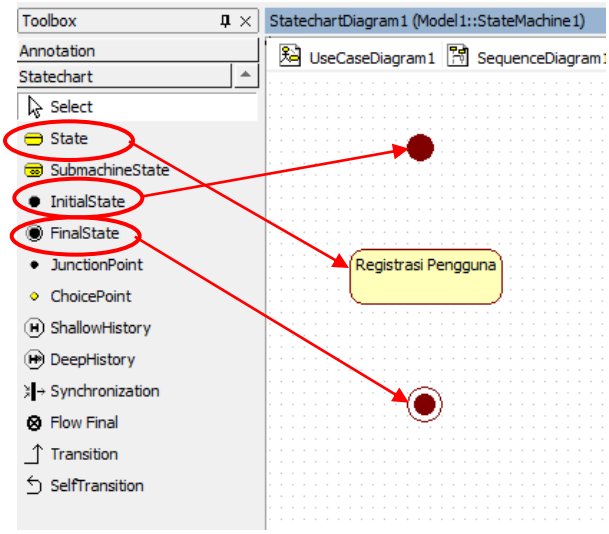
4. Statechart Diagram

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

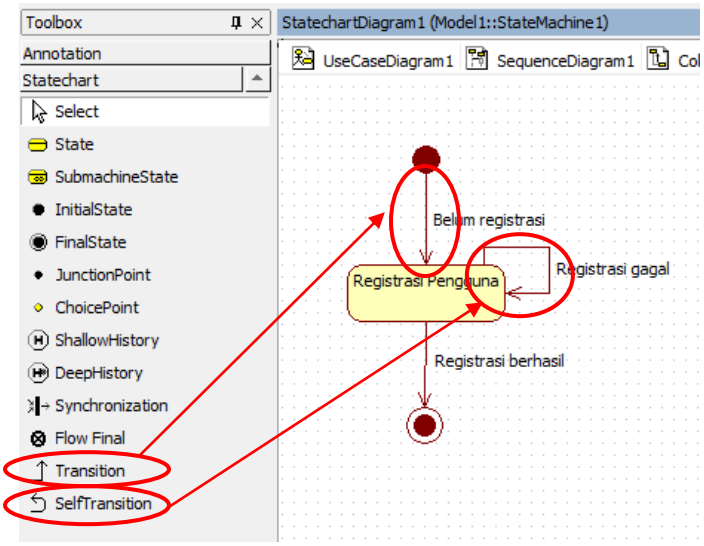
Membuat Statechart Diagram

- Pilih menu Model – Add Diagram, lalu pilih Statechart Diagram.

- Sekarang kita masukkan initialstate, state, dan finalstate ke dalam lembar kerja.



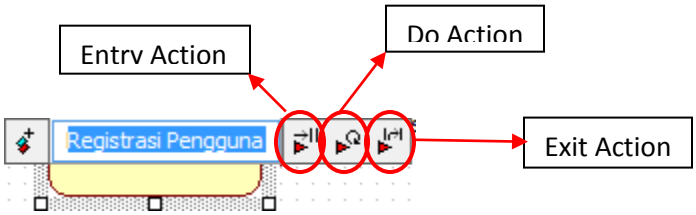
- Selanjutnya kita berikan transisi dari suatu state ke state lainnya, caranya pilih transition pada toolboox kemudian arahkan pada state awal dan tarik menuju state berikutnya, lakukan seterusnya sampai state akhir. Berikan juga keterangan pada transisi tersebut.



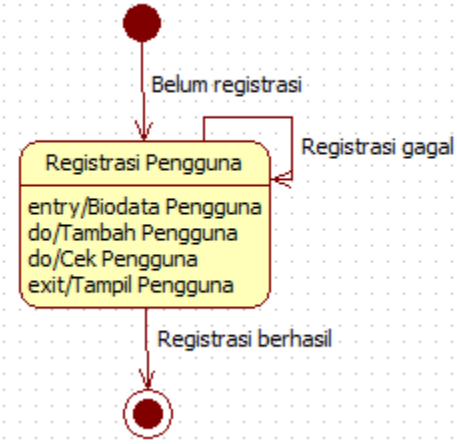
- Di dalam state kita juga dapat memberikan suatu aksi tertentu yang dapat dilakukan pada saat berada di state tersebut. Terdapat tiga tipe aksi yang dapat diberikan, yaitu:
  - Entry Action, aksi yang diberikan pada saat masuk state.

- Do Action, aksi yang diberikan pada saat berada di state.
- Exit Action, aksi yang diberikan pada saat keluar dari state.

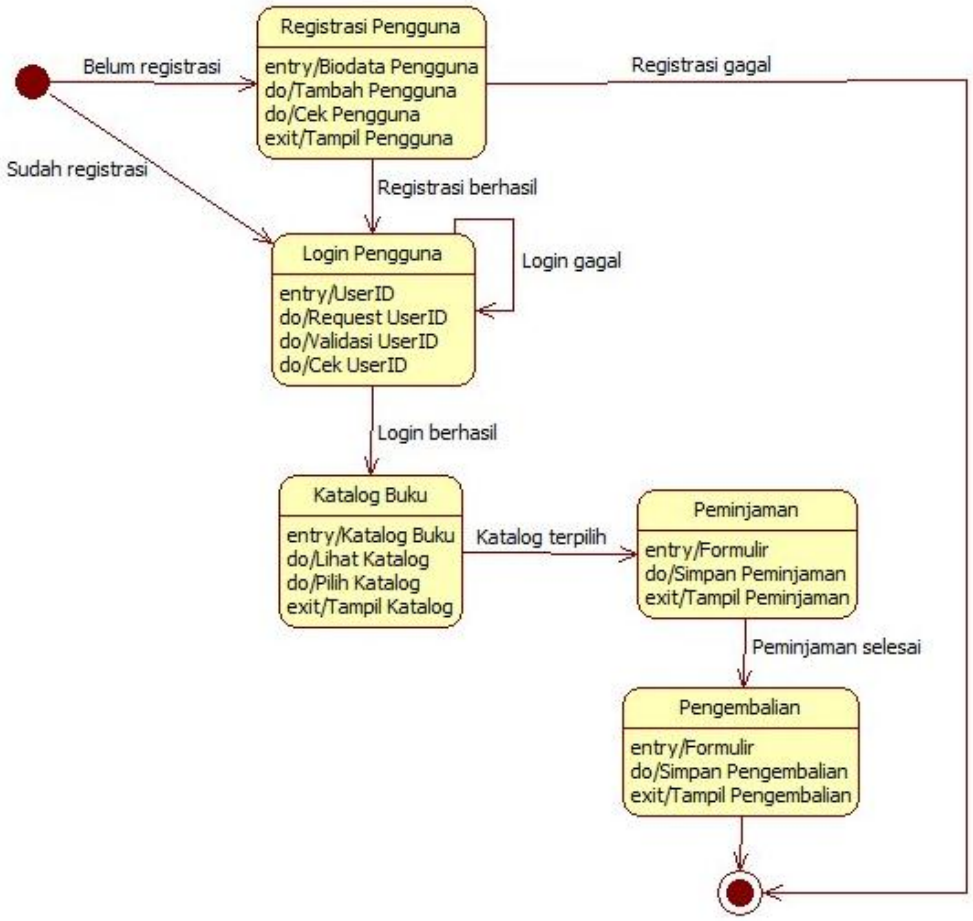
Caranya adalah dengan men-double click pada state yang akan diberikan aksi tertentu.



- Kemudian tentukan aksi-aksi yang akan diberikan pada state tersebut, sehingga menjadi seperti ini.



- Sama seperti langkah di atas, buatlah rancangan statechart diagram untuk kasus yang kita bahas di awal menjadi seperti ini.



5. Activity Diagram

Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, decision yang mungkin terjadi, dan bagaimana mereka berakhir. Activity diagram juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

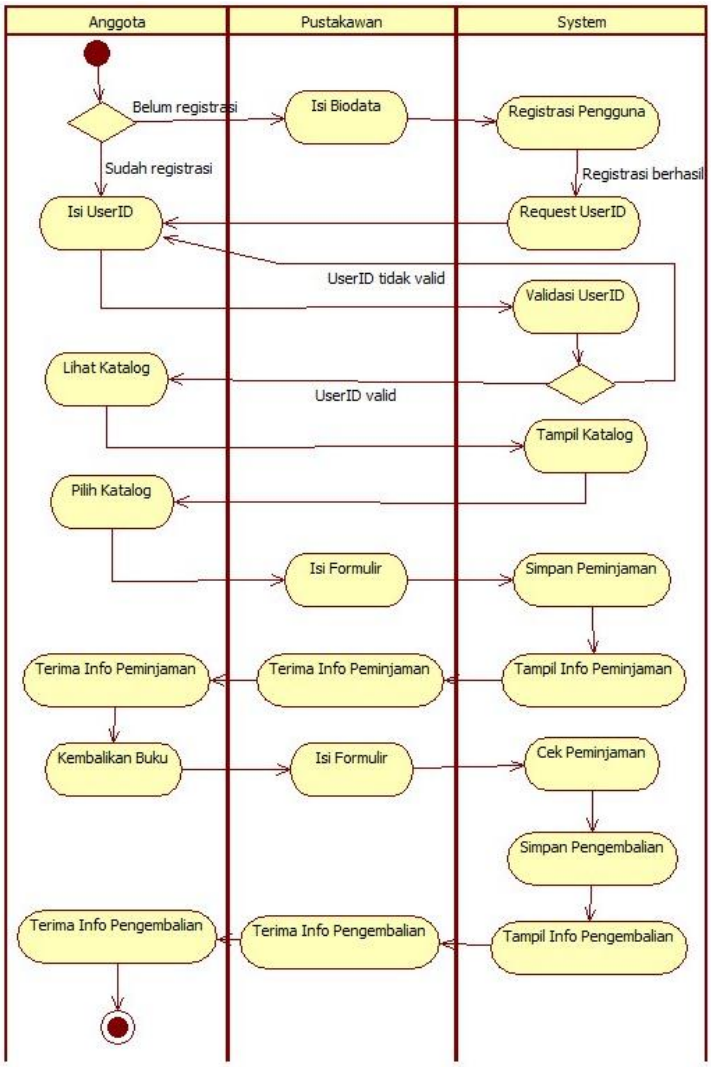
Activity diagram merupakan state diagram khusus, di mana sebagian besar state adalah action dan sebagian besar transisi di-trigger oleh selesainya state sebelumnya (internal processing).

Oleh karena itu activity diagram tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem)

secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Membuat Activity Diagram

- Pilih menu Model – Add Diagram, lalu pilih Activity Diagram.
- Pada langkah ini sama seperti pada langkah-langkah yang telah dibahas sebelumnya. Kita masukkan swimlane (vertical), initialstate, actionstate, dan finalstate ke dalam lembar kerja. Kemudian berikan juga transisi pada setiap state dan buatlah rancangan activity diagram untuk kasus yang kita bahas di awal menjadi seperti ini.



6. **Class Diagram**
7. **Component Diagram**
8. **Deployment Diagram**

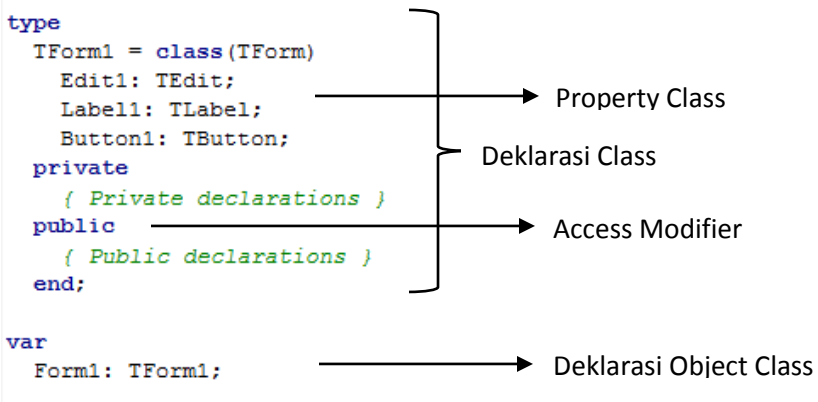
Pertemuan 3

OBJECT ORIENTED PROGRAMMING (FUNCTION DAN PROCEDURE)

Dalam pertemuan kali ketiga ini, akan dibahas mengenai konsep serta penerapan object oriented programming dalam object pascal. Pada sesi pertama dalam materi ini akan lebih dikhususkan untuk pembahasan terkait function dan procedure yang nantinya akan digunakan dalam pemrograman berorientasi objek.

3.1. Deklarasi Class dan Object

Dalam melakukan pendeklarasian class di dalam Delphi, tidak jauh berbeda dengan pendeklarasian class-class yang sudah dimiliki oleh Delphi yang biasanya kita langsung gunakan class-class tersebut di dalam aplikasi yang kita buat seperti TForm, TButton, TEdit, TLabel, dan sebagainya. Berikut ini merupakan contoh deklarasi class dari class TForm beserta object dari class tersebut.



Sekarang, bagaimana bila kita akan mendeklarasikan class yang akan kita buat sendiri. Untuk strukturnya tidak jauh berbeda dengan contoh class TForm di atas. Misalkan kita akan membuat class BangunDatar beserta object dari class tersebut, maka deklarasinya adalah sebagai berikut.

```
type
  TBangunDatar = class
    sisi: integer;
    panjang: integer;
    lebar: integer;
    tinggi: integer;
    jari: integer;
  end;

var
  Persegi: TBangunDatar;
  PersegiPanjang: TBangunDatar;
  Segitiga: TBangunDatar;
  Lingkaran: TBangunDatar;
```

3.2. Access Modifier

Penggunaan class tidak terlepas dari yang namanya access modifier. Bagian ini yang akan melakukan enkapsulasi (pembatasan akses) pada class dan object yang telah kita buat. Beberapa access modifier yang terdapat dalam Delphi antara lain sebagai berikut.

- Public, merupakan access modifier yang dapat digunakan agar class tersebut dapat dipanggil atau digunakan oleh class-class yang lain.
- Private, merupakan access modifier yang dapat digunakan agar class tersebut hanya dapat digunakan oleh class itu sendiri.
- Protected, merupakan access modifier yang dapat digunakan agar class tersebut dapat digunakan oleh class itu sendiri dan turunan dari class tersebut.
- Published, merupakan access modifier yang penggunaannya hampir sama dengan public, namun hanya extra generate compiler RunTime Type Information (RTTI) yang tidak dapat digunakan oleh class lain.

3.3. Function dan Procedure

Function dan procedure merupakan sebuah method yang dapat digunakan untuk mendefinisikan aksi yang dapat dilakukan oleh class

tersebut. Perbedaan antara function dengan procedure adalah function akan menghasilkan nilai balik (result) dengan tipe data yang telah ditentukan, contoh penggunaannya lebih sering untuk mendefinisikan perhitungan fungsi matematis. Sedangkan procedure tidak memiliki nilai balik sehingga tidak tepat bila digunakan untuk mendefinisikan fungsi matematis, penggunaannya lebih sering untuk mendefinisikan syntax secara bebas tujuannya agar lebih ringkas pada penulisan syntax yang digunakan secara berulang.

Dalam function dan procedure boleh terdapat suatu parameter yang akan membentuk fungsi tersebut serta boleh juga tidak berparameter. Untuk lebih memperjelas dalam memahami penggunaan function dan procedure dapat dilihat pada contoh deklarasi di bawah ini.

```
type
  TBangunDatar = class
    sisi: integer;
    panjang: integer;
    lebar: integer;
    tinggi: integer;
    jari: integer;
    luas: integer;
  public
    function setHitungLuas(sisi1, sisi2: integer): integer;
    function getLuas: string;
    procedure tampilLuas(luas: string);
    procedure resetLuas;
  end;
```

Tipe data nilai balik

parameter

function berparameter

function tanpa parameter

procedure berparameter

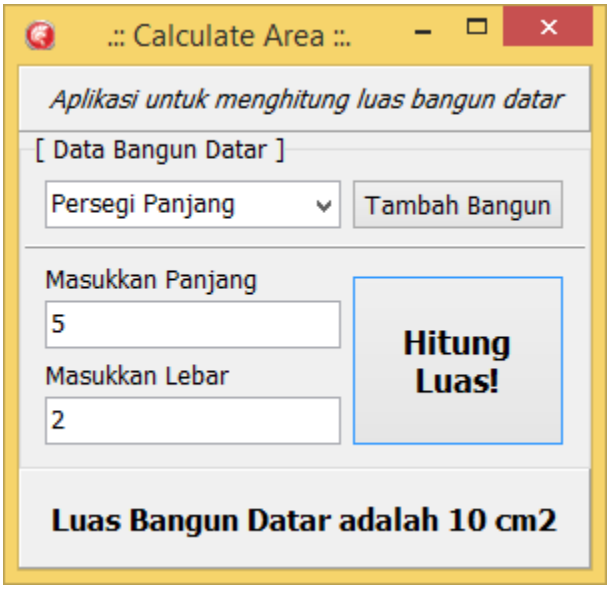
procedure tanpa parameter

Kemudian untuk pendefinisian dari function dan procedure di atas adalah seperti contoh di bawah ini.

nama class nama function

```
functio
begin
  luas
  resul
end;

functio
begin
  resul
end;

procedu
begin
  ShowM
end;

procedu
begin
  luas
end;
```

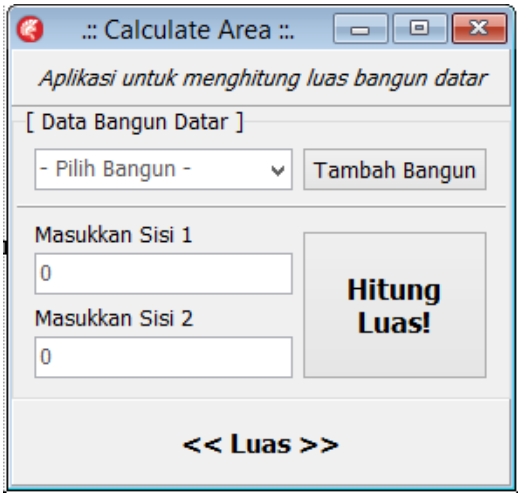
3.4. Creating Project: CalculateArea



Pada pertemuan ketiga ini, kita akan mencoba membuat sebuah aplikasi sederhana untuk menghitung luas bangun datar dengan

menerapkan pemrograman berorientasi objek. Untuk langkah-langkah pembuatannya adalah sebagai berikut.

1. Buat project baru dengan memilih VCL Forms Application.
2. Masukkan komponen-komponen ke dalam form serta aturlah agar terlihat seperti tampilan di bawah ini.



3. Kemudian atur juga properties untuk masing-masing komponen.

No.	Komponen	Properties
1.	Form1	BorderStyle = bsSingle Caption = .:: Calculate Area ::. Name = frmArea Position = poDesktopCenter
2.	Panel1	Align = alTop Caption = Aplikasi untuk menghitung luas bangun datar Font Style/fsItalic = True
3.	GroupBox1	Align = alTop Caption = [Data Bangun Datar]
4.	ComboBox1	Items = {Persegi, Persegi Panjang} Name = cbBangun TextHint = - Pilih Bangun -
5.	Button1	Caption = Tambah Bangun Cursor = crHandPoint Name = btTambah
6.	LabeledEdit1	EditLabel/Caption = Masukkan Sisi 1 Name = edSisi1 TextHint = 0
7.	LabeledEdit2	EditLabel/Caption = Masukkan Sisi 2 Name = edSisi2

		TextHint = 0
8.	Button2	Caption = Hitung Luas! Font/Size = 12 Font Style/fsBold = True Cursor = crHandPoint Name = btHitung
9.	Panel2	Align = alClient Caption = << LUAS >> Font/Size = 12 Font Style/fsBold = True

4. Selanjutnya double-click pada komponen btTambah, dan tambahkan syntax di bawah ini:

```
procedure TfrmArea.btTambahClick(Sender: TObject);
var bangun: string;
begin
  bangun := InputBox('Tambah Bangun Datar','Masukkan Bangun Datar
Baru','');
  if bangun.Length > 0 then cbBangun.Items.Add(bangun);
end;
```

5. Kemudian tambahkan juga syntax di bawah ini pada event OnChange komponen cbBangun. Syntax ini digunakan untuk mengubah property dari komponen TEdit untuk setiap bangun yang dipilih.

```
procedure TfrmArea.cbBangunChange(Sender: TObject);
begin
  case cbBangun.ItemIndex of
     0:
        begin
          edSisi1.EditLabel.Caption := 'Masukkan Sisi';
          edSisi2.Visible := false;
        end;
     1:
        begin
          edSisi1.EditLabel.Caption := 'Masukkan Panjang';
          edSisi2.Visible := true;
          edSisi2.EditLabel.Caption := 'Masukkan Lebar';
        end;
  end;
  edSisi1.SetFocus;
end;
```

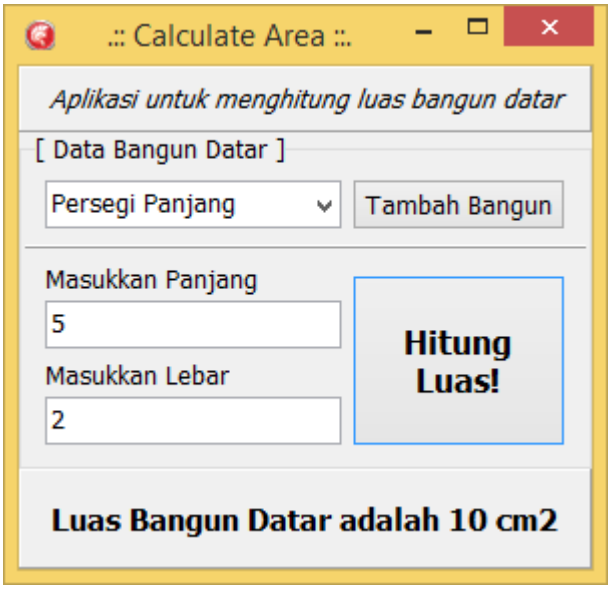
6. Double-click juga pada komponen btHitung, dan tambahkan syntax di bawah ini:

```
procedure TfrmArea.btHitungClick(Sender: TObject);
begin
  case cbBangun.ItemIndex of
     0:
        //penulisan syntax bisa dengan model seperti ini
        begin
          Persegi := TBangunDatar.Create;
          Persegi.sisi := StrToInt(edSisi1.Text);
          Persegi.setHitungLuas(Persegi.sisi, Persegi.sisi);
          pnLuas.Caption := 'Luas Bangun Datar adalah '+
Persegi.getLuas +' cm2';
          Persegi.tampilLuas(Persegi.getLuas);
          Persegi.resetLuas;
          Persegi.Free;
        end;
     1:
        //atau bisa juga menggunakan with - do
        with PersegiPanjang do
        begin
          PersegiPanjang := TBangunDatar.Create;
          panjang := StrToInt(edSisi1.Text);
          lebar := StrToInt(edSisi2.Text);
          setHitungLuas(panjang, lebar);
          pnLuas.Caption := 'Luas Bangun Datar adalah '+
PersegiPanjang.getLuas +' cm2';
          tampilLuas(PersegiPanjang.getLuas);
          resetLuas;
          PersegiPanjang.Free;
        end;
  end;
end;
```

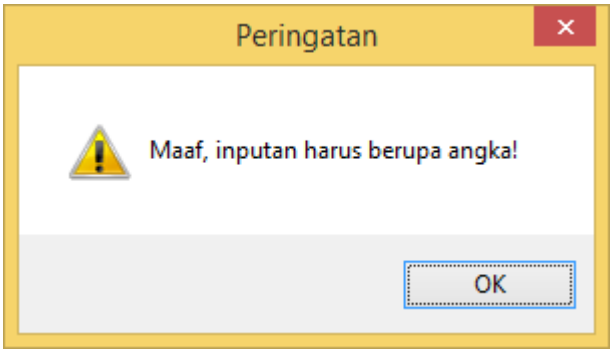
7. Tambahkan syntax lagi pada event OnKeypress komponen edSisi1. Syntax ini digunakan agar inputan hanya boleh diisi dengan angka.

```
procedure TfrmArea.edSisi1KeyPress(Sender: TObject; var Key: Char);
begin
  if not (key in['0'..'9',#13,#8]) then
    begin
      Application.MessageBox('Maaf, inputan harus berupa
angka!','Peringatan',mb_ok+MB_ICONWARNING);
      key := #0;
    end;
end;
```

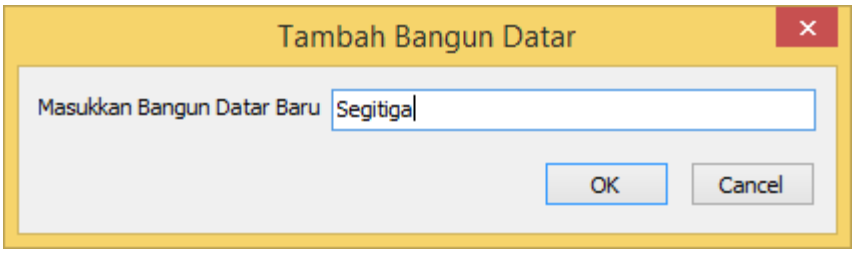
8. Terakhir, jalankan project yang telah dibuat dengan mengklik tombol Run Without Debugging atau tekan Ctrl+Shift+F9.
9. Setelah aplikasi berjalan, isikan data-datanya secara lengkap kemudian tekan tombol Hitung Luas!, maka akan muncul tampilan seperti di bawah ini.



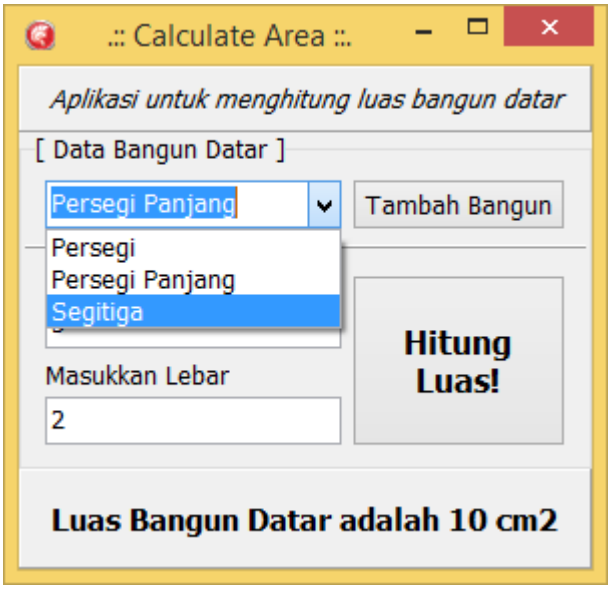
Sekarang kita coba untuk menginputkan selain angka di edit text, maka akan muncul pesan peringatan seperti di bawah ini. Hal tersebut dikarenakan di atas kita sudah tambahkan syntax untuk validasi inputan pada event OnKeypress TEdit.



Kemudian bila kita ingin menambahkan bangun baru bisa dengan mengklik tombol Tambah Bangun, maka akan muncul box inputan seperti di bawah ini dan isi bangun datar baru yang ingin ditambahkan, klik OK.



Maka dalam bagian pilihan bangun juga akan muncul bangun datar yang baru saja ditambahkan.



Pertemuan 4

OBJECT ORIENTED PROGRAMMING (INHERITANCE DAN POLYMORPHISM)

Dalam pertemuan kali keempat ini, akan dibahas mengenai konsep serta penerapan object oriented programming dalam object pascal. Pada sesi kedua dalam materi ini akan dibahas mengenai inheritance dan polymorphism yang merupakan bagian terpenting dalam pemrograman berorientasi objek.

4.1. Inheritance

Inheritance merupakan suatu keyword dalam pemrograman berorientasi objek yang berkaitan dengan pewarisan method dan atribut serta property lainnya yang dimiliki oleh class parent untuk diturunkan ke class turunannya. Dalam inheritance, setiap class-class turunan tersebut dapat mengakses atau menggunakan seluruh property yang dimilki parent-nya serta dapat juga memiliki property yang berbeda dari parent-nya. Object Pascal yang prinsipnya menggunakan pemrograman terstruktur (menggunakan procedure dan function) dapat juga menerapkan konsep pemrograman berorientasi objek ini, dan strukturnya pun tidak jauh berbeda dengan bahasa pemrograman lain yang menggunakan konsep OOP. Dalam Object Pascal, dikenal beberapa perintah yang dapat digunakan untuk menerapkan konsep inheritance ini, diantaranya:

a. Class, merupakan deklarasi tipe dari objek yang akan dibuat.
b. Constructor, merupakan deklarasi method dan property yang akan digunakan dalam objek class tersebut.
c. Destructor, merupakan deklarasi method yang dapat digunakan untuk menghilangkan (destroy) objek dari class.

d. Function, merupakan subroutine yang dapat menghasilkan nilai yang akan digunakan oleh objek untuk melakukan aksi tertentu.
e. Procedure, merupakan subroutine yang tidak menghasilkan nilai.
f. Overload, perintah yang mengizinkan beberapa subroutine dapat memiliki nama yang sama untuk satu objek yang sama.
g. Override, perintah yang digunakan untuk mengganti nilai atau menimpa nilai suatu method yang dimiliki oleh objek.

Secara umum, deklarasi inheritance dalam Object Pascal sebagai berikut:

```
type
  TBangun = class                                  → deklarasi class parent
  protected
    panjangBangun: byte;
    beratBangun: word;
    gayaBerat: word;
  published
    constructor Create(panjang: byte);             → constructor class parent
    procedure hitungBerat(massa: byte); virtual;
    function getBerat: word;
    procedure hitungGaya(gravitasi: byte);
    function getGaya: word;
  end;

  TBalok = class(TBangun)                          → deklarasi class turunan
  private
    lebarBalok: byte;
    tinggiBalok: byte;                             constructor class turunan
  published                                        ↗
    constructor Create(panjang, lebar, tinggi: byte);
    procedure hitungBerat(massa: byte); virtual;
  end;
```

Sedangkan struktur dalam memanggil constructor dari class parent adalah seperti ini:

```
1   Create;
    begin
      Inherited;  //dipanggil di awal constructor untuk meng-create
      ...
    end;
```

```
2   Create(arguments);
    begin
      Inherited Create(arguments);  //memanggil constructor parent
  disertai parameter
      ...
    end;

3   Destroy;
    begin
      ...
      Inherited;  //dipanggil di akhir constructor untuk men-destroy
    end;
```

4.2. Polymorphism

Polymorphism merupakan keyword yang digunakan untuk membuat suatu method atau property yang dimiliki object dapat memiliki banyak bentuk (dalam hal ini adalah nilai dari method atau property objek) yang diterapkan pada class objek yang berbeda. Dengan adanya polymorphism ini membuat satu method atau property yang sama yang dimiliki class parent dapat digunakan kembali oleh class-class turunannya dengan nilai yang berbeda-beda sesuai karakteristik objek tersebut. Konsep polymorphism ini adalah dengan menimpa nilai (override) property pada class parent yang akan digantikan dengan nilai dari class turunannya.

Deklarasi secara umum polymorpism dalam Object Pascal adalah sebagai berikut:

```
type
   TBangun = class
   protected
     panjangBangun: byte;
     beratBangun: word;
     gayaBerat: word;
   published
     constructor Create(panjang: byte);
     procedure hitungBerat(massa: byte); virtual;
     function getBerat: word;
     procedure hitungGaya(gravitasi: byte);
     function getGaya: word;
   end;

   TBalok = class(TBangun)
   private
     lebarBalok: byte;
     tinggiBalok: byte;
   published
     constructor Create(panjang, lebar, tinggi: byte);
     procedure hitungBerat(massa: byte); virtual;
   end;
```

method dengan polymorphism

Implementasi dari polymorphism di atas seperti ini:

```
procedure TBangun.hitungBerat(massa: byte);
begin
   beratBangun := massa * (4 * panjangBangun * panjangBangun);
end;

function TBangun.getBerat: word;
begin
   result := beratBangun;
end;

procedure TBalok.hitungBerat(massa: byte);
begin
   beratBangun := massa * panjangBangun * lebarBalok * tinggiBalok;
end;
```

nilai dari class parent

fungsi pemanggilan

nilai dari class turunan

Terlihat bahwa property beratBangun yang dimiliki oleh class parent juga digunakan oleh class turunannya, namun yang membedakan adalah nilai yang akan dihasilkan dari kedua class

tersebut berbeda antara beratBangun dari parent dan beratBangun dari turunannya.

4.3. Creating Project: Gaya Berat

Pada pertemuan keempat ini, kita akan mencoba membuat sebuah aplikasi sederhana untuk menghitung gaya berat bangun dengan menerapkan konsep inheritance dan polymorphism dalam pemrograman berorientasi objek. Untuk langkah-langkah pembuatannya adalah sebagai berikut.

1. Buat project baru dengan memilih VCL Forms Application.
2. Masukkan komponen-komponen ke dalam form serta aturlah agar terlihat seperti tampilan di bawah ini.

3. Kemudian atur juga properties untuk masing-masing komponen.

No.	Komponen	Properties

1.	Form1	BorderStyle = bsSingle Caption = .:: Gaya Berat ::. Name = frmGaya Position = poDesktopCenter
2.	Panel1	Align = alTop Caption = Aplikasi untuk menghitung gaya berat bangun Font Style/fsItalic = True
3.	GroupBox1	Align = alTop Caption = [Data Bangun]
4.	ComboBox1	Items = {Kubus, Balok} Name = cbBangun TextHint = - Pilih Bangun -
5.	Button1	Caption = Tambah Bangun Cursor = crHandPoint Name = btTambah
6.	LabeledEdit1	EditLabel/Caption = Berat per Kubik (Kg) Name = edBerat TextHint = 0
7.	LabeledEdit2	EditLabel/Caption = Berat Bangun (Kg) Name = edBeratBangun ReadOnly = true TextHint = 0
8.	LabeledEdit3	EditLabel/Caption = Gravitasi (m/s2) Name = edGravitasi TextHint = 0
9.	LabeledEdit4	EditLabel/Caption = Masukkan Sisi 1 Name = edSisi1 TextHint = 0
10.	LabeledEdit5	EditLabel/Caption = Masukkan Sisi 2 Name = edSisi2 TextHint = 0
11.	LabeledEdit6	EditLabel/Caption = Masukkan Sisi 3 Name = edSisi3 TextHint = 0
12.	Button2	Caption = Hitung Gaya! Font/Size = 12 Font Style/fsBold = True Cursor = crHandPoint Name = btHitung

13.	Panel2	Align = alClient Caption = << Gaya Berat >> Font/Size = 12 Font Style/fsBold = True

4. Tambahkan syntax-syntax ini untuk deklarasi dari class parent dan turunannya di bawah bagian implementation.

```
constructor TBangun.Create(panjang: byte);
begin
   panjangBangun  := panjang;
   beratBangun := 0;
   gayaBerat := 0;
end;

procedure TBangun.hitungBerat(massa: byte);
begin
   beratBangun := massa * (4 * panjangBangun * panjangBangun);
end;

function TBangun.getBerat: word;
begin
   result := beratBangun;
end;

procedure TBalok.hitungBerat(massa: byte);
begin
   beratBangun := massa * panjangBangun * lebarBalok * tinggiBalok;
end;

procedure TBangun.hitungGaya(gravitasi: byte);
begin
   gayaBerat := beratBangun * gravitasi;
end;

function TBangun.getGaya: word;
begin
   result := gayaBerat;
end;

constructor TBalok.Create(panjang: Byte; lebar: Byte; tinggi:
Byte);
begin
   inherited Create(panjang);
   lebarBalok := lebar;
   tinggiBalok := tinggi;
end;
```

5. Selanjutnya double-click pada komponen btTambah, dan tambahkan syntax di bawah ini:

```
procedure TfrmArea.btTambahClick(Sender: TObject);
var bangun: string;
begin
  bangun := InputBox('Tambah Bangun','Masukkan Bangun Baru','');
  if bangun.Length > 0 then cbBangun.Items.Add(bangun);
end;
```

6. Kemudian tambahkan juga syntax di bawah ini pada event OnChange komponen cbBangun. Syntax ini digunakan untuk mengubah property dari komponen TEdit untuk setiap bangun yang dipilih.

```
procedure TfrmArea.cbBangunChange(Sender: TObject);
begin
  case cbBangun.ItemIndex of
    0:
      begin
        edSisi1.EditLabel.Caption := 'Masukkan Sisi';
        edSisi2.Visible := false;
        edSisi3.Visible := false;
      end;
    1:
      begin
        edSisi1.EditLabel.Caption := 'Masukkan Panjang';
        edSisi2.EditLabel.Caption := 'Masukkan Lebar';
        edSisi3.EditLabel.Caption := 'Masukkan Tinggi';
        edSisi2.Visible := true;
        edSisi3.Visible := true;
      end;
  end;
end;
```

7. Double-click juga pada komponen btHitung, dan tambahkan syntax di bawah ini:

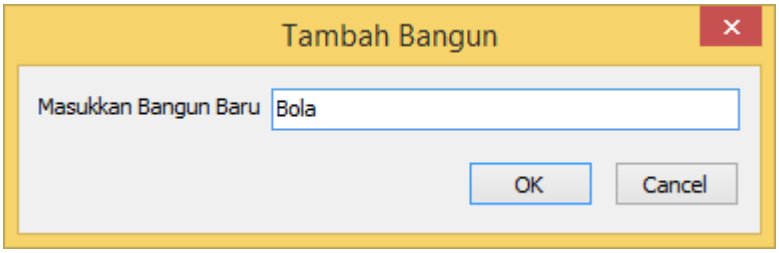
```
procedure TfrmArea.btHitungClick(Sender: TObject);
begin
var
   kubus: TBangun;
   balok: TBalok;
begin
  case cbBangun.ItemIndex of
     0:
       begin
         kubus := TBangun.Create(StrToInt(edSisi1.Text));
         kubus.hitungBerat(StrToInt(edBerat.Text));
         edBeratBangun.Text := IntToStr(kubus.getBerat);

         kubus.hitungGaya(StrToInt(edGravitasi.Text));
         pnGaya.Caption := 'Gaya Berat bangun tersebut adalah
'+IntToStr(kubus.getGaya)+' N';
       end;
     1:
       begin
         balok := TBalok.Create(StrToInt(edSisi1.Text),
StrToInt(edSisi2.Text), StrToInt(edSisi3.Text));
         balok.hitungBerat(StrToInt(edBerat.Text));
         edBeratBangun.Text := IntToStr(balok.getBerat);

         balok.hitungGaya(StrToInt(edGravitasi.Text));
         pnGaya.Caption := 'Gaya Berat bangun tersebut adalah
'+IntToStr(balok.getGaya)+' N';
       end;
  end;
end;
```

8. Tambahkan syntax lagi pada event OnKeypress komponen edSisi1. Syntax ini digunakan agar inputan hanya boleh diisi dengan angka.

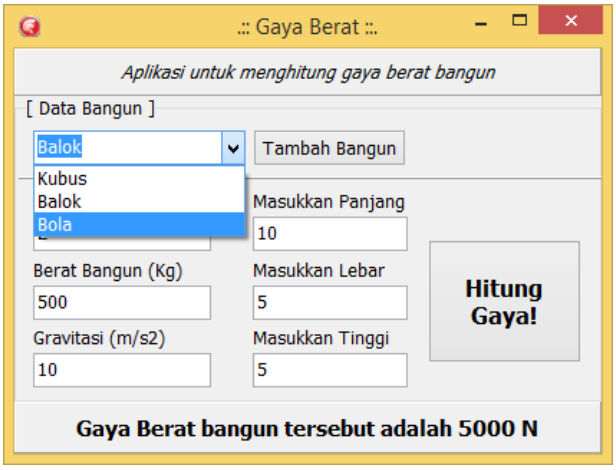
```
procedure TfrmArea.edSisi1KeyPress(Sender: TObject; var Key: Char);
begin
  if not (key in['0'..'9',#13,#8]) then
    begin
      Application.MessageBox('Maaf, inputan harus berupa
angka!','Peringatan',mb_ok+MB_ICONWARNING);
      key := #0;
    end;
end;
```

9. Terakhir, jalankan project yang telah dibuat dengan mengklik tombol Run Without Debugging atau tekan Ctrl+Shift+F9.
10. Setelah aplikasi berjalan, isikan data-datanya secara lengkap kemudian tekan tombol Hitung Gaya!, maka akan muncul tampilan seperti di bawah ini.

Kemudian bila kita ingin menambahkan bangun baru bisa dengan mengklik tombol Tambah Bangun, maka akan muncul box inputan seperti di bawah ini dan isi bangun baru yang ingin ditambahkan, klik OK.



Maka dalam bagian pilihan bangun juga akan muncul bangun yang baru saja ditambahkan.


.:: Gaya Berat ::.
Aplikasi untuk menghitung gaya berat bangun
[Data Bangun]
Balok
Tambah Bangun
Kubus
Balok
Bola
Masukkan Panjang
10
Berat Bangun (Kg)
500
Masukkan Lebar
5
Hitung Gaya!
Gravitasi (m/s2)
10
Masukkan Tinggi
5
Gaya Berat bangun tersebut adalah 5000 N

Pertemuan 5

KOMUNIKASI AGENT DENGAN JADE

Sebuah karakteristik dasar sistem multi-agent adalah bahwa agent berkomunikasi dan berinteraksi. Hal ini dilakukan melalui pertukaran pesan dan, untuk memahami satu sama lain, sangat penting bahwa agent setuju pada format dan semantik pesan ini. Jade mengikuti standar FIPA sehingga idealnya agent Jade bisa berinteraksi dengan agent yang ditulis dalam bahasa lain dan berjalan pada platform lainnya.

Ada banyak bagian tambahan untuk pesan selain konten, misalnya: penerima yang dituju, pengirim dan jenis pesan. Hal ini penting untuk pesan secara keseluruhan dapat menghormati format umum. Di Jade, pesan mengikuti aturan standar ACL (Agent Communication Language). Secara khusus, Jade mendukung SL (Semantic Language) dari FIPA, pengkodean LISP seperti konsep, tindakan dan predikat.

5.1. Struktur Pesan Jade

Berikut ini adalah daftar atribut dari pesan Jade ACL. Seperti yang dijelaskan dalam dokumentasi API, Jade menyediakan dan menetapkan metode untuk mengakses semua atribut.

- Performative – jenis pesan FIPA (INFORM, QUERY, PROPOSE, ...)
- Addressing (pengirim dan penerima)
- Content – merupakan konten utama pesan
- ConversationID – digunakan untuk menghubungkan pesan dalam percakapan yang sama
- Language – menentukan bahasa yang digunakan dalam konten
- Ontology – menetukan ontologi yang digunakan dalam konten
- Protocol – menetukan protokol
- ReplyWith – field lain untuk membantu membedakan jawaban

- InReplyTo – pengirim digunakan untuk membantu membedakan jawaban
- ReplyBy – digunakan untuk mengatur batas waktu jawaban

Ketika kita membuat pesan, terlebih dulu kita harus menentukan jenisnya dan kemudian mengatur konten pesannya, seperti ini.

```
ACLMessage msg = new ACLMessage( ACLMessage.INFORM );
    msg.setContent("Saya akan datang ke rumahmu." );
```

Pesan tersebut menggunakan performatif yang paling umum: INFORM dimana satu agen memberikan beberapa informasi lain yang berguna. Jenis lain adalah: QUERY untuk mengajukan pertanyaan, REQUEST untuk meminta yang lain untuk melakukan sesuatu dan USULAN untuk memulai perundingan. Performatif jawaban mencakup SETUJU atau MENOLAK.

5.2. Mengirim dan Menerima Pesan

Asumsikan kita sudah memiliki agent ID (misalkan "AG001") sebagai penerima pesan, untuk mengiriman pesan dapat menggunakan perintah seperti ini.

```
AID dest = "AG001"
msg.addReceiver( dest );
send(msg);
```

Catatan: Dalam perintah di atas menggunakan `addReceiver()` karena tidak ada method `setReceiver()`. Salah satu alasannya adalah kita bisa menambahkan beberapa penerima pesan dan salah satu mengirim pesan broadcast kepada semua penerima.

Pesan akan diarahkan kemanapun agent tujuan berada dan akan ditempatkan dalam antrian pesannya. Ada 2 cara agar penerima mendapatkan pesan, yang pertama dengan menggunakan `blockingReceive()`, semua aktivitas penerimaan agent akan ditunda sampai pesan tiba.

```
ACLMessage msg = blockingReceive();
```

Cara yang kedua adalah dengan menggunakan `receive()`, memeriksan antrian pesan, mengembalikan pesan jika ada satu atau sama sekali tidak ada. Cara ini merupakan teknik biasa yang digunakan ketika suatu agent terlibat di aktivitas parallel dan memiliki banyak Behaviour yang aktif.

```
ACLMessage msg = receive();
if (msg != null)
  // handle pesan
else
  //lakukan aksi tertentu
```

Method ini memiliki varian dengan argumen tambahan untuk mengubah behaviour. Dengan `blockingReceive()`, kita dapat menambahkan objek pola untuk memilih jenis pesan yang ingin kita terima. Kita juga dapat menentukan batas waktu akhir. Jika batas waktu berakhir sebelum pesan yang diinginkan tiba, method akan mengembalikan nilai null.

Dengan `receive()`, method tidak pernah memblok, hanya selector argumen pola yang akan diterima. Di bawah ini contoh kode untuk mencetak salinan dari semua pesan yang diterima.

```
public class Receiver extends Agent {
   protected void setup() {
      addBehaviour(new CyclicBehaviour(this) {
          public void action() {
             ACLMessage msg = receive();
             if (msg!=null)
                System.out.println("-"+myAgent.getLocalName()+"<-"+
                                   msg.getContent());
             block();
          }
      });
   }
}
```

Perhatikan penggunaan `block()` tanpa timeout. Ini berarti akan menahan behaviour sampai pesan berikutnya diterima, pada saat semua behaviour menunggu pesan atau berlalunya waktu (dengan `block(dt)`) akan diaktifkan dan method aksinya dipanggil satu per

satu. Jika kita tidak memanggil `block()`, maka behaviour akan terus aktif dan berulang-ulang. Umumnya semua method harus diakhiri dengan memanggil `block()`.

5.3. Menjawab Pesan

Semua pesan memiliki sebuah atribut yang berisi ID dari pengirim. Dengan demikian kita dapat menjawab pesan sebagai

```
ACLMessage msg = receive();
ACLMessage reply = new ACLMessage( ACLMessage.INFORM );
reply.setContent( "Pong" );
reply.addReceiver( msg.getSender() );
send(reply);
```

Untuk mempermudah dalam menjawab pesan, Jade menyediakan method `createReply()` yang akan menciptakan pesan baru dengan atribut pengirim dan penerima diaktifkan dan semua atribut lainnya diatur dengan benar. Umumnya, hanya konten dan performatif yang harus diubah sebelum mengirimnya kembali. Di bawah ini contoh untuk memodifikasi agent penerima yang akan menjawab semua pesan dengan "Pong":

```
public void action() {
  ACLMessage msg = receive();
  if (msg!=null) {
    System.out.println("-"+myAgent.getLocalName()+"<-"+
                       msg.getContent());
    ACLMessage reply = msg.createReply();
    reply.setPerformative( ACLMessage.INFORM );
    reply.setContent("Pong");
    reply.send();
  }
  block();
}
```

Dari contoh di atas, tidak ada banyak keuntungan dalam menggunakan `createReply()`, manfaat yang sebenarnya adalah saat diterapkan di dalam aplikasi ketika atribut-atribut lain seperti conversationID atau ontology harus sesuai dengan pesan asli.

5.4. Mencari Agent untuk Berkomunikasi

Sekarang, kita kembali ke masalah bagaimana untuk mencari agent lain dan mendapatkan AID sehingga kita dapat mengirim pesan. Kita sudah melihat bahwa ketika kita menerima pesan, kita bisa mendapatkan AID pengirim, dengan `createReply()`, kita bahkan tidak memerlukan ID pengirim.

Kita juga dapat membuat agent ID dari nama lokal agent, misalnya satu yang ditentukan pada baris perintah ketika memulai sebuah container. Hal ini sangat berguna ketika nama tetap untuk agent tertentu telah diputuskan sebelumnya. Asumsikan bahwa kita memiliki aplikasi dengan pembeli yang perlu berinteraksi dengan dua pemasok yang disebut "store1" dan "store2" dan program ini akan dimulai sebagai berikut:

```
% java jade.Boot fred:BuyerAgent
store1:SellerAgent  store2:SellerAgen
```

Kemudian dalam agent pembeli dapat mem-broadcast pesan ke pemasok. Perhatikan constructor AID yang mengambil kedua local name dan konstanta ISLOCALNAME.

```
ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
msg.setContent("Pong");
for (int i = 1; i<=2; i++)
  msg.addReceiver( new AID( "store" + i, AID.ISLOCALNAME) );
send(msg);
```

Pertemuan 6

KOMUNIKASI ANTAR AGENT

6.1. Pengiriman Pesan

Berikut ini adalah method dan konstraktor dasar pada pengiriman pesan.

Method / Procedure	Deskripsi
`ACLMessage msg = new ACLMessage(ACLMessage.INFORM) thows jade.lang.acl.ACLMessage`	Memanggil Class Object ACLMessage untuk tempat dari pesan yang akan dikirim. Nama dari Class Object ACLMessage yaitu "msg"
`Public void setContent() throws jade.lang.acl.ACLMessage`	Memanggil Method dari kelas ACLMessage untuk pengisian pesan
`Public void addReceiver() throws jade.lang.acl.ACLMessage`	Method yang digunakan untuk menambah nama agent sebagai penerima pesan
`Public void send() throws jade.lang.acl.ACLMessage`	Mengirimkan pesan sesuai dengan agent tujuan

Langkah-langkah untuk menciptakan kelas pengirim pesan sederhana:

1. Menambahkan behavior untuk pengiriman. Untuk contoh pengiriman menggunakan TickerBehavior:
   ```
   addBehaviour(new TB(this, 1000));
   ```
2. Membuat kelas object ACLMessage:
   ```
   ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
   ```
3. Membuat method untuk agent tujuan, sebagai contoh agent tujuannya "andi":

```
msg.addReceiver(new AID("andi", AID.ISLOCALNAME));
```

4. Membuat method untuk pengisian pesan:

```
msg.setContent("#" + myAgent.getLocalName() + "# ");
```

5. Memanggil method untuk pengiriman pesan:

```
myAgent.send(msg);
```

Untuk mengilustrasikan langkah-langkah di atas perhatikan contoh program berikut untuk mengirim pesan:

```
import jade.core.AID;
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.lang.acl.*;

public class ngirim extends Agent {
  protected void setup() {
    addBehaviour(new TB(this, 1000));
  }
}

class TB extends TickerBehaviour {
  TB(Agent a, int i) {
    super(a, i);
  }
  @Override

  protected void onTick() {
    System.out.println("NC Kirim Matriks : " +
myAgent.getLocalName());
    ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
    msg.setContent("#" + myAgent.getLocalName() + "# ");
    msg.addReceiver(new AID("keterima", AID.ISLOCALNAME));
    myAgent.send(msg);
  }
}
```

6.2. Penerimaan Pesan

Berikut ini adalah method dan konstraktor dasar pada penerimaan pesan.

Method / Procedure	Deskripsi
`Public void receive()` `throws`	Method untuk penerima kiriman

jade.lang.acl.ACLMessage	
Public void getContent() throws jade.lang.acl.ACLMessage	Method untuk membaca isi dari pesan yang didapat
Public void createReply() throws jade.lang.acl.ACLMessage	Method untuk membalas kiriman pesan secara otomatis
ACLMessage msg =new ACLMessage() throws jade.lang.acl.ACLMessage	Memanggil Class Object ACLMessage untuk memanggil method yang digunakan untuk penerimaan dan membalas pesan otomatis

Langkah-langkah untuk menciptakan kelas penerimaan sederhana:

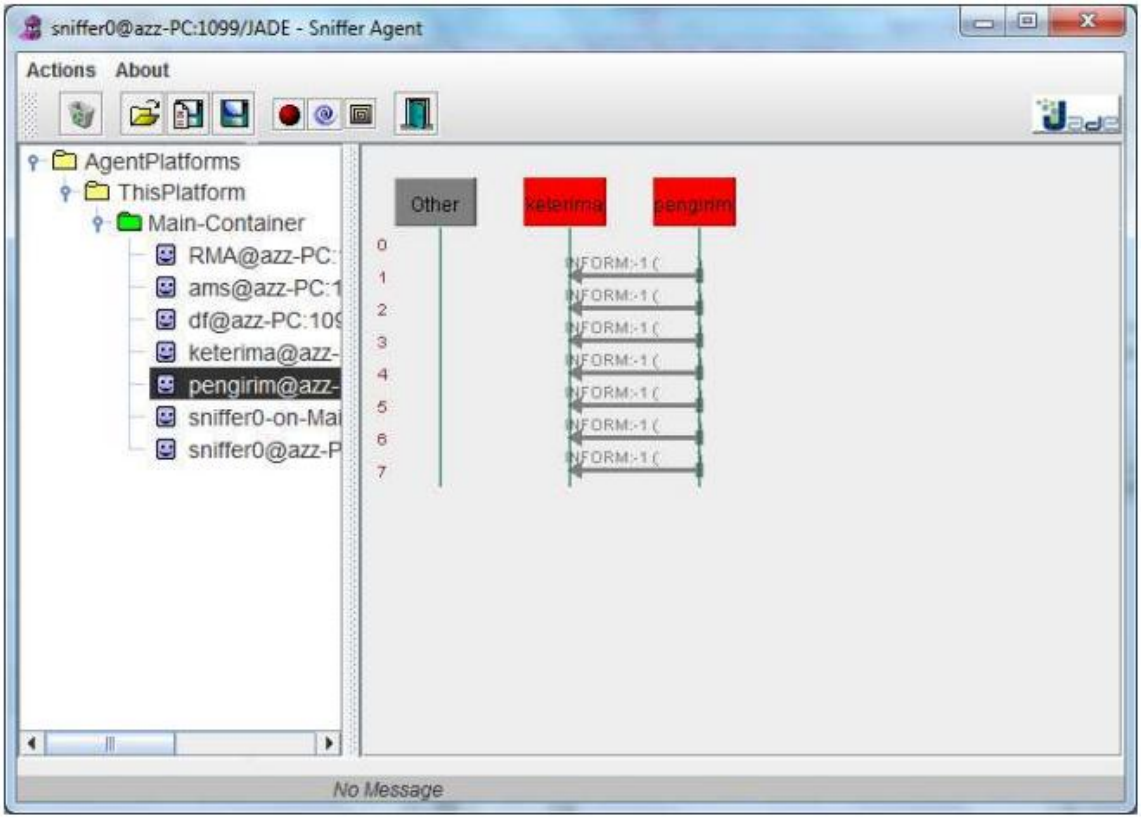
1. Menambahkan behavior untuk penerimaan. Untuk contoh penerimaan menggunakan CyclicBehavior:
   ```
   addBehaviour (new Nm_Bhvior(this));
   ```
2. Membuat kelas object ACLMessage dan memanggil method untuk penerimaan:
   ```
   ACLMessage msg = myAgent.receive();
   ```
3. Membuat method untuk pembacaan isi pesan dan simpan dalam variable String:
   ```
   String title = msg.getContent();
   ```

Untuk mengilustrasikan langkah-langkah di atas perhatikan contoh program berikut untuk menerima pesan:

```
import jade.core.Agent;
import jade.core.behaviours.*;
import jade.core.AID;
import jade.lang.acl.*;

public class terima extends Agent {
  @Override
  protected void setup() {
    addBehaviour (new Nm_Bhvior(this));
  }
}
```

```
class Nm_Bhvior extends CyclicBehaviour {
  public Nm_Bhvior(Agent a) {
    super(a);
  }

  public void action () {
    ACLMessage msg = myAgent.receive();
    if (msg!=null) {
      String title = msg.getContent();
      System.out.println("NM terima pesan "+title);
    }
  }
}
```

Ketika kode program di atas dijalankan dan tampilan sniff diaktifkan, maka akan tampak antara agent pengirim melakukan pengiriman informasi secara terus menerus dan diterima oleh agent keterima. Selanjutnya untuk menjalankan sniff dilakukan dengan "do sniff this agent" maka untuk menghentikannya pun hanya tinggal klik kanan pada agent lalu "do not sniff this agent". Berikut gambaran tampilan sniff:


sniffer0@azz-PC:1099/JADE - Sniffer Agent
Actions
About
AgentPlatforms
ThisPlatform
Main-Container
RMA@azz-PC:
ams@azz-PC:1
df@azz-PC:109
keterima@azz-
pengirim@azz-
sniffer0-on-Mai
sniffer0@azz-P
Other
keterima
pengirim
0
1
2
3
4
5
6
7
No Message

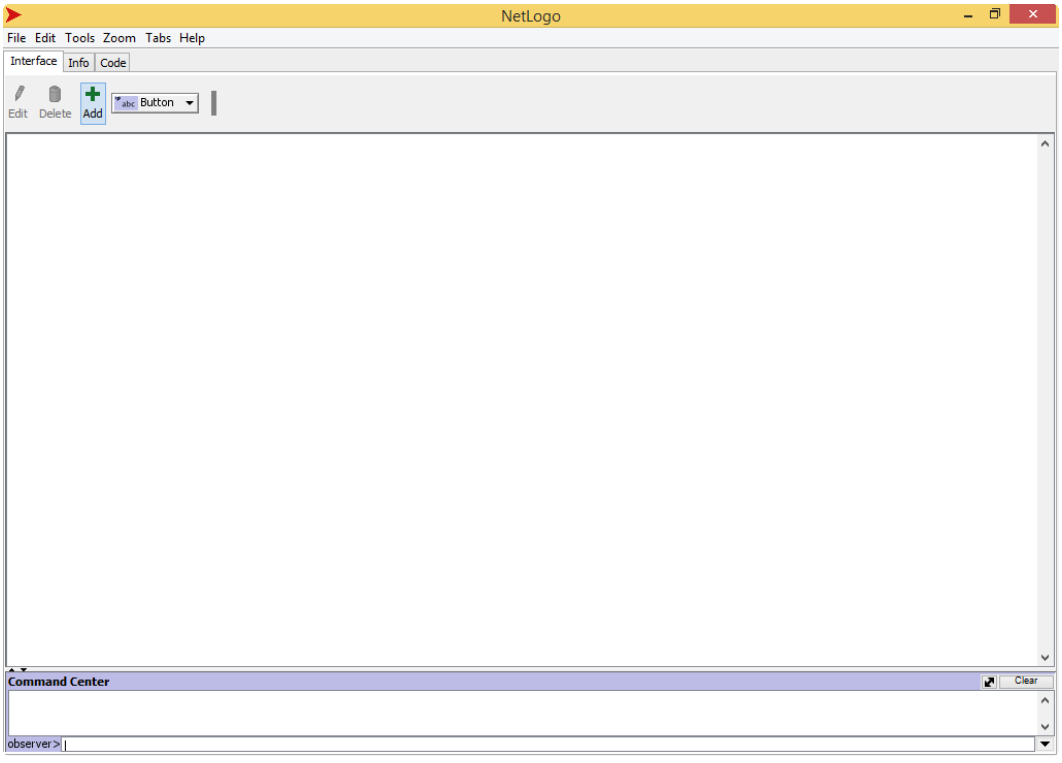
Pertemuan 7

PEMODELAN MULTI-AGENT DENGAN NETLOGO

7.1. Pengantar NetLogo

NetLogo merupakan model pemrograman untuk mensimulasikan suatu keadaan alam dan fenomena sosial yang terjadi. NetLogo merupakan bagian dari aplikasi untuk pengembangan sistem pemodelan yang kompleks. Pemodel dapat memberikan instruksi kepada seratus maupun seribu agent di semua operasi secara independen. Hal ini membuat NetLogo memungkinkan untuk memperluas koneksi antara individu dengan tingkah laku level mikro dengan pola level makro yang timbul dari interaksi mereka.

NetLogo merupakan generasi lanjutan dari bahasa pemodelan sistem multi-agent. NetLogo berjalan pada Java Virtual Machine, sehingga dapat berjalan pada semua platform yang tersedia. Berikut ini merupakan tampilan utama dari NetLogo.



7.2. Pemrograman NetLogo

1. Agent

   NetLogo dapat menciptakan suatu agent yang dapat mengikuti instruksi yang diberikan. Dalam NetLogo terdapat 4 tipe dari agent, yaitu: turtles, patches, links, dan observer.

   Turtles merupakan agent yang dapat berpindah berkeliling dunia. Dunia yang dimaksud adalah dua dimesi dan terbagi menjadi grid dari patches. Setiap patch merupakan kotak potongan dari tanah melewati turtles yang dapat berpindah. Links merupakan agent yang menghubungkan 2 turtles. Observer tidak memiliki lokasi, kita hanya dapat menggambarkannya seperti objek yang terlihat di atas dunia dari turtles dan patches. Observer tidak mengamati dengan pasif, tetapi dapat memberikan instruksi kepada agent lainnya.

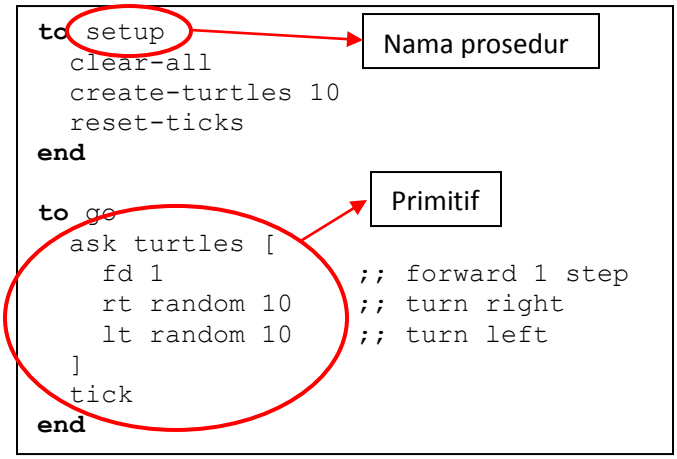
   Pada saat NetLogo berjalan, tidak ada turtles yang tercipta. Observer dan patches dapat menciptakan turtles baru. Patches memiliki koordinat yaitu pada titik (0,0) yang disebut dengan origin dan titik koordinat dari patches lain dapat berupa arah horizontal (pxcor) dan vertikal (pycor). Turtles juga memiliki titik koordinat yaitu xcor dan ycor. Koordinat dari patch selalu bernilai integer, namun titik koordinat turtles dapat bernilai desimal. Hal ini menunjukkan bahwa turtles dapat diposisikan pada sembarang titik. Links tidak memiliki titik koordinat. Setiap link memiliki 2 titik akhir, setiap titik akhir berupa turtles.

2. Procedure

   Dalam NetLogo, command dan reporter menjelaskan apa yang agent kerjakan. Command adalah aksi untuk suatu agent agar

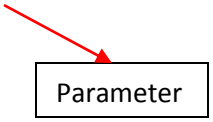
dapat menghasilkan beberapa efek. Reporter adalah instruksi untuk melakukan komputasi pada suatu nilai.

Pada dasarnya, nama command diawali dengan kata kerja seperti "create", "die", "jump", "inspect", ataupun "clear". Banyak nama reporter berupa benda atau frase benda. Command dan reporter di dalam NetLogo disebut dengan primitif. Command dan reporter yang mendefinisikan dirinya disebut dengan prosedur. Setiap prosedur memiliki nama dan diawali dengan kata "to" atau "to-report". Kata "end" menandakan akhir dari command di dalam prosedur. Berikut ini merupakan contoh dari prosedur (tanpa parameter).




```
to draw-polygon [num-sides len]  ;; turtle procedure
  pen-down
  repeat num-sides [
    fd len
    rt 360 / num-sides
  ]
end
```

Untuk prosedur yang memiliki parameter seperti di bawah ini.



Prosedur reporter diawali dengan kata "to-report" dan di dalam blok prosedur terdapat perintah "report" untuk melaporkan nilai yang ingin dilaporkan.

```
to-report absolute-value [number]
  ifelse number >= 0
    [ report number ]
    [ report (- number) ]
end
```

3. Variabel
   - Agent variabel

     Agent variabel ditempatkan pada nilai penyimpanan (seperti nomor) pada agent. Suatu agent variabel dapat menjadi variabel global, variabel turtle, variabel patch, atau variabel link. Jika variabel tersebut adalah variabel global, hanya ada satu nilai untuk sebuah variabel dan setiap agent dapat mengaksesnya.

     Variabel turtle, patch, dan link berbeda-beda. Setiap turtle memiliki nilainya sendiri untuk setiap variabel turtle, berlaku juga untuk patch dan link. Sebagai contoh, semua turtle dan link memiliki variabel "color" dan semua patch memiliki variabel "pcolor". Kita juga dapat mendefinisikan variabel sesuai keinginan sendiri. Kita juga dapat membuat variabel global dengan menambahkan switch, slider, chooser, atau input box ke dalam model yang dibuat, atau dengan menggunakan kata "globals" di awal dari code, seperti ini.

```
globals [score]
```

Kita juga dapat mendefinisikan variabel turtle, patch, dan link baru menggunakan kata "turtles-own", "patches-own", dan "links-own", seperti ini.

```
turtles-own [energy speed]
patches-own [friction]
links-own [strength]
```

Pada kondisi lain dimana kita ingin suatu agent dapat membaca variabel agent yang berbeda, maka dapat menggunakan kata "of", seperti ini.

```
show [color] of turtle 5
```

- Local variabel

  Local variabel didefinisikan dan hanya digunakan di dalam konteks dari sebagian prosedur atau bagian dari prosedur. Untuk membuat local variabel, gunakan perintah "let". Jika menggunakan "let" di atas prosedur, variabel akan berada di seluruh prosedur. Jika menggunakannya di dalam kumpulan kurung kotak, sebagai contoh di dalam kata "ask", seperti ini.

```
to swap-colors [turtle1 turtle2]
  let temp [color] of turtle1
  ask turtle1 [ set color [color] of turtle2 ]
  ask turtle2 [ set color temp ]
end
```

4. Tick Counter

   NetLogo menyertakan tick counter jadi kita dapat mengatur track seberapa banyak tick yang telah lewat. Di dalam code, untuk mengembalikan nilai saat ini dari tick counter, menggunakan "ticks" reporter. Perintah "tick" akan menaikkan tick counternya sebanyak 1 dan ditempatkan di akhir go procedure. Perintah "clear-all" akan membersihkan tick counter. Gunakan perintah "reset-ticks" ketika

model telah selesai diatur dan akan memulai kembali tick counternya. Perintah tersebut berada di akhir setup procedure.

```
to setup
  clear-all
  create-turtles 10
  reset-ticks
end

to go
  ask turtles [ fd 1 ]
  tick
end
```

5. Color
   - Color primitive

     Terdapat beberapa primitive yang membantu dalam hal yang berhubungan dengan warna, yang disebut dengan "wrap-color" primitive. "scale-color" primitive membantu dalam mengkonversi data numerik ke dalam warna.
   - RGB and RGBA color

     NetLogo juga merepresentasikan warna sebagai RGB (Red/Green/Blue) dan RGBA (Red/Green/Blue/Alpha). Saat menggunakan warna RGB, range warna akan tersedia secara penuh (0-255). Untuk warna RGBA kita dapat menggunakan semua warna RGB ditambah dengan warna transparan. Untuk menset variabel color dalam NetLogo menggunakan "color" untuk turtles dan link, serta "pclolor" untuk patch, seperti ini.

```
set pcolor [255 0 0]
```

6. Ask

NetLogo menggunakan perintah “ask” untuk memberikan perintah kepada turtle, patch, dan link. Semua code dapat dijalankan oleh turtle dan harus diletakkan pada turtle “context”. Kita dapat membuat turtle context menggunakan 3 langkah, yaitu:

- Pada button, dengan memilih “Turtles” dari menu popup. Beberapa code diletakkan di dalam button yang akan dijalankan oleh semua turtle.
- Pada command center, dengan memilih “Turtles” dari menu popup.
- Dengan menggunakan “ask turtles”, “hatch”, atau perintah lain yang dibentuk turtle context.

Berlaku juga untuk patch, link, dan observer. Berikut ini merupakan c

```
to setup
  clear-all
  create-turtles 100   ;; create 100 turtles with random headings
  ask turtles
    [ set color red    ;; turn them red
      fd 50 ]          ;; spread them around
  ask patches
    [ if pxcor > 0         ;; patches on the right side
        [ set pcolor green ] ]  ;; of the view turn green
  reset-ticks
end
```

p

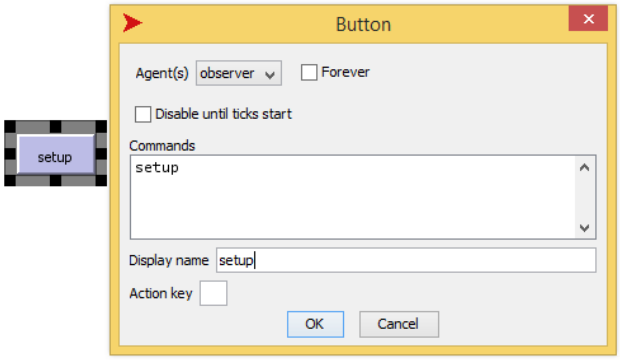
enggunaan “ask” di prosedur NetLogo:

Pertemuan 8

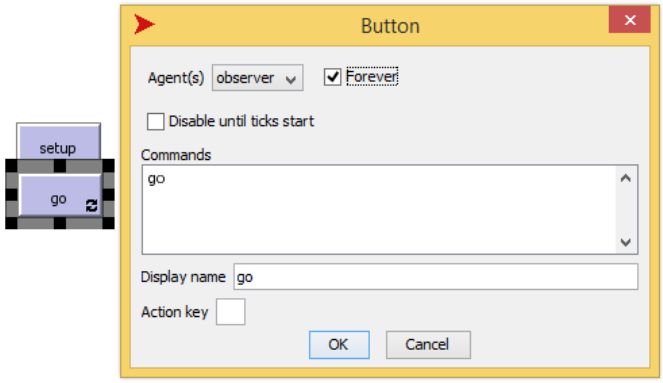
MULTI-AGENT: HATCH EXAMPLE

8.1. Perancangan Interface

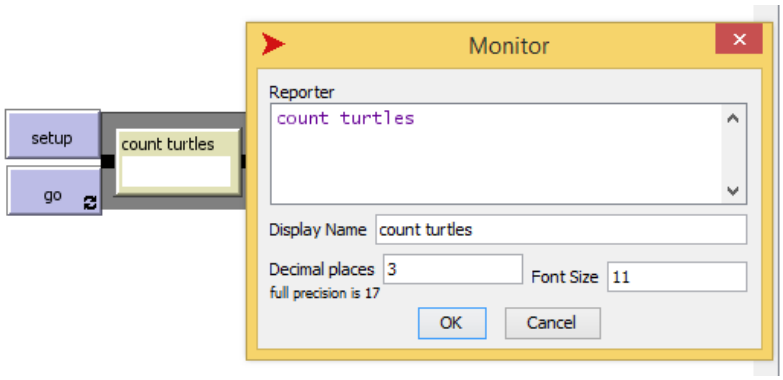
1. Tambahkan satu button, tentukan command dan display name dengan "setup".



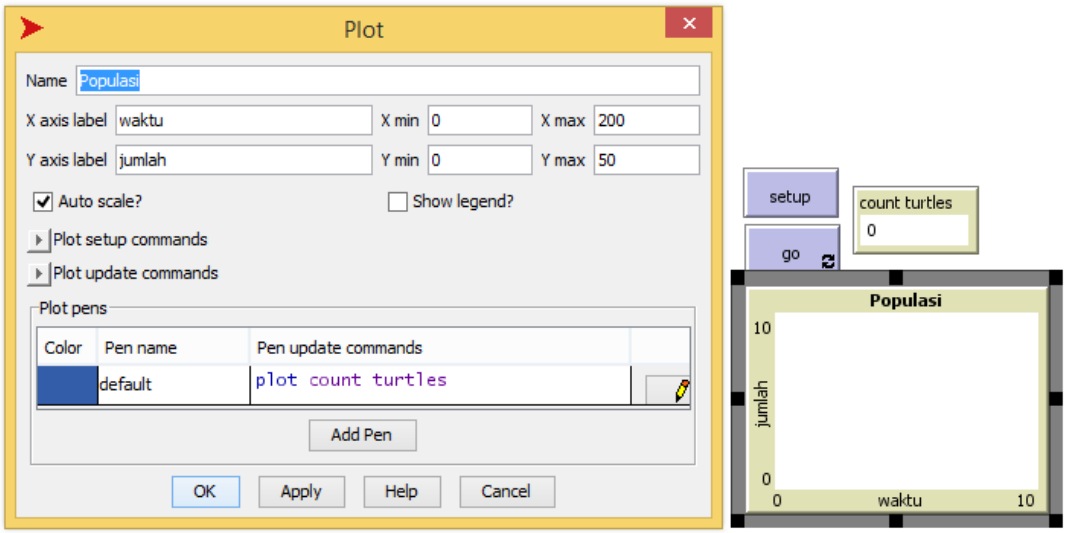
2. Tambahkan satu button lagi, tentukan juga command dan display name dengan "go", serta centang pada bagian forever (ini bertujuan agar button tetap tertekan pada saat diklik pertama, dan perintah akan terus dijalankan).



3. Tambahkan juga monitor, tentukan reporter dan display name dengan "count turtles", serta decimal places adalah "3".



4. Terakhir, tambahkan satu plot. Kemudian atur propertinya sebagai berikut:
   - Name = "Populasi"
   - X axis label = "waktu"
   - X max = 200
   - Y axis label = "jumlah"
   - Y max = 50



8.2. Pemberian Code

1. Untuk button setup, berikan procedure seperti ini.

```
to setup
  clear-all
  create-ordered-turtles 3 [
    set color pink
    fd 3
  ]
  reset-ticks
end
```

Penjelasan code di atas:

- `to setup` merupakan nama procedure
- `clear-all` untuk mereset world ke kondisi awal
- `create-ordered-turtles 3` untuk membuat agent turtles sebanyak 3 dan tersusun secara berurutan
- `set color pink` menentukan warna turtles dengan warna pink
- `fd 3` untuk memindahkan turtles agar maju 3 langkah
- `reset-ticks` untuk mereset tick counter ke kondisi awal

2. Untuk button go, berikan procedure seperti ini.

```
to go
  if not any? turtles [ stop ]
  ask turtles [
    wander
    reproduce
    grow-old
  ]
  tick
end
```

Penjelasan code di atas:

- `to go` merupakan nama procedure
- `if not any? turtles` merupakan kondisi pengecekan apakah terdapat turtles atau tidak, jika tidak maka `stop`
- `ask turtles` untuk memerintahkan agar semua turtles melakukan aksi yang terdapat pada blok bracket
- `wander, reproduce, grow-old` merupakan procedure yang diciptakan sendiri
- `tick` untuk meningkatkan nilai tick counter sebanyak satu tick

3. Tambahkan satu global variabel di paling atas code

```
turtles-own [ age ]
```

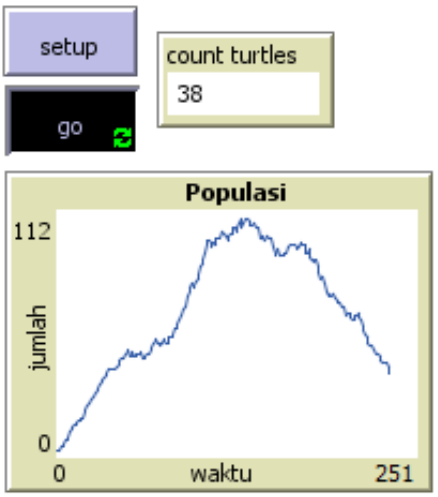
4. Tambahkan juga 3 procedure baru yang akan dipanggil oleh procedure go di atas.

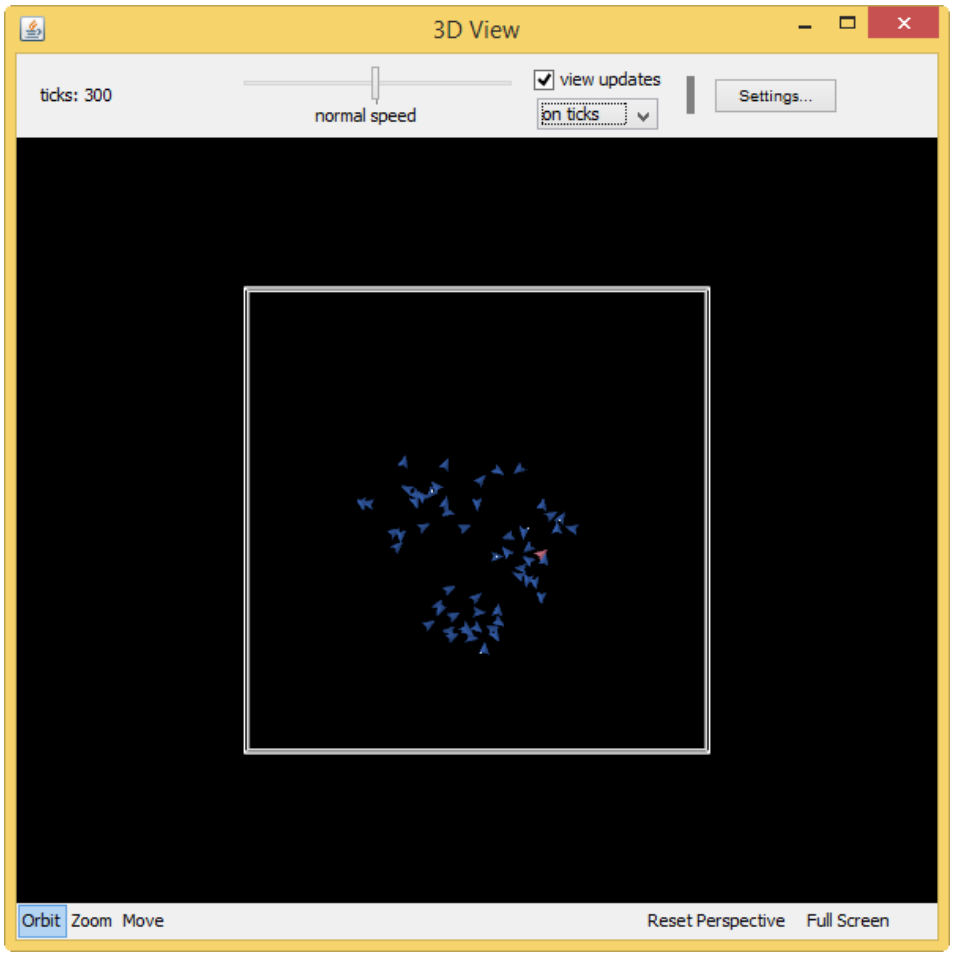
```
to grow-old
  set age age + 1
  if age > 50 [ die ]
end

to wander
  rt random 360
  fd 0.5
end

to reproduce
  if color = pink and random 5 = 0 [
    hatch 1 [
      if random 10 > 0 [ set color blue ]
      set age 0
    ]
  ]
end
```

Penjelasan code di atas:

- `to grow-old, to wander, to reproduce` merupakan nama procedure
- `set age age + 1` untuk menentukan age dengan nilai age + 1
- `if age > 50` merupakan kondisi untuk mengecek apakah age lebih dari 50 atau tidak, jika iya maka turtles tersebut `die`
- `rt random 360` untuk memutar ke arah kanan dengan sudut random antara 0 sampai 359
- `fd 0.5` untuk memindahkan turtles agar maju 0.5 langkah
- `if color = pink and random 5 = 0` merupakan kondisi untuk mengecek apakah warna turtles adalah pink dan nilai random 5 = 0 atau tidak, jika iya maka lakukan `hatch 1`
- `if random 10 > 0` merupakan kondisi untuk mengecek apakah nilai random 10 lebih dari 0 atau tidak, jika iya maka ubah warna turtles menjadi biru
- `set age 0` untuk menentukan age dengan nilai 0

5. Terakhir, jalankan model dengan cara klik pada button setup kemudian klik button go. Maka akan didapatkan hasil seperti ini.


setup
go
count turtles
38
Populasi
112
jumlah
0
0
waktu
251


3D View
ticks: 300
normal speed
view updates
on ticks
Settings...
Orbit
Zoom
Move
Reset Perspective
Full Screen

Pertemuan 9

SISTEM MULTI-AGENT DENGAN JADE

9.1. Pengantar Jade

Jade merupakan aplikasi middleware yang dapat digunakan untuk melakukan pengembangan sistem multi-agent. Dalam Jade terdapat bagian, yaitu:

1. Runtime environment, merupakan bagian dimana agent Jade dapat tinggal dan bagian tersebut harus aktif pada host yang diberikan sebelum satu atau lebih agent dapat dieksekusi pada host tersebut.
2. Library, merupakan bagian dimana programmer dapat menggunakannya secara langsung untuk membuat agent.
3. Grapichal tools, merupakan bagian untuk memonitoring dan melakukan pengaturan pada agent-agent yang sedang berjalan.

9.2. Container dan Platform

Setiap instance yang berjalan pada runtime Jade disebut dengan container yang dapat menampung beberapa agent. Sekumpulan container yang aktif disebut dengan platform. Sebuah main container harus selalu aktif dalam platform dan semua container-container yang lain akan ikut terregistrasi dengan container tersebut pada saat dijalankan. Container tersebut mengikuti container pertama untuk dijalankan di dalam sebuah platform yang harus menjadi main container apabila semua container-container lain menjadi container normal (bukan main container) dan harus bisa mengetahui dimana menemukan host dan port pada main container.

Command line di bawah ini akan menjalankan main container untuk mengaktifkan pengaturan manajemen GUI dari Jade (-gui).

<classpath> harus menyertakan semua class Jade dan juga semua class application-spesific yang dibutuhkan.

```
java -cp <classpath> jade.Boot -gui
```

Command line di bawah ini akan menjalankan container pendukung (-container option) yang terdaftar pada main container yang berjalan pada host avalon.tilab.com (-host option) dan akan mengaktifkan agent (misal Andi) dalam class myPackage.MyClass (-agents).

```
java -cp <classpath> jade.Boot -container –host avalon.tilab.com -agents
andi:myPackage.myClass
```

CREATING JADE AGENTS – AGENT CLASS

Pembuatan agent Jade dapat didefinisikan dengan menggunakan extend class `jade.core.Agent` dan diimplementasikan menggunakan method `setup()`, seperti pada code di bawah ini.

```
import jade.core.Agent;

public class BookBuyerAgent extends Agent {
  protected void setup() {
    System.out.println("Selamat datang! Buyer-agent"+
    getAID().getName()+" sudah siap.");
  }
}
```

Method `setup()` dimaksudkan untuk menyertakan inisialisasi dari agent.

9.3. Identifikasi Agent

Setiap agent diidentifikasikan oleh agent identifier yang direpresentasikan sebagai instance dari class `jade.core.AID`. Method `getAID()` dari class `Agent` memperbolehkan untuk mengembalikan agent idetifier. Setiap object `AID` menyertakan nama unik global ditambah juga nomor alamatnya. Nama dalam Jade memiliki bentuk *<nickname>@<platform-name>* jadi suatu agent (misal Andi) berada

pada suatu platform (misal Android0001) akan memiliki *Andi@Android0001* sebagai nama unik globalnya. Alamat yang termasuk dalam AID merupakan alamat-alamat dari platform dimana agent berada. Alamat-alamat ini hanya digunakan ketika suatu agent membutuhkan untuk berkomunikasi dengan agent lainnya pada platform yang berbeda.

Untuk mengetahui nickname dari agent, AID dapat diinisialisasikan seperti ini.

```
String nicname = "Andi";
AID id = new AID(nickname, AID.ISLOCALNAME)
```

Konstanta ISLOCALNAME mengindikasikan bahwa parameter pertama merepresentasikan nickname dan bukan merupakan nama unik global dari agent.

9.4. Menjalankan Agent

Dalam membuat agent dapat dicompile seperti ini.

```
javac –classpath <JADE-classes> BookBuyerAgent.java
```

Dalam menjalankan agent yang telah tercompile, runtime Jade harus dijalankan dan nicname agent yang akan dijalankan harus terpilih.

```
javac –classpath <JADE-classes>;. Jade.Boot buyer:BookBuyerAgent
```

Untuk hasil command line selengkapnya seperti ini.

```
C:\jade>java –classpath <JADE-classes>;. jade.Boot
buyer:BookBuyerAgent
10-ags-2014 06.54.45 jade.core.Runtime beginContainer
INFO:
-----------------------------------
This is JADE snapshot - revision 5995 of 2007/09/03 09:45:22
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
----------------------------------------
10-ags-2014 06.54.46 jade.core.BaseService init
INFO: Service jade.core.management.AgentM
anagement initialized
10-ags-2014 06.54.46 jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
```

```
10-ags-2014 06.54.47 jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
10-ags-2014 06.54.47 jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
10-ags-2014 06.54.47 jade.core.messaging.MessagingService
clearCachedSlice
INFO: Clearing cache
10-ags-2014 06.54.48 jade.mtp.http.HTTPServer <init>
INFO: HTTP-MTP Using XML parser
com.sun.org.apache.xerces.internal.parsers.SAXParser
10-ags-2014 06.54.49 jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://NBNT2004130496.telecomitalia.local:7778/acc
10-ags-2014 06.54.49 jade.core.AgentContainerImpl joinPlatform

INFO: ----------------------------------------
Agent container Main-Container@NBNT2004130496 is ready.
------------------------------------------------
Selamat datang! Buyer-agent buyer@NBNT2004130496:1099/JADE sudah
siap.
```

Bagian pertama dari output di atas merupakan disclaimer Jade yang tercetak setiap waktu saat runtime Jade dijalankan. Inisialisasi kernel service akan diaktifkan pada saat platform dijalankan. Pada bagian akhir yang mengindikasikan bahwa container yang disebut dengan "Main Container" telah siap dan runtime Jade berjalan. Ketika runtime Jade berjalan agent akan dijalankan dan akan menampilkan pesan selamat datang. Nickname dari agent adalah "buyer" dan nama platformnya adalah "NBNT2004130496:1099/JADE" yang akan otomatis tercatat pada host dan port yang dijalankan pada Jade.

9.5. Mengakhiri Agent

Meskipun tidak ada perintah lain yang harus dieksekusi setelah menampilkan pesan selamat datang di atas, agent tersebut masih akan tetap berjalan. Untuk mengakhiri agent dapat menggunakan method `doDelete()`. Sama seperti method `setup()` yang dipanggil oleh runtime Jade, agent akan berjalan segera dan dimaksudkan untuk menyertakan inisialisasi agent. Method `takeDown()` dipanggil hanya

sebelum agent berakhir dan dimaksudkan untuk menyertakan operasi pembersihan agent.

10.1. Melewati Argumen untuk Agent

Agent bisa jadi mendapatkan argumen yang spesifik yang diberikan pada command line. Argumen-argumen ini dapat dikembalikan sebagai array dari `Object` melalui method `getArguments()` dari class `Agent`. Misalkan kita ingin `BookBuyerAgent` mendapatkan judul dari buku yang akan dibeli sebagai argumen command line, maka kita ubah commandnya menjadi seperti ini.

```
import jade.core.Agent;
import jade.core.AID;

public class BookBuyerAgent extends Agent {
  private String targetBookTitle;
  private AID[] sellerAgents = {new AID(“seller1”,
AID.ISLOCALNAME), new AID(“seller2”, AID.ISLOCALNAME)};

  protected void setup() {
    System.out.println(“Selamat datang! Buyer-agent”+
    getAID().getName()+” sudah siap.”);

    Object[] args = getArguments();
    if(args != null && args.length > 0) {
      targetBookTitle = (String) args[0];
      System.out.println(“Coba membeli ” + targetBookTitle);
    } else {
      System.out.println(“Tidak ada judul buku”);
      doDelete();
    }
  }

  protected void takeDown() {
    System.out.println(“Buyer-agent”+getAID().getName()
+”berakhir.”);
  }
}
```

Untuk hasil argumen command line-nya seperti ini.

```
C:\jade>java jade.Boot buyer:BookBuyerAgent(Multi-Agent-
System)
...
...
10-ags-2014 06.58.21 jade.core.AgentContainerImpl
joinPlatform
INFO:
-----------------------------------------
Agent container Main-Container@NBNT2004130496 is ready.
-----------------------------------------
Selamat datang! Buyer-agent buyer@NBNT2004130496:1099/JADE
sudah siap.
Coba membeli Multi-Agent-System
```

Pertemuan 10

AGENT TASKS – BEHAVIOUR CLASS

Seperti yang telah dibahas pada pertemuan 9, tugas sebenarnya suatu agent adalah harus menunjukkan secara khusus behaviour-nya (tingkah laku) atau apa yang bisa dilakukan olehnya. Behaviour merepresentasikan tugas agent tersebut yang dapat menunjukkan dan mengimplementasikan suatu object dari class turunan `jade.core.behaviours.Behaviour`. Untuk membuat suatu agent menjalankan tugas yang diimplementasikan dengan behaviour object tersebut, cukup dengan menambahkan behaviour ke agent tersebut menggunakan method `addBehaviour()` dari class `Agent`. Behaviour yang dapat ditambahkan setiap waktu adalah ketika suatu agent berjalan (pada method `setup()`) atau dari behaviour lainnya.

Setiap class yang diturunkan dari class `Behaviour` harus diimplementasikan menggunakan method `action()`, yang sebenarnya mendefinisikan operasi yang ditampilkan pada saat behaviour berjalan. Method `done()` (menghasilkan nilai boolean) diimplementasikan untuk menunjukkan bahwa behaviour telah selesai dan harus diremove dari daftar behaviour agent.

10.1. Penjadwalan dan Pengeksekusian Behaviour

Suatu agent dapat menjalankan beberapa behaviour secara bersamaan. Bagaimanapun hal tersebut penting untuk memperhatikan bahwa jadwal dari behaviour agent tidak pre-emptive tetapi tetap saling bekerjasama. Hal ini menunjukkan ketika behaviour dijadwalkan untuk dijalankan menggunakan method `action()` yang dipanggil dan akan berjalan sampai method tersebut menghasilkan nilai balik (return).

Command di bawah ini menunjukkan bahwa behaviour akan dijalankan sejak method `action()` tidak pernah menghasilkan nilai balik.

```
public class OverbearingBehaviour extends Behaviour {
  public void action() {
    while(true) {
      // lakukan sesuatu
    }
  }

  public boolean done() {
    return true;
  }
}
```

Pada saat tidak ada behaviour untuk menjalankan thread agent maka akan menuju ke kondisi sleep dimana kondisi tersebut tidak akan memakan waktu CPU. Behaviour akan tersedia kembali jika akan melakukan suatu eksekusi tertentu.

10.2. One-shot Behaviour, Cyclic Behaviour, dan Generic Behaviour

1. One-shot behaviour merupakan behaviour yang akan segera selesai dan method `action()` hanya akan dijalankan sekali saja. Package jade.core.behaviours.OneShotBehaviour sudah mengimplementasikan method `done()` dengan menghasilkan nilai true dan dapat dengan mudah diturunkan untuk mengimplementasi one-shot behaviour.

   ```
   public class MyOneShotBehaviour extends OneShotBehaviour {
     public void action() {
       // perintah operasi X
     }
   }
   ```

   Perintah operasi X hanya akan ditampilkan sekali saja.
2. Cyclic behaviour merupakan behaviour yang tidak pernah selesai dan method `action()` akan menjalankan operasi yang sama setiap kali dipanggil. Package jade.core.behaviours.OneShotBehaviour

sudah mengimplementasikan method `done()` dengan menghasilkan nilai false dan dapat dengan mudah diturunkan untuk mengimplementasi cyclic behaviour.

```
public class MyCyclicBehaviour extends CyclicBehaviour {
  public void action() {
    // perintah operasi Y
  }
}
```

Perintah operasi Y akan ditampilkan secara berulang-ulang.

3. Generic behaviour, merupakan behaviour yang akan menanamkan status dan menjalankan operasi yang berbeda bergantung pada status tersebut. Behaviour tersebut akan selesai ketika diberikan kondisi yang terpenuhi.

```
public class MyThreeStepBehaviour extends Behaviour {
  private int step = 0;
  public void action() {
    switch(step) {
    case 0:
       //perintah operasi X
       step++;
       break;
    case 1:
       //perintah operasi Y
       step++;
       break;
    case 2:
       //perintah operasi Z
       step++;
       break;
    }
  }

  public boolean done() {
    return step == 3;
  }
}
```

Perintah operasi X,Y,Z akan ditampilkan sekali setelah yang lain dan kemudian behaviour selesai.

10.3. Penjadwalan Operasi

Jade menyediakan 2 class untuk melakukan penjadwalan (pada package `jade.core.behaviours`) dengan maksud agar memudahkan

dalam mengimplementasikan behaviour yang menjalankan operasi pada batas waktu yang diberikan.

1. `WakerBehaviour`; method `action()` dan `done()` sudah diimplementasikan untuk menjalankan method abstrak `handleElapsedTimeout()` setelah diberikan batas waktu (ditentukan pada constructor). Setelah menjalankan method `handleElapsedTimeout()` maka behaviour selesai.

```
public class MyAgent extends Agent {
  protected void setup() {
    System.out.println("Tambahkan waker behaviour");
    addBehaviour(new WakerBehaviour(this, 10000) {
      protected void handleElapsedTimeout() {
        // perintah operasi X
      }
    } );
  }
}
```

   Operasi X akan ditampilkan 10 detik setelah mencetak "Tambahkan waker behaviour".

2. `TickerBehaviour`; method `action()` dan `done()` sudah diimplementasikan untuk menjalankan method abstrak `onTick()` secara berulang-ulang dengan menunggu periode waktu yang diberikan (ditentukan pada constructor) setelah setiap kali dieksekusi. `TickerBehaviour` behaviour tidak pernah selesai.

```
public class MyAgent extends Agent {
  protected void setup() {
    addBehaviour(new TickerBehaviour(this, 10000) {
      protected void onTick() {
        // perintah operasi Y
      }
    } );
  }
}
```

   Operasi Y akan ditampilkan secara periodik setiap 10 detik.