



Expert System

Hands on Lab



September 2011

For the latest information, please see bluejack.binus.ac.id



Information in this document, including URL and other Internet Web site references, is subject to change without notice. This document supports a preliminary release of software that may be changed substantially prior to final commercial release, and is the proprietary information of Binus University.

This document is for informational purposes only. BINUS UNIVERSITY MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, AS TO THE INFORMATION IN THIS DOCUMENT.

The entire risk of the use or the results from the use of this document remains with the user. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Binus University.

Binus University may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Binus University, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2011 Binus University. All rights reserved.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Table of Content

OVERVIEW	1
Chapter01 - Basic of Expert System	2
Chapter02 - Building Facts	18
Chapter03 - Building Rules	29
Chapter04 - Knowledge Representation in Rules	45
Chapter05 - Build an Expert System	61



OVERVIEW

Chapter 01

- Basic of Expert System

Membahas syntax dasar dari library Jess (Java Expert System Shell) meliputi variabel, input, output, seleksi, perulangan, dan fungsi.

Chapter 02

- Building Facts

Membangun fakta sebagai dasar representasi objek pada Jess, termasuk cara inisialisasi, modifikasi, dan penghapusan fakta.

Chapter 03

- Building Rules

Membangun rules untuk melakukan pemrosesan terhadap fakta pada *working memory*.

Chapter 04

- Knowledge Representation in Rules

Membahas cara merepresentasikan pengetahuan ke dalam rules dengan mekanisme forward dan backward chaining, serta pencarian *breadth* dan *depth*.

Chapter 05

- Build an Expert System

Membangun Sistem Pakar berbasis GUI dengan menggunakan rules-rules yang sudah dibuat.

Chapter 01

Basic of Expert System



1.1. Jess

Jess (Java Expert System Shell) adalah sebuah rule engine yang ditulis dalam bahasa Java oleh Ernest Friedman-Hill di Sandia National Laboratories di Livermore, CA.

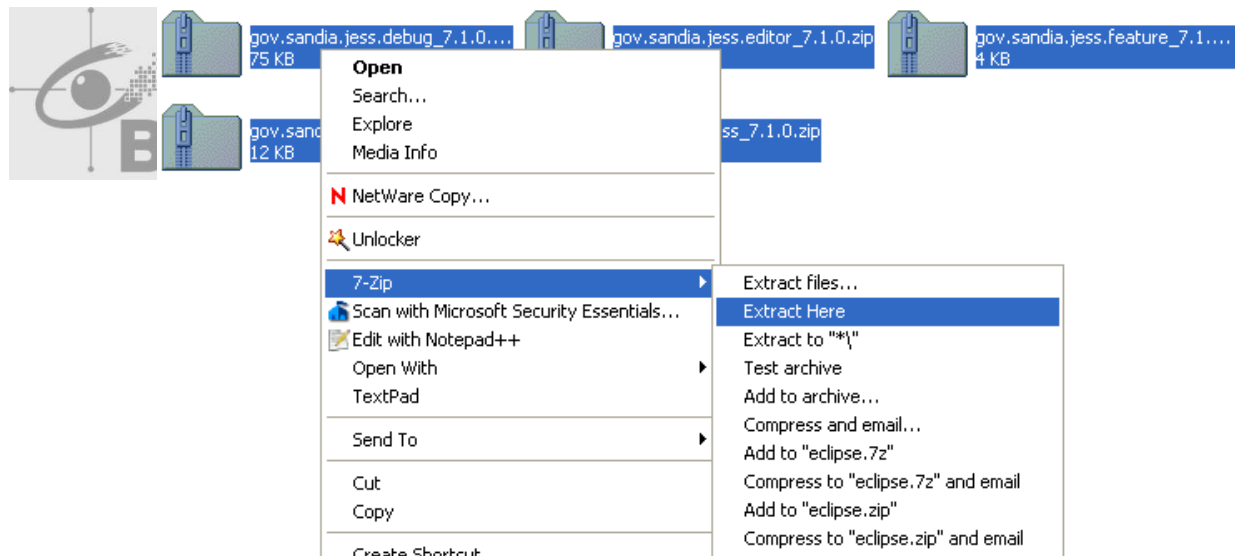
Melalui rule engine, suatu program berbasis pengetahuan dapat dibuat. Artinya, basis program bukan lagi prosedural (*step per step*), namun mengikuti aturan yang sudah ditentukan dari pengetahuan yang ada.

1.2. Instalasi Jess

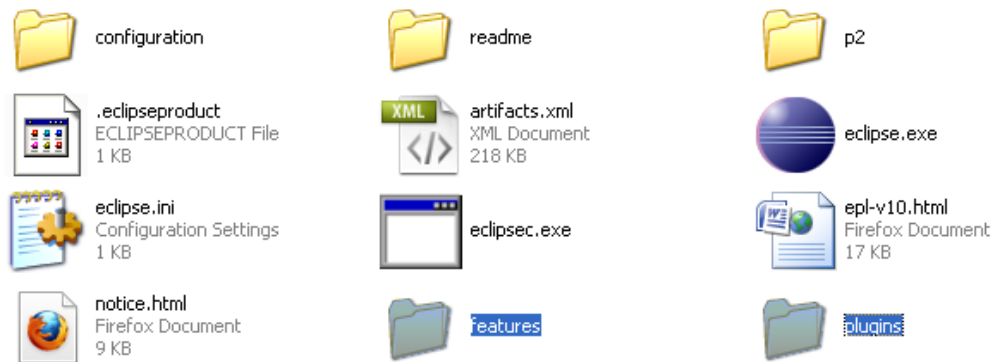
Jess dapat di-download secara gratis melalui website resmi di <http://www.jessrules.com/>.

Langkah-langkah untuk menghubungkan **Jess** dengan **Eclipse** adalah sebagai berikut:

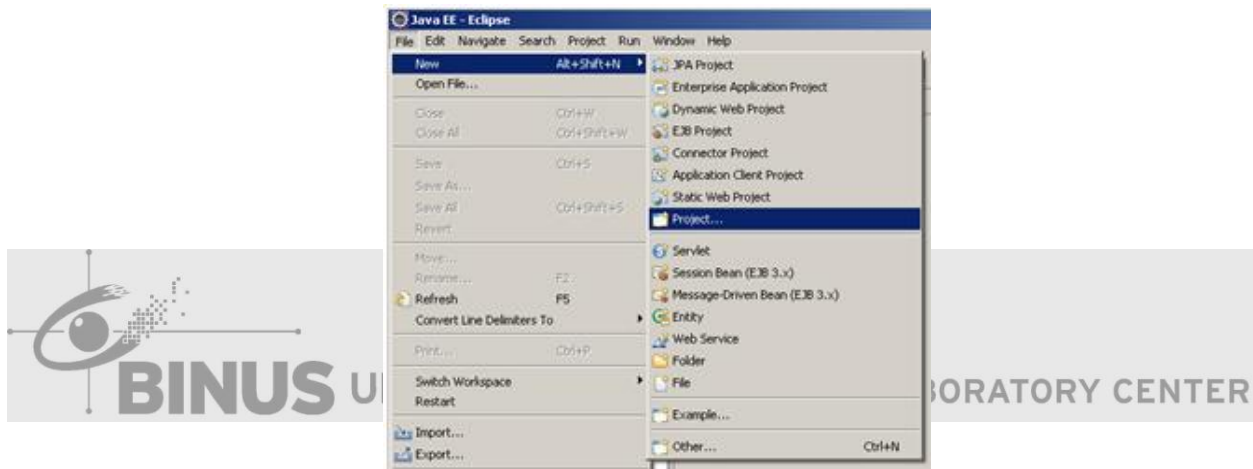
1. Extractlah hasil download dari <http://www.jessrules.com/>. Pada hasil extract akan terdapat folder **eclipse**. Masuk ke dalam folder tersebut, kemudian, lakukan ekstraksi terhadap semua file zip yang ada.



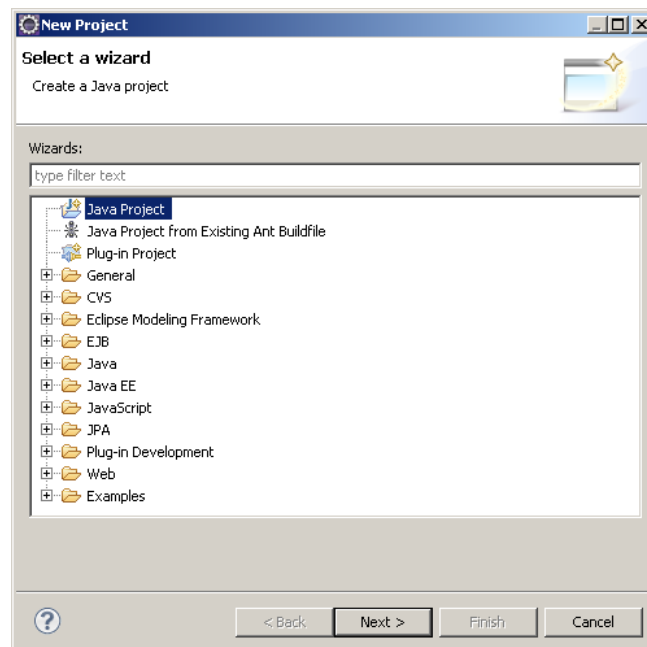
2. Hasil ekstraksi adalah folder **features** dan **plugins**. Lakukan copy terhadap kedua folder itu ke dalam folder eclipse. (Jika ada pesan apakah ingin overwrite, pilih yes).



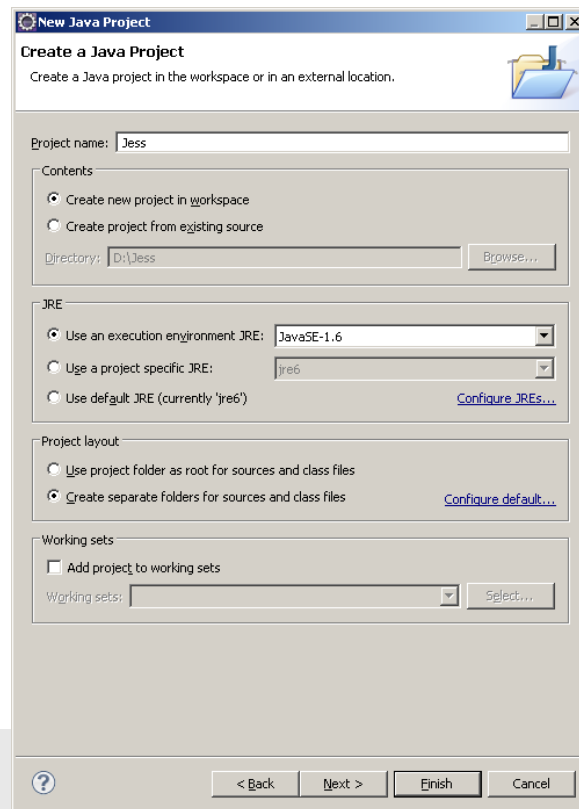
3. Buatlah Project Dengan menggunakan program Eclipse dengan cara **File> New> Project**.



4. Pilih Java Project

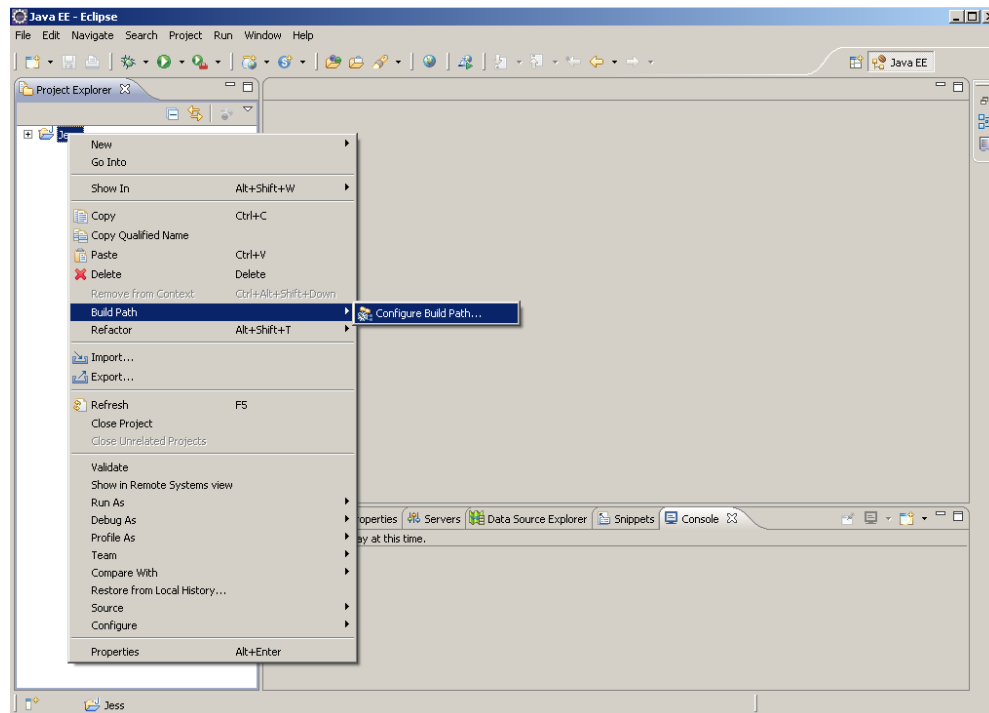


5. Masukkan nama Project lalu tekan tombol finish

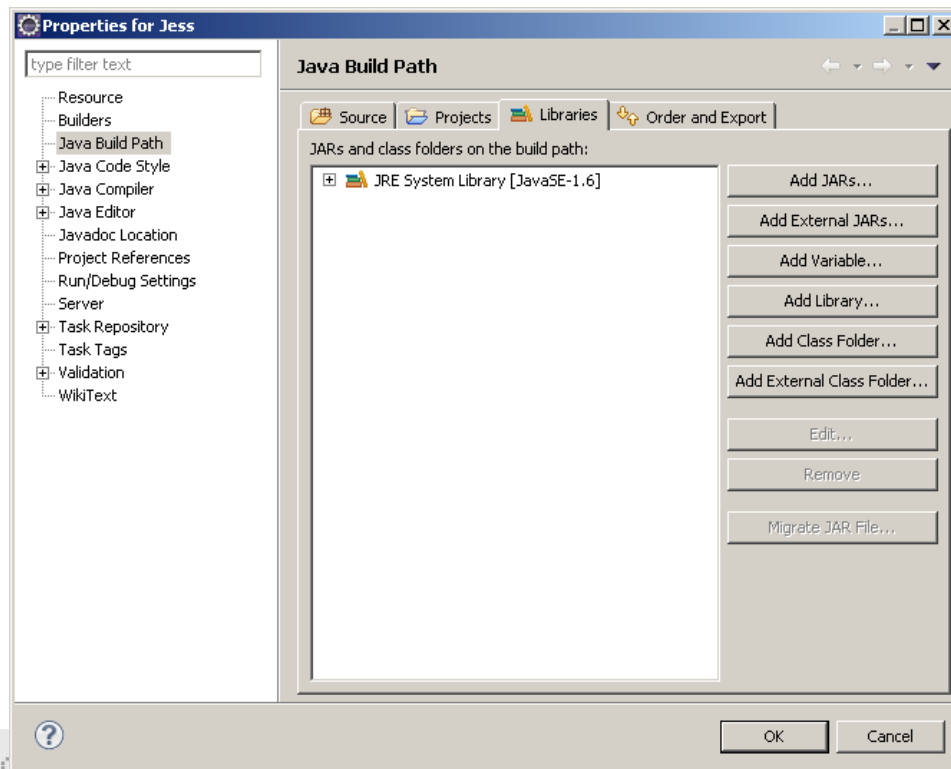


6. Copy file “jess.jar” dan “jsr94.jar” dan masukkan ke dalam “**Project Eclipse**”

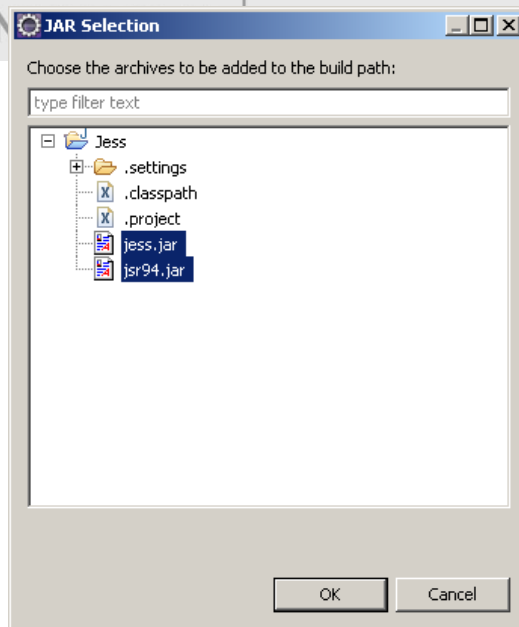
7. Klik kanan pada project lalu pilih Build Path > Configure Build Path



8. Pilih Tab Libraries dan tekan tombol **Add Jars**.



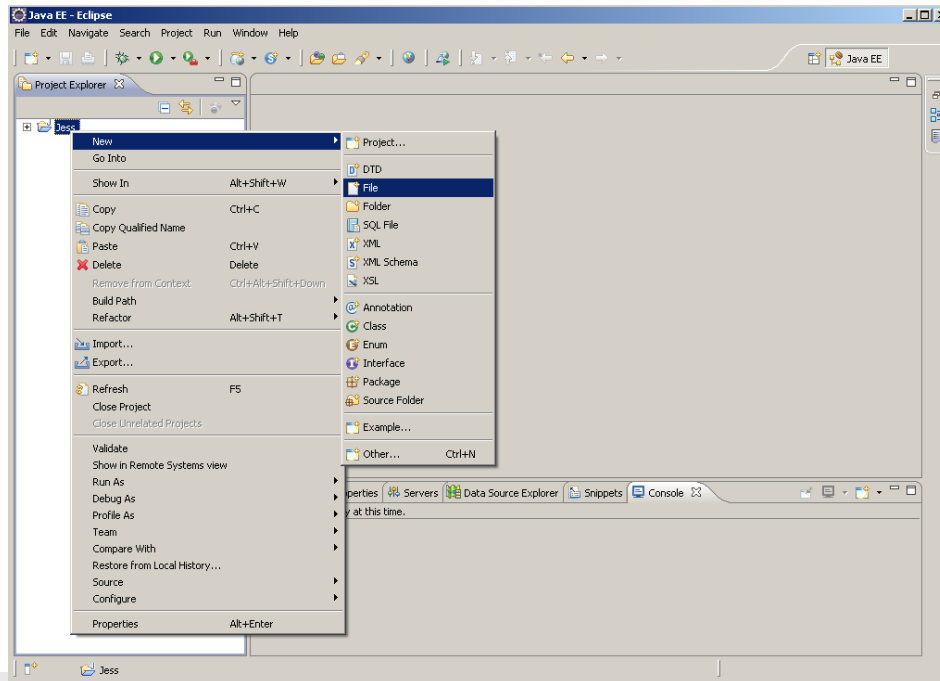
9. Pilih jess.jar dan jsr94.jar. Klik OK.



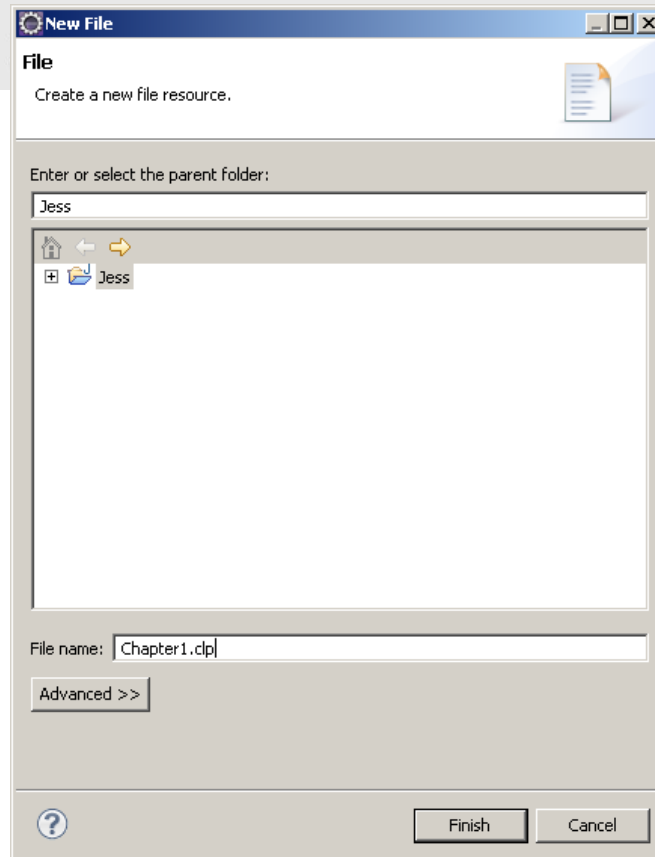
Jika jess.jar dan jsr.jar ada pada list, pastikan bahwa copy jar sudah benar-benar ada pada project, dan pastikan bahwa project sudah ter-*refresh*. Cara refresh project: Klik kanan project > Refresh.

File Jess Berektensi .clp. Cara pembuatan file adalah sebagai berikut:

1. Pada project yang telah dibuat klik kanan> **new> File**



2. Masukkan nama file diakhiri dengan **.clp**. Kemudian, klik **Finish**.



1.3. Jess Syntax

Dalam Jess, variabel tidak memiliki tipe data spesifik. Jadi sebuah variabel dapat menampung berbagai jenis data seperti angka, simbol, string, dan boolean. Variabel pada Jess dibagi menjadi 2, yaitu **slot** dan **multislot**.

Slot adalah variabel yang dapat menampung 1 nilai, sedangkan multislot adalah variabel yang dapat menampung lebih dari 1 nilai (list). Multislot identik dengan istilah array dalam bahasa pemrograman lain.

Cara pembuatan slot adalah “?<namavariabel>”. Cara pembuatan multislot adalah “\$?<variabel>”. Contoh pembuatan variable adalah :

```
(bind ?angka 1)
(bind ?pecahan 1.123)
(bind ?simbol Lalala)
(bind ?string "Hallo dunia")
(bind ?bool TRUE)
(bind $?array (create$ 12 10 3))
```

Pada bahasa pemrograman Java, code di atas identik dengan:

```
int angka = 1;
float pecahan = 1.123f;
String string = "Hallo dunia";
boolean bool = true;
int array[] = {12, 10, 3};
```

Untuk melakukan operasi aritmatika seperti penjumlahan, pengurangan, pembagian, dan perkalian, pada Jess digunakan notasi prefix.

Contoh:

```
(+ 1 1)
(/ (- 20 10) 2)
(* (- 3 2) (+ 3 1))
```

Code di atas identik dengan:

```
1 + 1
```

$(20 - 10) / 2$

$(3-2) * (3+1)$

Output

Untuk melakukan output ke layar konsole, dapat digunakan printout.

(printout t <String>)

Contoh:

```
(bind ?a 5)
(bind ?b 10)
(printout t "Selamat Datang")
(printout t "Baris ini diakhiri enter" \n)
(printout t "Mencetak variabel a " ?a \n)
(printout t ?a " x " ?b \n)
```

t pada printout selalu digunakan untuk menandakan output ke layar konsole.

\n adalah *clear line feed*, digunakan untuk pindah baris.

Input

Untuk melakukan input dari konsole, dapat digunakan perintah (read) atau (readline)

```
(bind ?angka 0)
(printout t "Masukkan angka: ")
(bind ?angka (read))

(bind ?string "")
(printout t "Masukkan string: ")
(bind ?string (readline))
```

(readline) digunakan untuk menerima inputan hingga tombol enter ditekan sehingga biasanya digunakan untuk menerima input string (data yang mengandung spasi), sedangkan (read) digunakan untuk menerima inputan hingga tombol enter atau spasi ditekan.

Increment dan decrement operator

Kegunaan operator increment dan decrement adalah mempersingkat penulisan dan mempermudah programmer dalam membaca code.

Increment dan decrement berlaku hanya pada variabel numerik.

Contoh penggunaan:

```
(bind ?i 5)
(++ ?i)
```

Code di atas identik dengan code berikut.

```
(bind ?i 5)
(bind ?i (+ ?i 1))
```

Seleksi

Jess hanya memiliki fungsi if untuk melakukan control seleksi.

Syntax if:

```
(if <kondisi> then
```

```
...
```

```
elif <kondisi> then
```

```
...
```

```
else
```

```
...
```

```
)
```

Contoh:

```
(if (eq ?angka 1) then
  (printout t "Angka adalah 1" crlf)
elif (eq ?angka 2) then
  (printout t "Angka adalah 2" crlf)
else
  (printout t "Angka bukan 1 dan 2" crlf)
)
```

eq (equals) digunakan untuk membandingkan dua buah data apakah sama. Jika kedua data memiliki nilai yang sama, maka nilai balikan adalah TRUE.

Sebaliknya **neq** (not equals) digunakan untuk membandingkan dua buah data apakah tidak sama. Jika kedua data memiliki nilai yang berbeda, maka nilai balikan barulah TRUE.

Perulangan

Terdapat berbagai jenis looping dalam Jess seperti:

a. While

```
(while <kondisi>
  <statement>
)
```

<kondisi> merupakan kondisi perulangan. Apabila kondisi masih terpenuhi, maka <statement> akan dijalankan

Contoh:

```
{while (< ?angka 5)
  {printout t "Angka adalah " ?angka}
  {++ ?angka}
}
```

b. For

```
(for <inisialisasi> <kondisi> <increment/decrement>
  <statement>
)
```

<inisialisasi> merupakan pemberian nilai awal

<kondisi> merupakan kondisi perulangan

<increment/decrement> merupakan penambahan atau pengurangan dalam perulangan

Contoh:

```
{for (bind ?angka 5) (>= ?angka 1) (-- ?angka)
  {printout t "Angka adalah " ?angka}
}
```

c. Foreach

```
(foreach ?var $?list
  <statement>
)
```

?var merupakan variabel sementara yang akan diisi oleh value dari \$?list

Contoh:

```
{bind $?data {create$ "Anto" "Budi" "Cica"}}
{foreach ?n ?data
  {printout t ?n \n}}
}
```

Fungsi

Cara pembuatan fungsi:

```
(defunction <nama fungsi>(<parameter>)
  <statement>
)
```

Contoh:

```
{defunction cetak()
  {printout t "Fungsi cetak" \n}}
}
```

Contoh fungsi yang menerima parameter:

```
{defunction jumlah (?a ?b)
  {return (+ ?a ?b)}}
}
```

Cara pemanggilan fungsi di atas adalah:

```
{cetak}
{jumlah 10 20}
```

Multislot

Multislot adalah suatu array atau list. List ini selalu dimulai dari indeks ke 1.

Pembuatan multislot yaitu dengan (**create\$** [data1] [data2] [data3] ..).

Contoh:

```
{bind $?data {create$ "Anto" "Budi" "Cica"}}
{foreach ?n ?data
  {printout t ?n \n}}
```

Code di atas identik dengan:

String data[] = {"Anto", "Budi", "Cica"}

Selain dengan cara di atas, multislot dapat dibuat dengan memisahkan string berdasarkan spasi dengan perintah **explode\$**. (khusus untuk string saja)

Contoh:

```
{bind $?data {explode$ "Anto Budi Cica"}}
{foreach ?n ?data
  {printout t ?n \n}}
```

Hasilnya akan sama seperti dengan memakai (create\$) di atas.

Untuk mengakses member indeks ke i dari multislot, digunakan perintah **nth\$**.

Contoh:

```
{bind $?data {create$ "Anto" "Budi" "Cica"}}
{printout t "Data ke 2 adalah " {nth$ 2 ?data}}
```

Maka data kedua adalah "Budi".

Untuk mencari data (mengetahui indeks) dapat digunakan member\$.

Contoh:

```
{bind $?data {create$ "Anto" "Budi" "Cica"}}
{printout t "Data ke Cica ada di indeks ke " {member$ "Cica" ?data}}
```


Untuk menambah data, dapat digunakan **create\$**.

Contoh:

```
(bind $?data (create$ "Anto" "Budi" "Cica"))
(bind ?data (create$ ?data "Dodi" "Erik"))
(foreach ?n ?data
  (printf t ?n \n))
```

Komentar

Untuk menyisipkan komentar pada Jess, gunakan tanda `/* ... */` atau `;`...

Contoh:

```
(bind ?a 0)
/* Komentar
   Baris Code ini tidak dijalankan */
(printf t "Hallo" \n) ; Untuk mencetak Hallo ke layar
```

1.4. Exercise

Buatlah sebuah program sederhana seperti di bawah ini :

Simple Program With Jess

Input Your Name [3..20]:

Nama yang diinput harus memiliki panjang karakter antara 3 sampai 20

Setelah menginput nama, maka akan muncul tampilan seperti berikut :

```
Simple Program With Jess

Input Your Name [3..20]: David Ang
Input Your Age [1..99]:
```

Umur yang diinput harus antara 1 sampai 99.

Program akan menampilkan hasil inputan yang telah diinput oleh user. Jika user memilih yes, maka program akan mengulang dari awal

```
Input Your Name [3..20]: David Ang
Input Your Age [1..99]: 22

Your Name : David Ang
Your Age : 22

Do You Want to repeat[yes/no]: yes
```

Jawab

Langkah-langkah pengerjaan :

1. Buat sebuah project dengan menggunakan **Eclipse**.
2. Buat File baru berekstensi .clp.
3. Buat fungsi clearScreen()

```


1 (deffunction clearScreen()
2   (for (bind ?i 0) (< ?i 30) (++ ?i)
3     (printout t crlf)
4   )
5   (printout t "                               Simple Program With Jess")
6   (for (bind ?i 0) (< ?i 10) (++ ?i)
7     (printout t crlf)
8   )
9 )

```

Penjelasan:

Fungsi bertujuan untuk menambahkan enter dan mencetak judul program. Perulangan digunakan untuk mencetak enter sebanyak 40 kali

4. Coding dari program yang dibuat :



```

11 (bind ?choose "")
12 (while (neq ?choose no)
13   (clearScreen)
14   (bind ?name "")
15   (while (or (< (str-length ?name) 3) (> (str-length ?name) 20) )
16     (printout t "Input Your Name [3..20]: " )
17     (bind ?name (readline))
18     (if (eq (lexemep ?name) FALSE) then
19       (bind ?name "")
20     )
21   )
22   (bind ?age 0)
23   (while (or (< ?age 1) (> ?age 99) )
24     (printout t "Input Your Age [1.99]: " )
25     (bind ?age (read))
26     (if (eq (numberp ?age) FALSE) then
27       (bind ?age 0)
28     )
29   )
30   (printout t crlf crlf "Your Name : " ?name crlf)
31   (printout t "Your Age : " ?age crlf)
32   (printout t crlf crlf crlf)
33
34   (bind ?choose "")
35   (while (and (neq ?choose yes) (neq ?choose no) )
36     (printout t "Do You Want to repeat[yes/no]: " )
37     (bind ?choose (read))
38   )
39 )

```

Penjelasan:

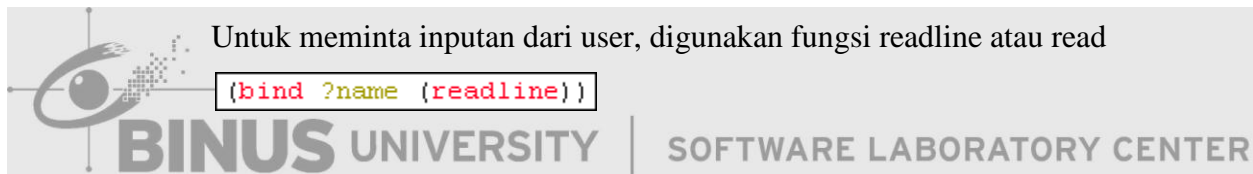
- a. Program ini dimulai dengan membuat sebuah variabel ?choose dan diberi nilai awal kosong.

- b. Program akan mengulang jika variabel ?choose bukan bernilai no.
- c. Setelah itu, program akan memanggil fungsi (clearScreen) untuk membersihkan layar.
- d. Untuk membuat inputan nama, maka dilakukan proses input sebagai berikut:

```
(bind ?name "")
(while (or (< (str-length ?name) 3) (> (str-length ?name) 20) )
  (printout t "Input Your Name [3..20]: " )
  (bind ?name (readline))
  (if (eq (lexemep ?name) FALSE) then
    (bind ?name ""))
  )
)
```

Fungsi str-length digunakan untuk mendapatkan panjang karakter dari sebuah variabel string.

Fungsi lexemep digunakan untuk mengecek sebuah variabel berisi string atau bukan. Selain fungsi lexemep, terdapat juga numberp yang digunakan untuk mengecek angka atau bukan.



Untuk meminta inputan dari user, digunakan fungsi readline atau read

```
(bind ?name (readline))
```

- e. Untuk proses menginput umur, sama seperti proses menginput nama

```
(bind ?age 0)
(while (or (< ?age 1) (> ?age 99) )
  (printout t "Input Your Age [1.99]: " )
  (bind ?age (read))
  (if (eq (numberp ?age) FALSE) then
    (bind ?age 0))
  )
)
```

- f. Setelah itu, program akan menampilkan hasil dan meminta inputan mengulang atau tidak

```
(printout t crlf crlf "Your Name : " ?name crlf)
(printout t "Your Age : " ?age crlf)
(printout t crlf crlf crlf)

(bind ?choose "")
(while (and (neq ?choose yes) (neq ?choose no) )
  (printout t "Do You Want to repeat[yes/no]: " )
  (bind ?choose (read))
  )
)
```

Chapter 02

Building Facts



2.1 Facts

Fakta adalah segala sesuatu yang tertangkap oleh indra manusia. Catatan atas pengumpulan fakta disebut data. Fakta seringkali diyakini oleh orang banyak sebagai hal yang sebenarnya, baik karena mereka telah mengalami kenyataan-kenyataan dari dekat maupun karena mereka dianggap telah melaporkan pengalaman orang lain yang sesungguhnya.

Contoh fakta:

1. Ada mobil di jalan.

```
{mobil-di-jalan}
```

2. Ada murid bernama Lala.

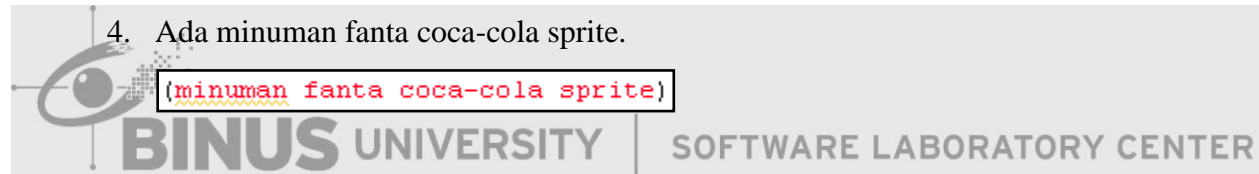
```
{Student Lala}
```

3. Ada Jurusan TI.

```
{Jurusan TI}
```

4. Ada minuman fanta coca-cola sprite.

```
{minuman fanta coca-cola sprite}
```



Jess memiliki memori kerja (*working memory*) untuk menyimpan fakta-fakta yang ada selama program berjalan. Fakta ini yang nantinya akan digunakan untuk diolah dalam suatu aturan.

2.2 Assert untuk menambahkan fakta

Assert digunakan untuk menambahkan fakta ke *working memory*.

Contoh:

```
{assert {mobil-di-jalan}}
{assert {student Lala}}
```

Penjelasan:

Menambahkan fakta (**mobil-di-jalan**), dan menambahkan fakta (**student Lala**). Setiap fakta yang ditambahkan ke *working memory* memiliki nomor fakta. Nomor fakta program JESS dimulai dari 0.

Dari contoh di atas, jika (**mobil-di-jalan**) di-assert, maka fakta tersebut memiliki nomor fakta 0, sedangkan nomor fakta dari (**student Lala**) adalah 1.

Fakta yang ditambahkan haruslah bersifat unik.

Contoh kode berikut.

```
{assert {student Lala}}
{assert {student Lala}}
{assert {student Lala}}
```

Kode di atas menambahkan fakta (**student Lala**) ke *working memory* hanya 1 kali saja. Oleh karenanya, tidak perlu lagi assert suatu fakta jika fakta itu sudah ada sebelumnya.

2.3 facts

(**facts**) digunakan untuk menampilkan semua fakta yang ada pada layar beserta dengan nomor faktanya, dan juga jumlah fakta yang ada.

2.4 Retract untuk menghapus fakta

Retract digunakan untuk menghapus fakta. Untuk menghapus suatu fakta harus diketahui dahulu nomor faktanya.

(retract <nomor fakta>)

Contoh:

```
{retract 1}
```

Penjelasan:

Menghapus fakta no 1. Jika menggunakan Contoh fakta diatas, maka fakta yang akan terhapus adalah fakta (**student Lala**). Nomor-nomor fakta yang lain tidak berubah.

2.5 Reset dan clear untuk menghapus semua fakta

Reset digunakan untuk menghapus semua fakta, menambah fakta yang terdapat pada deffacts, dan menambah fakta (initial-fact) dengan nomor fakta 0.

```
{reset}
```

Clear digunakan untuk menghapus semua fakta, rule, template, activations, dan lainnya.

```
{clear}
```

2.6 deffacts

Deffacts adalah kumpulan fakta-fakta untuk diisi ke program ketika fungsi (reset) dipanggil. Umumnya deffacts digunakan untuk inisialisasi fakta awal.

(deffacts <Nama Fakta>

<FAKTA1>

<FAKTA2>



Contoh:

```
{deffacts FAKTA
  (JURUSAN TI)
  (JURUSAN SI)
  (MATAKULIAH TI "Algoritma")
  (MATAKULIAH TI "Struktur Data")
  (MATAKULIAH SI "Programming I")
}
```

Penjelasan:

Pada deffacts FAKTA di atas, terdapat fakta JURUSAN dan MATAKULIAH. Fakta yang tersedia meliputi: Jurusan TI dan SI. Terdapat fakta Mata Kuliah TI dengan nama "Algoritma", Mata Kuliah TI dengan nama "Struktur Data", dan Mata Kuliah SI dengan nama "Programming I".

Ketika suatu (**reset**) dipanggil, maka kelima fakta di atas akan di-assert otomatis beserta juga dengan fakta (**initial-fact**) dengan nomor fakta 0, sehingga isi fakta pada *working memory* ketika (**reset**) dipanggil adalah:

f-0 (MAIN::initial-fact)
 f-1 (MAIN::JURUSAN TI)
 f-2 (MAIN::JURUSAN SI)
 f-3 (MAIN::MATAKULIAH TI "Algoritma")
 f-4 (MAIN::MATAKULIAH TI "Struktur Data")
 f-5 (MAIN::MATAKULIAH SI "Programming I")

2.7 deftemplate

Deftemplate adalah sebuah rancangan atau template untuk pembuatan fakta. Isi dari deftemplate dapat meliputi rancangan yang berupa slot atau multislot.

Contoh:

```
(deftemplate orang
  (slot nama)
  (slot umur)
  (multislot hobbi))
```

Penjelasan:

Untuk setiap fakta orang, maka orang tersebut memiliki 3 buah data, yaitu nama, umur, dan hobbi. Nama dan umur diberi slot, artinya hanya memiliki 1 nilai, sedangkan hobbi diberi multislot, artinya setiap orang bisa memiliki banyak hobbi.

Jika ingin memasukkan fakta orang dengan nama “Budi”, umur 10, dan hobbi makan dan tidur, maka fakta tersebut ditulis sbb:

```
(assert (orang (nama "Budi") (umur 10) (hobbi makan tidur)))
```

Urutan tidak berpengaruh, sehingga code di bawah memiliki arti yang sama seperti code di atas:

```
(assert (orang (hobbi makan tidur) (nama "Budi") (umur 10)))
```

Slot untuk setiap deftemplate dapat diberi nilai default.

Contoh:

```
(deftemplate orang
  (slot nama (default ""))
  (slot umur (default 0))
  (multislot hobbi (default nil))
)
```

Artinya jika default untuk nama adalah "", default untuk umur adalah 0 dan default untuk hobbi adalah nil.

Contohnya, jika fakta orang ditambahkan tanpa mendefinisikan nama, umur, dan hobbi, seperti code di bawah:

```
(assert (orang))
```

maka orang tersebut memiliki nama, umur, dan hobbi sesuai dengan nilai default.

Kegunaan deftemplate adalah untuk memudahkan programmer dalam menyusun fakta, sehingga fakta dapat diasumsikan seperti objek yang memiliki atribut-attribut.

Jika suatu fakta tidak memiliki template, maka template dari fakta tersebut secara default adalah

```
(deftemplate <nama-fakta> (multislot __data))
```

2.8 modify

```
(modify <no-fakta> <modif>)
```

Modify berfungsi untuk mengubah atribut suatu fakta.

Contoh:

```
(deftemplate orang
  (slot nama)
  (slot umur)
  (multislot hobbi)
)
(assert (orang (nama "Budi") (umur 10) (hobbi makan)))
(modify 0 (nama "Buddi"))
(facts)
```

Modify di atas mengubah fakta dengan no 0. Nama sebelumnya diubah menjadi "Buddi".

2.9 duplicate

(duplicate <no-fakta> <diff>)

Duplicate berfungsi untuk menduplikasi suatu fakta dengan perubahan pada attribut.

Contoh:

```
{deftemplate orang
  (slot nama)
  (slot umur)
  (multislot hobbi)
}
(assert (orang (nama "Budi") (umur 10) (hobbi makan)))
(duplicate 0 (nama "Buddi"))
(facts)
```

Duplicate akan menggandakan fakta 0 dengan menciptakan perbedaan pada nama, yaitu “Buddi”, sehingga hasil akhir adalah 2 buah fakta, sebagai berikut.

f-0 (MAIN::orang (nama "Budi") (umur 10) (hobbi makan))

f-1 (MAIN::orang (nama "Buddi") (umur 10) (hobbi makan))



2.10 Exercise

Buatlah sebuah program yang menggunakan seperti berikut :

```
Simple Program With Jess

Menu
1. Add Data
2. View Data
3. Exit
Input Your Choice [1..3]:
```

Jika user memilih tombol 1 "Add Data", maka program akan seperti berikut:

```

Simple Program With Jess

Menu
1. Add Data
2. View Data
3. Exit
Input Your Choice [1..3]: 1
Input Your Name [3..20]: Dav Dav
Input Your Age [1..99]: 28
Success Added Data...

```

Jika user memilih tombol 2 "View Data", maka program akan menampilkan fakta-fakta yang tersedia sebagai berikut:

```

Simple Program With Jess

Menu
1. Add Data
2. View Data
3. Exit
Input Your Choice [1..3]: 2
The List Data:
=====
f-0    (MAIN::initial-fact)
f-1    (MAIN::PEOPLE (name David) (age 20))
f-2    (MAIN::PEOPLE (name Steve) (age 10))
f-3    (MAIN::PEOPLE (name Chris) (age 35))
f-4    (MAIN::PEOPLE (name Dav Dav) (age 28))
For a total of 5 facts in module MAIN.
=====
Press Enter To Continue...

```

Jawaban :

1. Buat project baru dari Eclipse, dan buat file .clp.
2. Buat deftemplate PEOPLE

```
1 (deftemplate PEOPLE
2   (multislot name)
3   (slot age)
4 )
```

Penjelasan :

Template PEOPLE terdiri dari name dan age. Untuk name menggunakan multislot karena nama akan disimpan ke dalam list. Untuk age menggunakan slot.

3. Buat list fakta pada saat awal

```
6 (deffacts FAKTA
7   (PEOPLE (name David) (age 20))
8   (PEOPLE (name Steve) (age 10))
9   (PEOPLE (name Chris) (age 35))
10 )
```

Penjelasan :

List fakta yang ada pada saat program dijalankan.

4. Buat fungsi clearScreen untuk menghapus layar (secara teknis dilakukan dengan mencetak enter sekitar 40 kali).
5. Buat fungsi menu()

```
21 (deffunction menu()
22   (printout t "Menu" crlf)
23   (printout t "1. Add Data" crlf)
24   (printout t "2. View Data" crlf)
25   (printout t "3. Exit" crlf)
26 )
```

Penjelasan:

Fungsi menu digunakan untuk mencetak daftar menu yang tersedia

6. Kode lengkap

```

28 (reset)
29 (bind ?pilih 0)
30 (while (neq ?pilih 3)
31   (clearScreen)
32   (menu)
33   (bind ?pilih 0)
34   (while (or (< ?pilih 1) (> ?pilih 3) )
35     (printout t "Input Your Choice [1..3]: " )
36     (bind ?pilih (read))
37     (if (eq (numberp ?pilih) FALSE) then
38       (bind ?pilih 0)
39     )
40   )
41   (if (eq ?pilih 1) then
42     (bind ?name "")
43     (while (or (< (str-length ?name) 3) (> (str-length ?name) 20) )
44       (printout t "Input Your Name [3..20]: " )
45       (bind ?name (readline))
46       (if (eq (lexemep ?name) FALSE) then
47         (bind ?name "")
48       )
49     )
50     (bind ?age 0)
51     (while (or (< ?age 1) (> ?age 99) )
52       (printout t "Input Your Age [1..99]: " )
53       (bind ?age (read))
54       (if (eq (numberp ?age) FALSE) then
55         (bind ?age 0)
56       )
57     )
58     (assert (PEOPLE (name (explode$ ?name)) (age ?age)))
59     (printout t "Success Added Data...")
60     (readline)
61   elif (eq ?pilih 2) then
62     (printout t "The List Data: " crlf)
63     (printout t "===== " crlf)
64     (facts)
65     (printout t "===== " crlf)
66     (printout t "Press Enter To Continue...")
67     (readline)
68   else
69     (printout t "Thank You.." crlf)
70   )
71 )

```

Penjelasan:

- Fungsi (**reset**) berguna untuk menghapus semua fakta pada awal dan menambah fakta pada deffacts.
- Pada statement assert, terdapat fungsi explode\$ yang berguna untuk mengubah string menjadi list. Karena ?nama adalah string, sedangkan deftemplate PEOPLE pada name adalah list, maka ?nama harus diubah menjadi list. Selain fungsi explode\$, terdapat fungsi implode\$ yang bertujuan untuk mengubah list menjadi string
- facts yang berguna untuk menampilkan list fakta yang ada.

Chapter 03

Building Rules



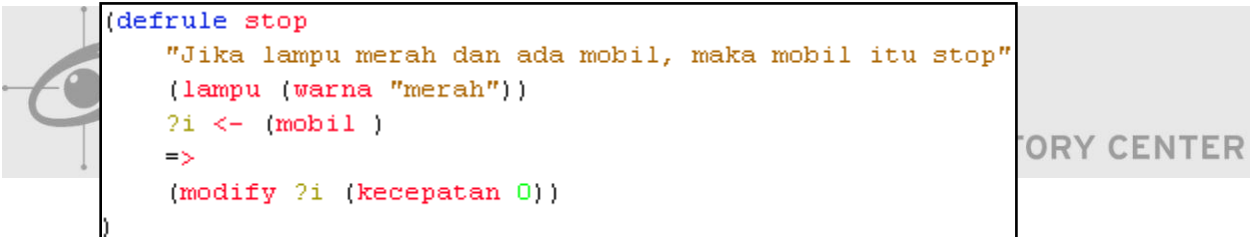
3.1. Rule

Rule pada JESS adalah suatu aturan yang digunakan untuk pemroses fakta-fakta yang ada pada working memory.

Syntax defrule:

```
(defrule <nama-rule>
  "comment"
  <LHS>
  =>
  <RHS>
)
```

Sebagai contoh, rule yang sederhana seperti : Jika lampu berwarna merah, maka mobil harus berhenti. Jika dituliskan ke dalam Jess:



```
(defrule stop
  "Jika lampu merah dan ada mobil, maka mobil itu stop"
  (lampu (warna "merah"))
  ?i <- (mobil )
  =>
  (modify ?i (kecepatan 0))
)
```

LHS (*Left Hand Side*) adalah ruas kiri dari rule, berfungsi sebagai syarat suatu rule apakah dijalankan atau tidak. Pada LHS, setidaknya terdapat satu atau lebih fakta sebagai kondisi.

RHS (*Right Hand Side*) adalah ruas kanan dari rule, berisi statement yang akan dijalankan (*fire*).

3.2. run

(**run**) adalah suatu fungsi yang digunakan untuk menjalankan **rule-rule yang sudah aktif** pada working memory.

3.3. Kapan Suatu Rule aktif

Suatu rule aktif jika terjadi **perubahan pada fakta pada working memory**, seperti ketika suatu fakta di-assert, di-retract, di-modify, dan di-duplicate.

Suatu rule dapat aktif berkali-kali asalkan diaktivasi oleh fakta yang berbeda.

Contoh:

```
(defrule rule-cetak
  "Jika ada orang, cetak hai"
  (orang)
  =>
  (printout t "hai"crLf))
```

Jika dilakukan assert 3 buah fakta orang yang berbeda, semisal orang dengan nama “a”, “b”, dan “c”, maka akan mengaktivasi rule-cetak sebanyak 3 kali. Ketika (run) dipanggil, maka akan tercetak 3 hai di layar.

```
(deftemplate orang
  (slot name))

(defrule rule-cetak
  "Jika ada orang, cetak hai"
  (orang)
  =>
  (printout t "hai"crLf))

(assert (orang (name "a")))
(assert (orang (name "b")))
(assert (orang (name "c")))
(run)
```

SOFTWARE LABORATORY CENTER

3.4. Mengambil atribut pada fakta untuk diproses

Suatu atribut dari fakta dapat ditampung ke dalam variabel. Pada contoh di bawah, setiap variabel name pada fakta orang ditampung ke variabel ?n, sehingga dapat diproses di RHS.

```
(deftemplate orang
  (slot name))

(defrule rule-cetak
  "Jika ada orang, cetak namanya"
  (orang (name ?n))
=>
  (printout t "Nama orang tsb " ?n crlf))

(assert (orang (name "a")))
(assert (orang (name "b")))
(assert (orang (name "c")))
(run)
```

Hasil:

Nama orang tsb c

Nama orang tsb b

Nama orang tsb a

3.5. Mengambil nomor fakta pada rule

Operator <- dapat digunakan untuk mengambil nomor fakta pada rule. Nomor fakta kemudian ditampung ke dalam variabel. Pada code di bawah, nomor fakta dari rule yang ada ditampung ke variabel ?no.

```
(deftemplate orang
  (slot name))

(defrule rule-cetak
  ?no <- (orang (name ?n))
=>
  (printout t "No faktanya " ?no crlf)
  (printout t "Nama orang tersebut " ?n crlf crlf))

(assert (orang (name "a")))
(assert (orang (name "b")))
(assert (orang (name "c")))
(run)
```

Hasil code di atas:

No faktanya <Fact-2>

Nama orang tersebut c

No faktanya <Fact-1>

Nama orang tersebut b

No faktanya <Fact-0>

Nama orang tersebut a

Nomor fakta digunakan jika kita ingin melakukan pemrosesan fakta yang melibatkan nomor faktanya, seperti:

(**retract** <nomor-fakta>)

(**modify** <nomor-fakta> <changes>)

(**duplicate** <nomor-fakta> <differences>)

3.6. Rule yang melibatkan beberapa fakta

Suatu rule bisa melibatkan lebih dari 1 fakta. Contoh:

```
(deftemplate orang
  (slot name))

(defrule rule-cetak
  (orang (name ?n1))
  (orang (name ?n2))
  =>
  (printout t "Nama orang 1 " ?n1 crlf)
  (printout t "Nama orang 2 " ?n2 crlf crlf))

(assert (orang (name "a")))
(assert (orang (name "b")))
(assert (orang (name "c")))
(run)
```

Hasil rule tersebut:

Nama orang 1 c

Nama orang 2 c

Nama orang 1 b

Nama orang 2 c

Nama orang 1 c

Nama orang 2 b

Nama orang 1 c

Nama orang 2 a

Nama orang 1 a

Nama orang 2 c

Nama orang 1 b

Nama orang 2 b

Nama orang 1 a

Nama orang 2 b

Nama orang 1 b

Nama orang 2 a

Nama orang 1 a

Nama orang 2 a

Ketika code (`assert (orang (name "a"))`) dijalankan, maka rule akan aktif sekali, yaitu `<Fact-0>-<Fact-0>`

Ketika code (`assert (orang (name "b"))`) dijalankan, maka rule akan aktif 3 kali, yaitu `<Fact-0>-<Fact-1>` , `<Fact-1>- <Fact-0>` dan `<Fact-1>- <Fact-1>`.

Ketika code (`assert (orang (name "c"))`) dijalankan, maka rule akan aktif 5 kali, yaitu `<Fact-0>-<Fact-2>` , `<Fact-2>-<Fact-0>`, `<Fact-1>-<Fact-2>`, `<Fact-2>-<Fact-1>` dan `<Fact-2>- <Fact-2>`.

Maka, total rule tersebut akan aktif adalah sebanyak 9 kali.

3.7. Konstrain

Suatu rule dapat diberi konstrain.

a. &: dan test

```
(deftemplate orang
  (slot name)
  (slot age))

(defrule rule-cetak-age>20
  (orang (name ?n) (age ?a&(> ?a 20)))
  =>
  (printout t "Nama " ?n crlf))

(reset)
(assert (orang (name "a") (age 20)))
(assert (orang (name "b") (age 25)))
(assert (orang (name "c") (age 15)))
(run)
```

Rule di atas akan aktif jika orang tersebut memiliki umur di atas 20.

Rule di atas ekuivalen jika menggunakan (test) seperti di bawah.

```
(defrule rule-cetak-age>20
  (orang (name ?n) (age ?a))
  (test (> ?a 20))
  =>
  (printout t "Nama " ?n crlf))
```

b. And dan or

And digunakan agar semua kondisi fakta harus memenuhi, sedangkan or digunakan agar satu kondisi saja memenuhi maka rule akan aktif.

Contoh:

```
(defrule rule-cetak
  (or (orang (name "a"))
      (orang (age 20)))
  =>
  (printout t "Hai" crlf))
```

Penggunaan and dan or bisa saling nested. Contoh:

```
(defrule rule-cetak
  (or (and (orang (name "a"))
            (orang (age 20)))
      (and (orang (name "b"))
            (orang (age 15))))
  =>
  (printout t "Hai" crlf))
```

c. Not

Not digunakan ketika suatu fakta tidak ada, maka rule tersebut akan aktif.

Contoh:

```
(defrule rule-cetak
  (not (orang (name "d")))
  =>
  (printout t "Tidak ada orang dengan nama d" crlf))
```

d. Exists

Exists digunakan agar ketika suatu fakta ada (ditemukan), maka rule tersebut akan aktif cukup satu kali.

```
(defrule rule-cetak
  (exists (orang ))
  =>
  (printout t "Ada orang" crlf))
```

Hasil: Tercetak "Ada orang" sebanyak 1 kali di layar.

e. Accumulate

Accumulate memberikan fasilitas untuk menghitung jumlah fakta, menambah field, memasukkan data ke koleksi, dan sebagainya.

Syntax accumulate:

(accumulate <initializer> <action> <result> <conditional element>)

Contoh:

```
(defrule rule-cetak
  ?c <- (accumulate (bind ?i 0) (++ ?i) ?i (orang))
  =>
  (printout t "Ada " ?c " orang" crlf))
```

f. Forall

Forall akan memeriksa semua kebenaran dari fakta yang dilingkupinya. Jika benar semua, maka RHS akan dijalankan 1 kali saja. Contoh:

```
(deftemplate orang
  (slot name)
  (slot age))

(deftemplate car
  (slot owner))

(defrule rule-cetak
  (forall (orang (name ?n))
    (car (owner ?n)))
  =>
  (printout t "Setiap orang punya mobil" crlf))

(reset)
(assert (orang (name "a") (age 20)))
(assert (orang (name "b") (age 25)))
(assert (orang (name "c") (age 15)))
(assert (car (owner "a")))
(assert (car (owner "b")))
(assert (car (owner "c")))
(run)
```

3.8. defglobal

Defglobal berisi variable global yang akan dikenali di semua rule maupun fungsi yang ada pada program tersebut. Cara pembuatan variable global adalah `?*<namavariabel>*`. Pada saat pendeklarasian variable global, harus disertakan nilai awal untuk variable tersebut. Variable ini dibuat dalam lingkup defglobal.

Contoh:

```
(defglobal
  ?*find* = FALSE
  ?*total* = 0
)
```


3.9. Exercise

Buatlah sebuah program yang menggunakan deffacts, deftemplate, dan defrule seperti berikut :

```
Menu
1. Add Data
2. View Data
3. Search Data
4. Exit
Input Your Choice [1..4]:
```

Jika user memilih tombol 1 "Add Data", maka program akan seperti berikut:


```
Menu
1. Add Data
2. View Data
3. Search Data
4. Exit
Input Your Choice [1..4]: 1
Input Your Name [3..20]: Simple
Input Your Age [1..99]: 20
Success Added Data...
```

Jika user memilih tombol 2 "View Data", maka program akan menampilkan fakta-fakta yang tersedia sebagai berikut:

```
Menu
1. Add Data
2. View Data
3. Search Data
4. Exit
Input Your Choice [1..4]: 2
The List Data:
=====
f-0    (MAIN::initial-fact)
f-1    (MAIN::PEOPLE (name "David") (age 20))
f-2    (MAIN::PEOPLE (name "Steve") (age 10))
f-3    (MAIN::PEOPLE (name "Chris") (age 35))
f-4    (MAIN::PEOPLE (name "Chris") (age 20))
f-5    (MAIN::PEOPLE (name "Simple") (age 20))
For a total of 6 facts in module MAIN.
=====
Press Enter To Continue...
```

Jika user memilih tombol 3 “Search Data”, maka program akan:

- Meminta inputan nama yang dicari.
- Jika nama yang dicari ada didalam fakta, maka data akan ditampilkan, sertakan juga nomor urut untuk data yang ditemukan.
- Jika nama yang dicari tidak ada didalam fakta, maka tampilkan pesan “No Data Found”.



```
Menu
1. Add Data
2. View Data
3. Search Data
4. Exit
Input Your Choice [1..4]: 3
Input Name [3..20]: Chris
Person named Chris is found:
== 1. =====
Name : Chris
Age  : 20
=====
== 2. =====
Name : Chris
Age  : 35
=====
Press Enter To Continue..
```

Tampilan jika data ditemukan

```
Menu
1. Add Data
2. View Data
3. Search Data
4. Exit
Input Your Choice [1..4]: 3
Input Name [3..20]: Program
Data not found
Press Enter To Continue..
```

Tampilan jika data tidak ditemukan

Jawaban:

1. Buat project baru dari Eclipse.
2. Buat deftemplate PEOPLE

```

3 (deftemplate PEOPLE
4   (slot name)
5   (slot age)
6 )

```

Penjelasan :

Template People terdiri dari name dan age. Keduanya diberi tipe data slot.

3. Buat list fakta pada saat awal

```

8 (deffacts FAKTA
9   (PEOPLE (name "David") (age 20))
10  (PEOPLE (name "Steve") (age 10))
11  (PEOPLE (name "Chris") (age 35))
12  (PEOPLE (name "Chris") (age 20))
13 )

```

Penjelasan :

List fakta yang ada pada saat program dijalankan.

4. Buat fungsi menu() dan clearScreen()

```

18 (deffunction clearScreen()
19   (for (bind ?i 0) (< ?i 30) (++ ?i)
20     (printout t crlf)
21   )
22 )
23
24 (deffunction menu()
25   (printout t "Menu" crlf)
26   (printout t "1. Add Data" crlf)
27   (printout t "2. View Data" crlf)
28   (printout t "3. Search Data" crlf)
29   (printout t "4. Exit" crlf)
30 )

```

Penjelasan:

Fungsi menu digunakan untuk mencetak daftar menu yang tersedia. Fungsi clearScreen digunakan untuk mencetak enter 30 kali, untuk membersihkan layar.

5. Buat sebuah defglobal ?*num*

```
1 (defglobal ?*num* = nil)
```

Penjelasan:

Variabel global ini dibuat untuk menampilkan nomor urut pada saat mencetak fakta yang dicari.

6. Buat sebuah rule search

```
32 (defrule search-found
33   (search ?n)
34   (exists (PEOPLE (name ?n)) )
35   =>
36   (printout t "Person named "?n " is found: " crlf)
37   (bind ?*num* 1)
38 )
39
40 (defrule search-loop
41   (search ?n)
42   (PEOPLE (name ?n) (age ?age))
43   =>
44   (printout t "== "?*num* ". =====" crlf)
45   (printout t "Name : " ?n crlf)
46   (printout t "Age : " ?age crlf)
47   (printout t "===== " crlf crlf)
48   (++ ?*num*)
49 )
50
51 (defrule search-not-found
52   (search ?n)
53   (not (PEOPLE (name ?n)))
54   =>
55   (printout t "Data not found" crlf)
56 )
57 (defrule search-end
58   ?i<- (search ?search)
59   =>
60   (retract ?i)
61 )
```

Penjelasan:

Terdapat 4 buah rule untuk search, yaitu search-found, search-loop, search-not-found, dan search-end

- Search-found

Jika ada fakta (search) dan setidaknya ada satu data (PEOPLE) pada working memory, maka program akan mencetak pesan bahwa orang tersebut ada, kemudian akan memberi nilai awal untuk nomor urut dengan 1.

- Search-loop

Jika ada fakta (search), untuk setiap fakta (orang), maka cetak nomor urut, nama, dan umurnya.


- Search-not-found

Jika ada fakta (search) namun tidak ada fakta (people) sesuai dengan yang dicari, maka tampilkan pesan bahwa “Data not found”.

- Search-end

Jika ada fakta (search), maka hapus fakta ini sehingga fakta (search) dapat mentrigger aktivasi rule-search lagi.

7. Kode lengkap untuk main program



```

63 (reset)
64 (bind ?pilih 0)
65 (while (or (eq (numberp ?pilih) FALSE) (neq ?pilih 4))
66   (clearScreen)
67   (menu)
68   (printout t "Input Your Choice [1..4]: " )
69   (bind ?pilih (read))
70   (if (eq ?pilih 1) then
71     (bind ?name "")
72     (while (or (eq (lexemep ?name) FALSE) (< (str-length ?name) 3)
73       (> (str-length ?name) 20) )
74       (printout t "Input Your Name [3..20]: " )
75       (bind ?name (readline))
76     )
77     (bind ?age 0)
78     (while (or (eq (numberp ?age) FALSE) (< ?age 1) (> ?age 99) )
79       (printout t "Input Your Age [1..99]: " )
80       (bind ?age (read))
81     )
82     (assert (PEOPLE (name ?name) (age ?age)))
83     (printout t "Success Added Data...")
84     (readline)
85   elif (eq ?pilih 2) then
86     (printout t "The List Data: " crlf)
87     (printout t "===== " crlf)
88     (facts)
89     (printout t "===== " crlf)
90     (printout t "Press Enter To Continue...")
91     (readline)
92   elif (eq ?pilih 3) then
93     (bind ?name "")
94     (while (or (< (str-length ?name) 3) (> (str-length ?name) 20) )
95       (printout t "Input Name [3..20]: " )
96       (bind ?name (readline))
97     )
98     (assert (search ?name))
99     (run)
100    (printout t "Press Enter To Continue..")
101    (readline)
102  )
103 )

```

Penjelasan:

- Fungsi (reset) berguna untuk menghapus semua fakta pada awal dan menambah fakta pada deffacts ke *working memory*.
- Fungsi (run) berguna untuk menjalankan semua rule yang sudah teraktivasi.
- Terdapat fungsi (facts) yang berguna untuk menampilkan list fakta yang ada.

- Readline berfungsi untuk meminta inputan user sampai user menekan tombol enter.
- Pemanggilan (assert (search ?name)), akan mengaktivasi rule Search.

Chapter 04

Knowledge Representation in Rules



4.1. Knowledge Representation in Rules

Pengetahuan direpresentasikan didalam rule. Sebagai contoh, gejala penyakit Malaria adalah demam, sakit kepala, dan mual. Di dalam Jess akan tertulis sebagai berikut :

```
(defrule malaria
  {fever}
  {headache}
  {nausea}
  =>
  (assert (disease (name malaria)))
)
```

Pengetahuan dalam rule di atas dikhususkan untuk *backward chaining*.

Secara umum, terdapat 2 cara/metode untuk mencari solusi:

1. Forward chaining

Program akan mencocokkan LHS (Left Hand Side) untuk menjalankan RHS (*Right Hand Side*).

Untuk menggunakan metode *forward chaining*, diperlukan suatu gambaran yang jelas dari awal permasalahan. Oleh karenanya, suatu *flowchart* dibutuhkan untuk menggambarkan alur dari permasalahan hingga menemukan solusi.

2. Backward Chaining

Sistem akan melihat RHS, kemudian mencari cara agar LHS terpenuhi.

Pada Jess, backward chaining dapat dilakukan dengan memanggil fungsi
(*do-backward-chaining* <fact>)

Pada contoh (defrule malaria) di atas, backward chaining untuk rule tersebut bias dilakukan jika menambahkan code berikut di atasnya:

```
(do-backward-chaining (fever))
(do-backward-chaining (headache))
(do-backward-chaining (nausea))
```

Jika suatu fakta (a) dibutuhkan untuk mencapai RHS, backward-chaining akan meng-assert fakta

```
(need-a)
```

Contohnya, jika ada fakta (**headache**) dan fakta (**nausea**), namun belum ada fakta (**fever**), maka engine akan meng-assert fakta (**need-fever**)

Untuk menggunakan metode *backward chaining*, yang diperlukan adalah bagaimana suatu solusi didapat. Oleh karenanya, fokus masalah adalah bagaimana suatu solusi dapat dicapai/dicari. Proses pencarian dapat berupa *breadth*, *depth*, dan sebagainya.

Untuk set strategi pencarian, gunakan fungsi
(**set-strategy** <strategy>)

Misalnya: (**set-strategy** breadth) untuk pencarian *breadth* atau melebar.

4.2. Exercise

Buatlah program untuk mencari permasalahan pada komputer yang bermasalah ketika *boot-up*. Program akan bertanya pada user mengenai sejumlah pertanyaan yang relevan, dan akan menampilkan solusi.

Tampilan layar:

```
Does computer turn on? [yes/no] : yes
Anything on screen? [yes/no] : yes
Computer boots every time? [yes/no] : yes
Does computer beep? [yes/no] : yes
There is motherboard trouble
```

Flowchart:



Jawaban:

Cara 1: Forward Chaining

Cara ini akan melihat LHS yang sesuai dan dijalankan RHSnya. Jika tidak ada Rule yang sesuai, maka program akan berhenti.


1. Buat deftemplate untuk question, ask, answer, dan solution

```

1 (deftemplate question
2   (slot id)
3   (slot text))
4
5 (deftemplate ask
6   (slot id)
7   (slot text))
8
9 (deftemplate answer
10  (slot id)
11  (slot answer))
12
13 (deftemplate solution
14  (slot text))

```

2. Buat deffacts untuk pertanyaan-pertanyaan yang disediakan beserta idnya.



```

16 (def facts questions
17   "list of questions"
18   (question (id 1) (text "Does computer turn on?"))
19   (question (id 2) (text "Anything on screen?"))
20   (question (id 3) (text "Adequate power source and environment?"))
21   (question (id 4) (text "Computer boots every time?"))
22   (question (id 5) (text "Power switch working?"))
23   (question (id 6) (text "Does computer beep?"))
24   (question (id 7) (text "New part added?"))
25   (question (id 8) (text "Do you hear hard drive spin up?"))
26   (question (id 9) (text "Power supply to motherboard correct?"))
27   (question (id 10) (text "PC can boot from CD or floppy?"))
28   (question (id 11) (text "Try another power IDE cable & slot. Does it work?"))
29   (question (id 12) (text "Test Power supply. Is it damaged?"))
30   (question (id 13) (text "Does drive work in another PC?")))

```

3. Buat defrule init-ask untuk memulai pertanyaan

```

182 (defrule init-ask
183   (question (id 1) (text ?text))
184   =>
185   (assert (ask (id 1) (text ?text))))
186
187 (reset)
188 (run)

```

Penjelasan:

Pada saat program pertama kali dijalankan, program akan menanyakan pertanyaan id ke-1.

4. Buat defrule ask-question yang digunakan untuk menanyakan pertanyaan ke user

```

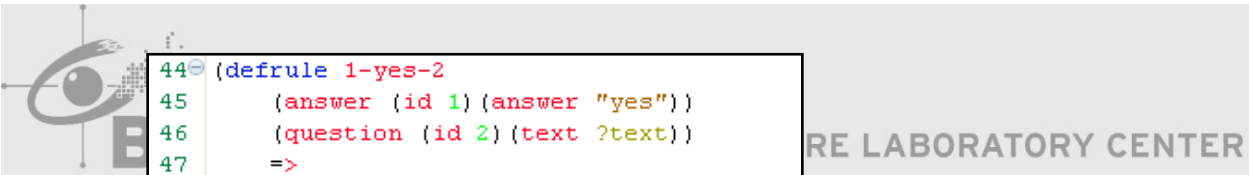
32 (defrule ask-question
33   "If there is an ask, an question but there is no answer yet, then ask it"
34   (ask (id ?i) (text ?text))
35   (not (answer (id ?i)))
36   =>
37   (bind ?answer "")
38   (while (and (neq (str-compare ?answer "yes") 0)
39               (neq (str-compare ?answer "no") 0)) do
40     (printout t ?text " [yes/no] : ")
41     (bind ?answer (readline)))
42   (assert (answer (id ?i) (answer ?answer))))

```

Penjelasan:

Pada LHS, terdapat 2 kondisi dimana jika ada pertanyaan yang belum ada jawaban, maka akan dijalankan RHS untuk bertanya. Setelah jawaban didapat, maka hasil jawaban akan disimpan ke dalam fakta answer.

5. Buat rule untuk pertanyaan no 1



```

44 (defrule 1-yes-2
45   (answer (id 1) (answer "yes"))
46   (question (id 2) (text ?text))
47   =>
48   (assert (ask (id 2) (text ?text))))
49
50 (defrule 1-no-3
51   (answer (id 1) (answer "no"))
52   (question (id 3) (text ?text))
53   =>
54   (assert (ask (id 3) (text ?text))))

```

Penjelasan:

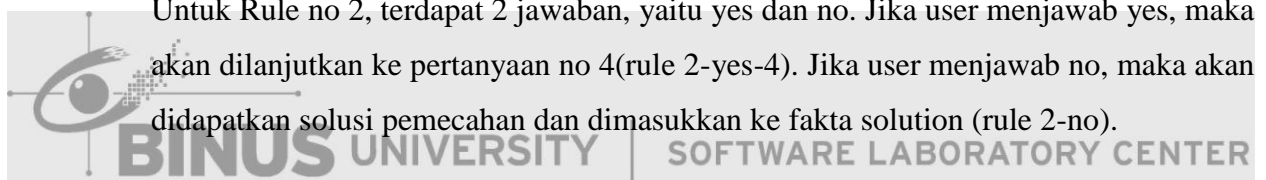
Rule dibuat 2 jenis, dimana jika user menjawab “yes”, maka akan dilanjutkan ke pertanyaan no 2. Jika user menjawab “no”, maka akan dilanjutkan ke pertanyaan no 3.

6. Contoh lain untuk rule no 2 dan 3

```

56 (defrule 2-no
57   (answer (id 2) (answer "no"))
58   =>
59   (assert (solution (text "There is trouble with the video"))))
60
61 (defrule 2-yes-4
62   (answer (id 2) (answer "yes"))
63   (question (id 4) (text ?text))
64   =>
65   (assert (ask (id 4) (text ?text))))
66
67 (defrule 3-yes-5
68   (answer (id 3) (answer "yes"))
69   (question (id 5) (text ?text))
70   =>
71   (assert (ask (id 5) (text ?text))))


```

Penjelasan:


Untuk Rule no 2, terdapat 2 jawaban, yaitu yes dan no. Jika user menjawab yes, maka akan dilanjutkan ke pertanyaan no 4(rule 2-yes-4). Jika user menjawab no, maka akan didapatkan solusi pemecahan dan dimasukkan ke fakta solution (rule 2-no).

Untuk Rule no 3, terdapat 1 jawaban yaitu yes. Jika user menjawab yes, maka akan dilanjutkan ke pertanyaan no 5. Jika user menjawab no, maka program akan berhenti dan tidak ditemukan solusi.


7. Lanjutkan untuk semua rule untuk forward chaining.



```

73 (defrule 4-yes-6
74   (answer (id 4) (answer "yes"))
75   (question (id 6) (text ?text))
76   =>
77   (assert (ask (id 6) (text ?text))))
78
79 (defrule 4-no
80   (answer (id 4) (answer "no"))
81   =>
82   (assert (solution (text "Replace the power supply"))))
83
84 (defrule 5-yes-9
85   (answer (id 5) (answer "yes"))
86   (question (id 9) (text ?text))
87   =>
88   (assert (ask (id 9) (text ?text))))
89
90 (defrule 5-no
91   (answer (id 5) (answer "no"))
92   =>
93   (assert (solution (text "Replace switch or
94                     substitute front panel reset button"))))
95
96 (defrule 6-yes
97   (answer (id 6) (answer "yes"))
98   =>
99   (assert (solution (text "There is trouble with motherboard"))))
100
101 (defrule 6-no-7
102   (answer (id 6) (answer "no"))
103   (question (id 7) (text ?text))
104   =>
105   (assert (ask (id 7) (text ?text))))
106
107 (defrule 7-yes
108   (answer (id 7) (answer "yes"))
109   =>
110   (assert (solution (text "Remove and retry if it works,
111                     then power supply is not enough." ))))
112
113 (defrule 7-no-8
114   (answer (id 7) (answer "no"))
115   (question (id 8) (text ?text))
116   =>
117   (assert (ask (id 8) (text ?text))))
118
119 (defrule 8-yes-10
120   (answer (id 8) (answer "yes"))
121   (question (id 10) (text ?text))
122   =>
123   (assert (ask (id 10) (text ?text))))

```



```

125 (defrule 8-no-11
126   (answer (id 8) (answer "no"))
127   (question (id 11) (text ?text))
128   =>
129   (assert (ask (id 11) (text ?text))))
130
131 (defrule 9-yes-12
132   (answer (id 9) (answer "yes"))
133   (question (id 12) (text ?text))
134   =>
135   (assert (ask (id 12) (text ?text))))
136
137 (defrule 12-yes
138   (answer (id 12) (answer "yes"))
139   =>
140   (assert (solution (text "Replace the power supply"))))
141
142 (defrule 12-no
143   (answer (id 12) (answer "no"))
144   =>
145   (assert (solution (text "There is trouble with motherboard"))))
146
147 (defrule 10-yes
148   (answer (id 10) (answer "yes"))
149   =>
150   (assert (solution (text "There is trouble with hard drive"))))
151
152 (defrule 10-no
153   (answer (id 10) (answer "no"))
154   =>
155   (assert (solution (text "There is trouble with motherboard" ))))
156
157 (defrule 11-yes
158   (answer (id 11) (answer "yes"))
159   =>
160   (assert (solution (text "Defective Power Suply or cable"))))
161
162 (defrule 11-no-13
163   (answer (id 11) (answer "no"))
164   (question (id 13) (text ?text))
165   =>
166   (assert (ask (id 13) (text ?text))))
167
168 (defrule 13-yes
169   (answer (id 13) (answer "yes"))
170   =>
171   (assert (solution (text "Replace the power supply"))))

```


8. Buatlah rule untuk mencetak solusi

```

171 (defrule print-solution
172   "If there is a solution, print it"
173   (solution (text ?t))
174   =>
175   (printout t ?t crlf))
176
177 (defrule no-solution
178   "If there is no solution"
179   (not (solution))
180   =>
181   (printout t "There is no solution given. Consult your technician." crlf))

```

Penjelasan:

Rule print-solution akan dijalankan jika solusi sudah ditemukan. Sedangkan rule “no-solution” dijalankan jika tidak ditemukan solusi dari jawaban yang diberikan user.

Cara 2 : Backward Chaining

Untuk menggunakan *backward chaining*, pengetahuan sebaiknya dibuat dalam matriks, supaya dapat menggambarkan bagaimana suatu solusi didapat.

	1	2	3	4	5	6	7	8	9	10	11	12	13
video trouble	yes	no											
power supply	yes	yes		no									
frontpanel	no		yes		no								
motherboard	yes	yes		yes		yes							
remove&retry	yes	yes		yes		no	yes						
replace pow	no		yes		yes				yes			yes	
motherboard	no		yes		yes				yes			no	
motherboard	yes	yes		yes		no	no	yes		no			
harddrive	yes	yes		yes		no	no	yes		yes			
def pow sup	yes	yes		yes		no	no	no			yes		
replace pow	yes	yes		yes		no	no	no			no		yes

Penjelasan:

Pada bagian kiri, adalah solusi yang ada. Pada bagian atas adalah id untuk masing-masing pertanyaan.

Contoh:

- Untuk solusi video trouble, harus didapat jawaban yes untuk pertanyaan no 1 dan no untuk pertanyaan no 2
- Untuk solusi power supply, harus didapat jawaban yes untuk pertanyaan no 1 dan yes untuk pertanyaan no 2, dan no untuk pertanyaan no 4.

Langkah:

1. Tentukan jenis pencarian

```
1 (set-strategy depth)
```

Penjelasan:

Artinya pencarian akan dilakukan secara *depth* atau mendalam. Kita dapat mengubah *depth* dengan *breadth* jika kita ingin melakukan pencarian ke samping. Secara default, pencarian dilakukan secara *depth*, jika set-strategy tidak diberikan.

2. Buat deftemplate untuk question, ask, answer, dan solution

```
3 (deftemplate question
4   (slot id)
5   (slot text))
6
7 (deftemplate ask
8   (slot id)
9   (slot text))
10
11 (deftemplate solution
12   (slot text))
```

```


14 (deftemplate answer
15   (slot id)
16   (slot answer))
17 (do-backward-chaining answer)

```

Penjelasan:

Fungsi do-backward-chaining answer artinya program akan melakukan backward chaining untuk fakta answer. Artinya, jika pada suatu rule membutuhkan fakta answer untuk mencapai RHS, maka rule engine akan mengusahakan untuk mendapatkan fakta answer itu.

3. Buat deffacts untuk pertanyaan-pertanyaan yang disediakan beserta idnya.



```

19 (def facts questions
20   "list of questions"
21   (question (id 1) (text "Does computer turn on?"))
22   (question (id 2) (text "Anything on screen?"))
23   (question (id 3) (text "Adequate power source and environment?"))
24   (question (id 4) (text "Computer boots every time?"))
25   (question (id 5) (text "Power switch working?"))
26   (question (id 6) (text "Does computer beep?"))
27   (question (id 7) (text "New part added?"))
28   (question (id 8) (text "Do you hear hard drive spin up?"))
29   (question (id 9) (text "Power supply to motherboard correct?"))
30   (question (id 10) (text "PC can boot from CD or floppy?"))
31   (question (id 11) (text "Try another power IDE cable & slot. Does it work?"))
32   (question (id 12) (text "Test Power supply. Is it damaged?"))
33   (question (id 13) (text "Does drive work in another PC?"))

```

4. Buat defrule untuk solusi video trouble

```

35 (defrule video-trouble
36   (answer (id 1) (answer "yes"))
37   (answer (id 2) (answer "no"))
38   =>
39   (assert (solution (text "There is trouble with video"))))

```

Penjelasan:

Untuk mendapatkan solusi video trouble, kita menambahkan kondisi answer untuk id 1 answer yes dan id 2 answer no. Jika kondisi tersebut terpenuhi, maka solusi akan ditambahkan.

5. Contoh beberapa rule yang lain

```
41 (defrule power-supply
42   (or
43     (and (answer (id 1) (answer "yes"))
44          (answer (id 2) (answer "yes"))
45          (answer (id 4) (answer "no")))
46     (and (answer (id 1) (answer "no"))
47          (answer (id 3) (answer "yes"))
48          (answer (id 5) (answer "yes"))
49          (answer (id 9) (answer "yes"))
50          (answer (id 12) (answer "yes")))
51     (and (answer (id 1) (answer "yes"))
52          (answer (id 2) (answer "yes"))
53          (answer (id 4) (answer "yes"))
54          (answer (id 6) (answer "no"))
55          (answer (id 7) (answer "no"))
56          (answer (id 8) (answer "no"))
57          (answer (id 11) (answer "no"))
58          (answer (id 13) (answer "yes"))))
59   =>
60   (assert (solution (text "Replace the power supply"))))
```

```

62 (defrule front-panel-reset-button
63   (answer (id 1) (answer "no"))
64   (answer (id 3) (answer "yes"))
65   (answer (id 5) (answer "no"))
66   =>
67   (assert (solution (text "Replace switch or substitute
68     front panel reset button"))))
69
70 (defrule motherboard
71   (or
72     (and (answer (id 1) (answer "yes"))
73          (answer (id 2) (answer "yes"))
74          (answer (id 4) (answer "yes"))
75          (answer (id 6) (answer "yes")))
76     (and (answer (id 1) (answer "no"))
77          (answer (id 3) (answer "yes"))
78          (answer (id 5) (answer "yes"))
79          (answer (id 9) (answer "yes"))
80          (answer (id 12) (answer "no")))
81     (and (answer (id 1) (answer "yes"))
82          (answer (id 2) (answer "yes"))
83          (answer (id 4) (answer "yes"))
84          (answer (id 6) (answer "no"))
85          (answer (id 7) (answer "no"))
86          (answer (id 8) (answer "yes"))
87          (answer (id 10) (answer "no"))))
88   =>
89   (assert (solution (text "There is motherboard trouble"))))
90
91 (defrule remove-and-retry-part
92   (answer (id 1) (answer "yes"))
93   (answer (id 2) (answer "yes"))
94   (answer (id 4) (answer "yes"))
95   (answer (id 6) (answer "no"))
96   (answer (id 7) (answer "yes"))
97   =>
98   (assert (solution (text "Remove and retry if it works,
99     then power supply is not enough.")))
100
101 (defrule hard-drive-trouble
102   (answer (id 1) (answer "yes"))
103   (answer (id 2) (answer "yes"))
104   (answer (id 4) (answer "yes"))
105   (answer (id 6) (answer "no"))
106   (answer (id 7) (answer "no"))
107   (answer (id 8) (answer "yes"))
108   (answer (id 10) (answer "yes"))
109   =>
110   (assert (solution (text "There is harddrive trouble"))))
111
112 (defrule defective-power-supply-or-cable
113   (answer (id 1) (answer "yes"))
114   (answer (id 2) (answer "yes"))
115   (answer (id 4) (answer "yes"))
116   (answer (id 6) (answer "no"))
117   (answer (id 7) (answer "no"))
118   (answer (id 8) (answer "no"))
119   (answer (id 11) (answer "yes"))
120   =>
121   (assert (solution (text "Defective Power Supply or cable"))))

```

Penjelasan:

Untuk rule front-panel, terdapat 3 kondisi untuk mendapatkan solusi, yaitu id 1 answer no, id 3 answer yes, dan id 5 answer no. Jika program mendapatkan answer no untuk id 1, maka program akan melanjutkan ke pertanyaan id 3 untuk mendapatkan jawaban. Rule motherboard bisa didapat dengan kondisi lebih dari 1. Jika itu terjadi, kita dapat menggunakan operasi **and** atau **or**.

6. Buat rule untuk menangani pertanyaan yang membutuhkan jawaban

```
123 (defrule need-to-ask
124     "If there is a need to an answer, then ask it"
125     (need-answer (id ?id))
126     (question (id ?id) (text ?text))
127     =>
128     (assert (ask (id ?id) (text ?text))))
```

Rule ini akan dijalankan jika program membutuhkan jawaban untuk id tertentu. Jika fakta question sesuai, maka akan ditanyakan. Fungsi need-answer disesuaikan dengan backward chaining yang dilakukan. Format: (need-<fact>)

7. Buat Rule untuk bertanya

```
130 (defrule ask-rule
131     "If there are ask facts, then ask. Remove it after use."
132     ?i <- (ask (id ?id) (text ?text))
133     =>
134     (bind ?answer "")
135     (while (and (neq (str-compare ?answer "yes") 0)
136                (neq (str-compare ?answer "no") 0)) do
137         (printout t ?text " [yes/no] : ")
138         (bind ?answer (readline)))
139     (assert (answer (id ?id) (answer ?answer)))
140     (retract ?i))
```

Penjelasan:

Ask-rule digunakan untuk bertanya suatu pertanyaan dengan id ?id dan text ?text. Jika rule selesai dijalankan, maka pertanyaan akan dihapus. Fungsi yang digunakan adalah retract.

8. Buat Rule untuk mencetak solusi

```

142 (defrule print-solution
143   "If there is a solution, print it"
144   (solution (text ?t))
145   =>
146   (printout t ?t crlf))
147
148 (defrule print-no-solution
149   "If there is no any solution, print it"
150   ; for breadth strategy
151   ; (declare (salience -100))
152   (not (solution))
153   =>
154   (printout t "There is no solution given. Consult your technician." crlf))
155
156 (reset)
157 (run)

```

Penjelasan:

Jika solusi sudah didapatkan, maka solusi akan dicetak. Jika solusi tidak ditemukan, maka tampilkan pesan bahwa solusi tidak ada.

Rule untuk print-no-solution tidak akan dijalankan jika pencariannya menggunakan strategy breadth. Oleh karenanya, rule di atas perlu ditambahkan salience, menjadi:

```

(defrule print-no-solution
  "If there is no any solution, print it"
  (declare (salience -100))
  (not (solution))
  =>
  (printout t "There is no solution given. Consult your technician." crlf))

```

(declare (salience -100)) artinya memberikan salience sebesar -100 untuk rule tersebut. Semakin sedikit salience, semakin terakhir akan dijalankan oleh rule engine.

9. Jalankan fungsi reset dan run

```

156 (reset)
157 (run)

```

Chapter 05

Build an Expert System



5.1. Rete

Untuk menghubungkan Java dengan Jess, maka gunakan objek Rete. Rete adalah suatu objek di Java yang memiliki *working memory* sekaligus *rule engine* sehingga dapat menyimpan semua fakta, rules, aktivasi, dan dapat menjalankan rule tersebut.

Class Rete memiliki beberapa fungsi penting, yaitu:

1. batch (String filename)
digunakan untuk menjalankan file dalam bahasa Jess.
2. eval (String cmd)
Digunakan untuk mengeksekusi ekspresi Jess pada *global context*.
3. reset()
Digunakan untuk menjalankan perintah reset pada rule engine.
4. run()
digunakan untuk menjalankan perintah run pada rule engine
5. runQueryStar (String name, ValueVector param)
digunakan untuk mengaktifasi defquery dengan nama “name” dan parameter “param” dan mengembalikan hasilnya.

Class ValueVector adalah representasi dari list/multiset di Jess. Misalnya, membuat list (“aa” “bb” “cc”) dilakukan dengan cara:

```
ValueVector v = new ValueVector().add("aa").add("bb").add("cc");
```

5.2. Melakukan query terhadap Working Memory

Untuk melakukan query terhadap working memory diperlukan 3 step:

1. Membuat query
Query dibuat dengan menggunakan defquery.
Format defquery:
(defquery <nama_defquery>
 (declare (<var1> <var2> <var3> <...>))
 <facts>
)

Contoh defquery:

```
(defquery search-by-name
  "Mencari orang berdasarkan nama"
  (declare (variables ?n))
  (person (name ?n) (age ?a)))
```

2. Memanggil query

Query dipanggil dengan menggunakan fungsi runQueryStar. Contoh pemanggilan defquery di atas untuk mencari fakta person bernama “Budi”:

```
Rete engine = new Rete();
QueryResult res = engine.runQueryStar("search-by-name", new ValueVector().add("Budi"));
```

3. Menggunakan hasil query

Setelah hasil query ditampung pada QueryResult, maka semua hasilnya bisa ditampilkan dengan menggunakan looping.

```
while(res.next()){
    System.out.println(res.getString("n") + " " + res.get("a"));
}
```

Res.next() digunakan untuk berpindah ke result yang berikutnya.

Res.getString(“n”) digunakan untuk mengambil attribute “n” (yang merupakan String) pada result.

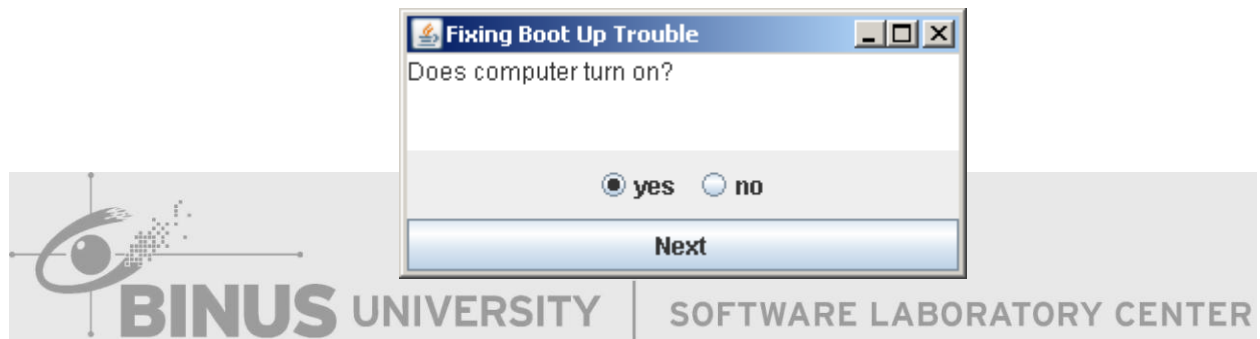
Res.get(“a”) digunakan untuk mengambil attribute “a” pada result.

5.3. Exercise

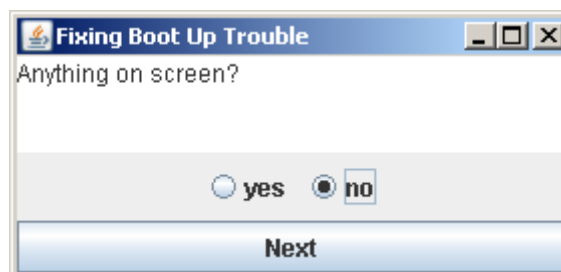
Berdasarkan knowledge base dari chapter sebelumnya, buatlah program “Fixing Boot Up Trouble” dengan matriks pengetahuan sebagai berikut.

	1	2	3	4	5	6	7	8	9	10	11	12	13
video trouble	yes	no											
power supply	yes	yes		no									
frontpanel	no		yes		no								
motherboard	yes	yes		yes		yes							
remove&retry	yes	yes		yes		no	yes						
replace pow	no		yes		yes				yes			yes	
motherboard	no		yes		yes				yes			no	
motherboard	yes	yes		yes		no	no	yes		no			
harddrive	yes	yes		yes		no	no	yes		yes			
def pow sup	yes	yes		yes		no	no	no			yes		
replace pow	yes	yes		yes		no	no	no			no		yes

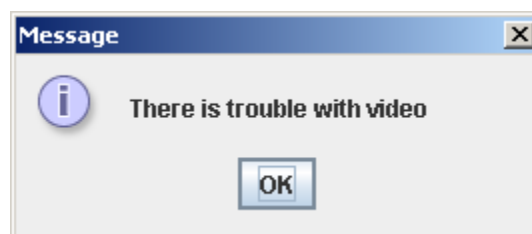
Tampilan program diperlihatkan di bawah.



Jika user menjawab yes, maka akan dilanjutkan seperti berikut:



Jika user menjawab no, maka akan didapatkan solusi dan ditampilkan dalam **messageBox**



Answer:

Program akan menggunakan **backward Chaining**.

Penjelasan solusi masih tetap sama dengan chapter sebelumnya.

Langkah:

1. Kita menggunakan file clp pada chapter 4 dan menambah isinya.
2. Hapus defrule **ask-rule**, **print-solution**, dan **print-no-solution** karena hasil tidak dicetak dalam console, tetapi akan dipindahkan ke GUI melalui defquery. Hapus juga fungsi (**reset**) dan (**run**).
3. Tambahkan deftemplate remove-ask



Penjelasan:

Deftemplate ini digunakan untuk menambahkan fakta berisi id pertanyaan yang ingin dihapus.

4. Tambahkan defquery find-ask dan find-solution

```
22 (defquery find-ask
23   "retrieve ask facts"
24   (declare (variables ))
25   (ask (id ?id) (text ?text)))
26
27 (defquery find-solution
28   "retrieve solution facts"
29   (declare (variables ))
30   (solution (text ?text)))
```

Penjelasan:

Defquery find-ask ini digunakan untuk mencari pertanyaan yang tersedia. Sedangkan defquery find-solution digunakan untuk mencari solusi yang ada

5. Buat defrule untuk remove-ask

```
141 (defrule remove-ask
142   "rule-to-remove-ask"
143   ?i <- (remove-ask (id ?id))
144   ?j <- (ask (id ?id))
145   =>
146   (retract ?i)
147   (retract ?j))
```

Penjelasan:

Rule ini digunakan untuk menghapus fakta (ask) yang sudah ditanyakan. Fakta (remove-ask) akan di-assert ketika pertanyaan sudah ditanyakan. Rule ini diperlukan agar setiap kali defquery dipanggil untuk mencari pertanyaan yang ada pada *working memory*, maka hasil pencarian pasti selalu berjumlah 1, yaitu pertanyaan baru yang belum ditanyakan.

Langkah pengerjaan untuk GUI

6. Buat sebuah class dengan nama FixingComputer dan tambahkan **extends JFrame**

```
20 public class FixingComputer extends JFrame{
```

7. Buat variable dan object yang dipakai

```

21     Rete engine = new Rete();
22     QueryResult res, sol;
23     Value val;
24     JTextArea textArea = new JTextArea(3, 25);
25     JButton nextButton = new JButton("Next");
26     JPanel panel = new JPanel();
27     JRadioButton yesbutton = new JRadioButton("yes", true);
28     JRadioButton noButton = new JRadioButton("no");
29     ButtonGroup bg = new ButtonGroup();

```

Penjelasan:

- Rete engine = new Rete() digunakan untuk membuat objek Rete.
- QueryResult digunakan untuk menampung hasil query. Value digunakan untuk menampung tipe data jess.
- JTextArea, JButton, JPanel, JRadioButton, dan ButtonGroup adalah komponen untuk menampilkan berbagai komponen pada GUI.

8. Buat Constructor dari class Utama untuk pembuatan awal

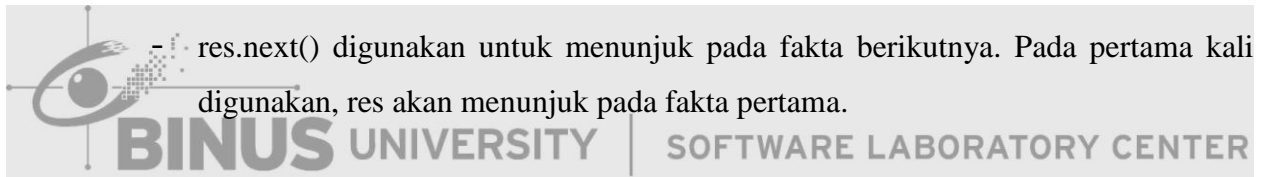
```

30     public FixingComputer() throws JessException {
31         /*setting Rete */
32         engine.batch("Chapter05.clp");
33         engine.reset();
34         engine.run();
35         /* end Setting Rete*/
36
37         // add textArea
38         textArea.setWrapStyleWord(true);
39         textArea.setEditable(false);
40         res = engine.runQueryStar("find-ask", new ValueVector());
41         if(res.next()){
42             textArea.setText(res.getString("text"));
43             val = res.get("id");
44         }
45         add(textArea, "North");
46
47         /* Setting panel */
48         bg.add(yesbutton);
49         bg.add(noButton);
50         panel.add(yesbutton);
51         panel.add(noButton);
52         add(panel, "Center");
53         /*end set panel*/


```

Penjelasan:

- Engine.batch("Chapter05.clp") digunakan untuk menjalankan codingan pada file "Chapter05.clp". Secara umum, batch digunakan untuk mempersiapkan *working memory*.
- Engine.reset() dan engine.run() digunakan untuk menjalankan fungsi reset dan run.
- engine.runQueryStar("find-ask", new ValueVector()) digunakan menjalankan query "find-ask" (mencari fakta ask yang ada pada *working memory*). New ValueVector() digunakan untuk memberikan parameter untuk query "find-ask". Karena tidak dibutuhkan parameter pada "find-ask", maka isi ValueVector adalah kosong. Hasil query ditampung pada res dengan tipe data Query Result. Berarti, res berisi semua fakta "ask".

- 
- res.next() digunakan untuk menunjuk pada fakta berikutnya. Pada pertama kali digunakan, res akan menunjuk pada fakta pertama.
 - Res.getString("text") digunakan untuk mengambil attribute "text" pada fakta ask, sedangkan res.get("id") digunakan untuk mengambil attribute "id" pada fakta ask. Isi text akan dimasukkan ke textArea, sedangkan id akan disimpan di variable val.

9. Buat ActionListener untuk Button



```

55 nextButton.addActionListener(new ActionListener() {
56     public void actionPerformed(ActionEvent arg0) {
57         String text;
58         if(yesbutton.isSelected())text = "yes";
59         else text = "no";
60         try {
61             engine.eval("(assert (answer (id "+ val+" ) (answer \"\"+ text +\"\\\")))");
62             engine.run();
63             engine.eval("(assert (remove-ask (id "+ val + ")))");
64             engine.run();
65             res = engine.runQueryStar("find-ask", new ValueVector());
66             if(res.next()){ // if there are more question
67                 textArea.setText(res.getString("text"));
68                 val = res.get("id");
69             } // if there is no questions left.
70         } else{
71             // retrieve the solutions
72             sol = engine.runQueryStar("find-solution", new ValueVector());
73             String result = "";
74             while(sol.next()){
75                 result += sol.getString("text");
76                 result += "\n";
77             }
78             if(result.equals("")){ // if there is no solution
79                 JOptionPane.showMessageDialog(null, "There is no solution. " +
80                     "Contact your technician");
81             }
82             else{ // if there is at least one solution
83                 JOptionPane.showMessageDialog(null, result);
84             }
85             // back to initialization
86             engine.reset();
87             engine.run();
88             res = engine.runQueryStar("find-ask", new ValueVector());
89             if(res.next()){
90                 textArea.setText(res.getString("text"));
91                 val = res.get("id");
92             }
93             yesbutton.setSelected(true);
94             // end initialization
95         }
96     } catch (JessException e) {
97         e.printStackTrace();
98     }
99 }
100 });

```

Penjelasan:

- Program akan melihat jawaban user. Jawaban user akan disimpan di variable text.

- Setelah itu, program akan memasukkan jawaban, me-remove fakta ask dengan id pertanyaan tersebut, serta menampilkan pertanyaan selanjutnya jika pertanyaan masih ada.
- Jika Pertanyaan tidak tersisa, maka akan dicari solusi. Jika solusi ditemukan, maka tampilkan messageDialog.

```
sol = engine.runQueryStar("find-solution", new ValueVector());
String result = "";
while(sol.next()){
    result += sol.getString("text");
    result += "\n";
}
if(result.equals("")){ // if there is no solution
    JOptionPane.showMessageDialog(null, "There is no solution. " +
        "Contact your technician");
}
else{ // if there is at least one solution
    JOptionPane.showMessageDialog(null, result);
}
```

- Setelah solusi ditemukan, program akan mengulang dari awal

```
engine.reset();
engine.run();
res = engine.runQueryStar("find-ask", new ValueVector());
if(res.next()){
    textArea.setText(res.getString("text"));
    val = res.get("id");
}
yesbutton.setSelected(true);
```

10. Masukkan button dan tampilkan frame.

```
101         add(nextButton, "South");
102
103         setTitle("Fixing Boot Up Trouble");
104         setDefaultCloseOperation(EXIT_ON_CLOSE);
105         pack();
106         setLocationRelativeTo(null);
107         setVisible(true);
108     }
109
110     public static void main(String[] args) throws JessException {
111         new FixingComputer();
112     }
113 }
```