

# **Penerapan Design Pattern dalam Rancang Ulang Sistem Informasi Penerimaan Mahasiswa Baru PI Del**

## **Tugas Akhir**

Disampaikan sebagai Bagian dari Persyaratan Kelulusan Diploma 3  
Program Studi Teknik Informatika

### **Oleh :**

Julianti Munthe	11105024
Franciscus M Sianturi	11105039
Irma Sari Barus	11105078



**Politeknik Informatika Del  
2008**

## **Lembar Pengesahan Tugas Akhir**

**Politeknik Informatika Del**

### **Penerapan Design Pattern dalam Rancang Ulang Sistem Informasi Penerimaan Mahasiswa Baru PI Del**

**Oleh:**

Julianti Munthe	11105024
Franciscus M Sianturi	11105039
Irma Sari Barus	11105078

**Dinyatakan memenuhi syarat dan karenanya disetujui dan disahkan sebagai  
Laporan Tugas Akhir Diploma 3  
Program Studi Teknik Informasi**

Pembimbing 1,

Pembimbing 2,

Ir. Hira Laksmiwati, M.Sc.  
NIP : 131471329

Henry Edison Sitorus, S.T.  
NIDN : 0123048001

## **Prakata**

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa atas perlindungan dan rahmat-Nya, sehingga penulis bisa menyelesaikan laporan tugas akhir ini dengan baik. Tugas akhir yang berjudul "**Penerapan Design Pattern dalam Rancang Ulang Sistem Informasi Penerimaan Mahasiswa Baru PI Del**" merupakan syarat kelulusan Diploma 3 yang telah ditempuh selama tiga tahun di Politeknik Informatika Del.

Penulis mengucapkan terima kasih kepada dosen penguji, Bapak Ari Wibowo, S.T dan Ibu Ir. Hira L. Zoro, Msc sekaligus sebagai dosen pembimbing atas saran yang membangun sehingga laporan tugas akhir dapat diselesaikan dengan baik.

Penulis juga mengucapkan terima kasih kepada dosen pembimbing, Bapak Henry Edison Sitorus,S.T yang telah memberikan motivasi, bimbingan dan arahan kepada penulis sejak awal penyusunan sampai selesaiannya laporan tugas akhir ini.

Selanjutnya, penulis juga mengucapkan terimakasih yang sedalam-dalamnya kepada Direktur Politeknik Informatika Del Ibu Inggriani Liem atas bimbingan dan telah meluangkan waktu untuk memberikan pengarahan kepada penulis dalam menyelesaikan tugas akhir.

Melalui laporan tugas akhir ini, pembaca diharapkan dapat mengetahui dan memahami bagaimana merancang suatu sistem perangkat lunak berorientasi objek dengan menerapkan *design pattern*. Selain itu pembaca juga diharapkan mengetahui lebih jelas mengenai rancangan Sistem Informasi Penerimaan Mahasiswa Baru Politeknik Informatika Del (SI PMB PI Del) dengan menggunakan metode berorientasi objek.

Sitoluama, 2 September 2008

Julianti Munthe	11105024
Franciscus M Sianturi	11105039
Irma Sari Barus	11105078

## **Abstrak**

*Design pattern* atau pola rancangan adalah sebuah istilah dalam rekayasa perangkat lunak yang mengacu kepada solusi umum yang dapat digunakan secara berulang.

Pada laporan tugas akhir ini dibahas mengenai penerapan *design pattern* untuk merancang ulang Sistem Informasi Penerimaan Mahasiswa Baru Politeknik Informatika Del (SI PMB PI Del), yang dinamakan SI PMB TA2008, setelah mempelajari aplikasi yang sudah ada yaitu SI PMB PI Del dan hasil Proyek Akhir 2 (SI PMB PA2). SI PMB PI Del adalah versi aplikasi yang tidak dilengkapi dengan dokumentasi sehingga tidak dapat digunakan sebagai acuan dalam melakukan rancang ulang dan hanya dapat memberikan gambaran mengenai interaksi dengan *user*. SI PMB PA2 diacu struktur tabelnya karena dilengkapi dengan dokumentasi.

*Design pattern* yang diterapkan dalam rancangan SI PMB TA2008 dipilih berdasarkan kesesuaian masalah yang ada pada SI PMB PI Del dan SI PMB PA2. Hasil akhir laporan ini adalah rancangan SI PMB TA2008 yang telah menerapkan 3 (tiga) *design pattern* yaitu *singleton*, *mediator* dan *decorador pattern*.

Kata kunci : ***Design pattern, rancangan, SI PMB***

## DAFTAR ISI

Prakata.....	1
Abstrak .....	2
Bab 1 Pendahuluan.....	5
1.1    Latar Belakang.....	5
1.2    Tujuan.....	5
1.3    Lingkup.....	5
1.4    Batasan.....	5
1.5    Pendekatan.....	6
1.6    Sistematika Penyajian.....	7
Bab 2 Tinjauan Pustaka.....	8
2.1    Object Oriented Analysis and Design (OOAD) .....	8
2.1.1    Menentukan use case diagram .....	8
2.1.2    Menentukan daerah pemodelan .....	9
2.1.3    Menentukan interaksi diagram.....	9
2.1.4    Menentukan <i>class diagram</i> .....	10
2.2    Design pattern.....	10
2.2.1    Creational pattern.....	12
2.2.2    Structural Pattern .....	17
2.2.3    Behavioral Pattern.....	24
2.3    Studi Aplikasi PMB.....	34
2.3.1    Studi Aplikasi SI PMB PI Del .....	35
2.3.2    Studi Aplikasi SI PMB PA2 .....	36
Bab 3 Analisis .....	37
3.1    Analisis dan Spesifikasi Kebutuhan .....	37
3.2    Arsitektur Aplikasi Web .....	40
3.3    Rancangan Kelas Tanpa Pattern .....	41
3.4    Penerapan Design Pattern SI PMB TA 2008.....	42
3.4.1    Creational Patterns .....	43
3.4.2    Structural Patterns .....	43
3.4.3    Behavioral Patterns .....	44
Bab 4 Penerapan Design Pattern.....	47
4.1    Singleton pattern.....	47
4.2 <i>Decorator pattern</i> .....	48
4.3    Mediator pattern .....	48
Bab 5 Kesimpulan dan Saran .....	50
5.1    Kesimpulan.....	50
5.2    Saran .....	50
Daftar Pustaka dan Rujukan.....	51
Daftar Pustaka.....	51
Rujukan.....	51
Lampiran A .....	52
Lampiran B .....	56
Lampiran C .....	57

## DAFTAR GAMBAR

Gambar 1 Tahapan dalam merancang .....	8
Gambar 2 Model <i>use case</i> .....	9
Gambar 3 Model domain untuk permainan dadu .....	9
Gambar 4 <i>Sequence diagram</i> .....	10
Gambar 5 <i>Class diagram</i> .....	10
Gambar 6 Struktur <i>abstract factory pattern</i> .....	13
Gambar 7 Struktur <i>builder pattern</i> .....	14
Gambar 8 Struktur <i>Factory Method Pattern</i> .....	15
Gambar 9 struktur <i>prototype pattern</i> .....	16
Gambar 10 struktur <i>singleton pattern</i> .....	17
Gambar 11 Struktur <i>Adapter pattern</i> .....	18
Gambar 12 Struktur <i>Bridge Pattern</i> .....	19
Gambar 13 Struktur <i>Composite Pattern</i> .....	19
Gambar 14 Struktur <i>Decorator Pattern</i> .....	20
Gambar 15 Struktur <i>Facade Pattern</i> .....	21
Gambar 16 Struktur <i>Flyweight Pattern</i> .....	22
Gambar 17 Struktur <i>Proxy Pattern</i> .....	23
Gambar 18 Struktur <i>Chain of Responsibility Pattern</i> .....	24
Gambar 19 Struktur <i>Command Pattern</i> .....	25
Gambar 20 Struktur <i>Interpreter Pattern</i> .....	26
Gambar 21 Struktur <i>Iterator Pattern</i> .....	27
Gambar 22 Struktur <i>Mediator Pattern</i> .....	28
Gambar 23 Struktur <i>Memento Pattern</i> .....	29
Gambar 24 Struktur <i>Observer Pattern</i> .....	30
Gambar 25 Struktur <i>State Pattern</i> .....	31
Gambar 26 Struktur <i>Strategy Pattern</i> .....	32
Gambar 27 Struktur <i>Template Method Pattern</i> .....	33
Gambar 28 Struktur <i>Visitor Pattern</i> .....	34
Gambar 29 Rancangan kelas SI PMB.....	42
Gambar 30 Diagram kelas yang menerapkan <i>singleton pattern</i> .....	47
Gambar 31 Diagram kelas yang menerapkan <i>decorator pattern</i> .....	48
Gambar 32 Diagram kelas yang menerapkan <i>mediator pattern</i> .....	49

## DAFTAR TABEL

Tabel 1 <i>Design pattern</i> berdasarkan tujuan .....	12
Tabel 2 Daftar spesifikasi fungsi SI PMB PI Del.....	35
Tabel 3 Karakteristik pengguna SI PMB PI Del.....	36
Tabel 4 Daftar tabel SI PMB PA2 .....	36
Tabel 5 Daftar spesifikasi fungsi SI PMB TA2008 .....	38

## **Lembar Pengesahan Tugas Akhir**

**Politeknik Informatika Del**

### **Penerapan Design Pattern dalam Rancang Ulang Sistem Informasi Penerimaan Mahasiswa PI Del**

**Oleh:**

Julianti Munthe 11105024

Franciscus M Sianturi 11105039

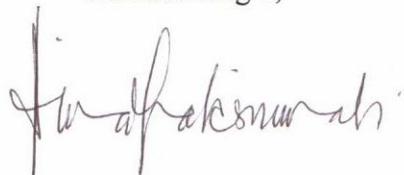
Irma Sari Br Barus 11105056

**Dinyatakan memenuhi syarat dan karenanya disetujui dan disahkan sebagai**

**Laporan Tugas Akhir Diploma 3**

**Program Studi Teknik Informasi**

Pembimbing 1,



Ir. Hira Laksmiwati, M.Sc.

Pembimbing 2,



Henry Edison Sitorus, S.T.

NIP : 131471329

NIDN : 0123048001

## Prakata

Puji syukur penulis ucapkan kepada Tuhan Yang Maha Esa atas perlindungan dan rahmat-Nya, sehingga penulis bisa menyelesaikan laporan tugas akhir ini dengan baik. Tugas akhir yang berjudul "**Penerapan Design Pattern dalam Rancang Ulang Sistem Informasi Penerimaan Mahasiswa Baru PI Del**" merupakan syarat kelulusan Diploma 3 yang telah ditempuh selama tiga tahun di Politeknik Informatika Del.

Penulis mengucapkan terima kasih kepada dosen penguji, Bapak Ari Wibowo, S.T dan Ibu Ir. Hira L. Zoro, Msc sekaligus sebagai dosen pembimbing atas saran yang membangun sehingga laporan tugas akhir dapat diselesaikan dengan baik.

Penulis juga mengucapkan terima kasih kepada dosen pembimbing, Bapak Henry Edison Sitorus,S.T yang telah memberikan motivasi, bimbingan dan arahan kepada penulis sejak awal penyusunan sampai selesaiannya laporan tugas akhir ini.

Selanjutnya, penulis juga mengucapkan terimakasih yang sedalam-dalamnya kepada Direktur Politeknik Informatika Del Ibu Inggriani Liem atas bimbingan dan telah meluangkan waktu untuk memberikan pengarahan kepada penulis dalam menyelesaikan tugas akhir.

Melalui laporan tugas akhir ini, pembaca diharapkan dapat mengetahui dan memahami bagaimana merancang suatu sistem perangkat lunak berorientasi objek dengan menerapkan *design pattern*. Selain itu pembaca juga diharapkan mengetahui lebih jelas mengenai rancangan Sistem Informasi Penerimaan Mahasiswa Baru Politeknik Informatika Del (SI PMB PI Del) dengan menggunakan metode berorientasi objek.

Sitoluama, 2 September 2008

Julianti Munthe	11105024
Franciscus M Sianturi	11105039
Irma Sari Barus	11105078

## **Abstrak**

*Design pattern* atau pola rancangan adalah sebuah istilah dalam rekayasa perangkat lunak yang mengacu kepada solusi umum yang dapat digunakan secara berulang.

Pada laporan tugas akhir ini dibahas mengenai penerapan *design pattern* untuk merancang ulang Sistem Informasi Penerimaan Mahasiswa Baru Politeknik Informatika Del (SI PMB PI Del), yang dinamakan SI PMB TA2008, setelah mempelajari aplikasi yang sudah ada yaitu SI PMB PI Del dan hasil Proyek Akhir 2 (SI PMB PA2). SI PMB PI Del adalah versi aplikasi yang tidak dilengkapi dengan dokumentasi sehingga tidak dapat digunakan sebagai acuan dalam melakukan rancang ulang dan hanya dapat memberikan gambaran mengenai interaksi dengan *user*. SI PMB PA2 diacu struktur tabelnya karena dilengkapi dengan dokumentasi.

*Design pattern* yang diterapkan dalam rancangan SI PMB TA2008 dipilih berdasarkan kesesuaian masalah yang ada pada SI PMB PI Del dan SI PMB PA2. Hasil akhir laporan ini adalah rancangan SI PMB TA2008 yang telah menerapkan 3 (tiga) *design pattern* yaitu *singleton*, *mediator* dan *decorador pattern*.

Kata kunci : ***Design pattern, rancangan, SI PMB***

## Bab 1 Pendahuluan

Pada bab pendahuluan diuraikan latar belakang, tujuan, batasan, pendekatan dan sistematika penyajian dalam menyelesaikan persoalan yang menjadi objek pembahasan dalam laporan tugas akhir.

### 1.1 Latar Belakang

Proses pengembangan suatu perangkat lunak, yaitu perancangan perangkat lunak berorientasi objek sulit dilakukan [ERR95]. Hal ini disebabkan karena harus menemukan objek-objek yang saling berkaitan dan harus menyusun objek-objek tersebut menjadi kelas-kelas dengan cara yang benar. Selain itu, kelas *interface*, hierarki *interface* dan hubungan diantaranya juga harus ditentukan.

Dengan menerapkan *design pattern*, menjadikan desain lebih fleksibel yang artinya desain mudah disesuaikan dengan masalah yang harus diselesaikan atau desain tidak berpengaruh besar terhadap perubahan. Selain itu, penerapan *design pattern* membuat rancangan dapat digunakan secara berulang dan bersifat modular. Dari kemudahan yang disediakan oleh *design pattern* tersebut, diterapkanlah *design pattern* pada suatu rancangan. Dalam laporan tugas akhir, *design pattern* diterapkan pada rancangan SI PMB TA2008.

### 1.2 Tujuan

Tujuan yang akan dicapai adalah menghasilkan rancangan SI PMB TA2008 yang menerapkan *design pattern*.

### 1.3 Lingkup

Lingkup penggerjaan laporan tugas akhir ini adalah membuat rancangan SI PMB TA2008 berdasarkan hasil analisis dengan mempelajari aplikasi SI PMB PI Del dan SI PMB PA2. Kemudian rancangan dimodelkan dengan *unified model language* (UML) yaitu *class diagram*. Setelah rancangan kelas selesai maka diterapkanlah *design pattern* yang sesuai dan digambarkan pada *class diagram*.

### 1.4 Batasan

Batasan yang diterapkan dalam penggerjaan laporan tugas akhir adalah hasil rancangan tidak dilanjutkan dengan implementasi. Hal tersebut dilakukan karena waktu penggerjaan TA yang singkat dan masih perlu pembelajaran terhadap *design pattern*.

## **1.5 Pendekatan**

Pendekatan yang dilakukan untuk memperoleh hasil dan kesimpulan dalam penggeraan tugas akhir adalah:

- 1. Studi Literatur**

Mengumpulkan bahan atau data yang berhubungan dengan perancangan yang menggunakan metoda berorientasi objek dan *design pattern*.

- 2. Studi SI PMB PIDEL**

Sebelum tahap analisis, kegiatan yang dilakukan adalah eksplorasi SI PMB PIDEL. Karena tidak ada *source code* dan dokumen yang dapat diacu, maka kegiatan tersebut dilakukan dengan mempelajari aplikasi yang dapat diakses pada <http://spmb.del.ac.id>

- 3. Studi SI PMB PA2**

Sebagai dasar tambahan dalam melakukan rancang ulang SI PMB PIDEL, dilakukan juga eksplorasi SI PMB PA2 untuk mempelajari struktur tabel sebagai inspirasi dalam penyusunan tabel dan kelas yang dibutuhkan dalam melakukan rancangan. SI PMB PA2 adalah aplikasi SI PMB yang dikembangkan oleh mahasiswa tingkat 2 sebagai hasil proyek akhir 2 (PA2).

- 4. Analisis**

Melakukan analisis kebutuhan SI PMB TA2008. Hasil analisis yaitu daftar fungsi SI PMB TA2008, *class diagram* sebelum menerapkan *design pattern* berdasarkan daftar fungsi, dan analisis penerapan *design pattern*.

- 5. Rancangan**

Pada tahap rancangan dihasilkan *class diagram* SI PMB TA2008 yang telah menerapkan *design pattern*. *Design pattern* yang diterapkan sesuai dengan hasil analisis terhadap penerapan *design pattern*.

## **1.6 Sistematika Penyajian**

Setelah bab satu diuraikan pada bab pendahuluan, maka bab selanjutnya diuraikan dengan sistematika sebagai berikut.

### **1. Bab 2 Tinjauan Pustaka**

Pada bab dua dibahas mengenai tinjauan pustaka yang dihimpun dari pustaka yang berkesesuaian untuk digunakan sebagai referensi dalam meyelesaikan rancangan SI PMB TA2008.

### **2. Bab 3 Analisis**

Pada bab tiga dibahas mengenai hasil analisis terhadap perangkat lunak SI PMB TA2008 antara lain spesifikasi fungsi, arsitektur aplikasi web, rancangan kelas tanpa *design pattern*, dan penerapan *design pattern*.

### **3. Bab 4 Penerapan *Design Pattern***

Pada bab empat dijelaskan mengenai hasil rancangan perangkat lunak SI PMB TA2008 sesudah menerapkan *design pattern*. Selain itu, juga dijelaskan pembahasan mengenai hasil rancangan tersebut.

### **4. Bab 5 Kesimpulan dan Saran**

Pada bab enam disampaikan kesimpulan dari hasil penggerjaan tugas akhir dan saran yang membangun.

### **5. Daftar pustaka dan rujukan**

Pada bab daftar pustaka dan rujukan diuraikan seluruh referensi yang digunakan sebagai bahan tulisan atau data rujukan dalam melengkapi penulisan laporan tugas akhir.

### **6. Lampiran**

Data lain yang berhubungan dengan penggerjaan laporan tugas akhir diikutsertakan sebagai lampiran.

## Bab 2 Tinjauan Pustaka

Pada bab tinjauan pustaka dijelaskan rangkuman informasi yang digunakan sebagai dasar teori dalam melakukan perancangan. Materi yang dicakup dalam tinjauan pustaka meliputi *object oriented analysis and design* (OOAD) dan *design pattern*.

Pembahasan OOAD diikutsertakan karena konsep OOAD diperlukan untuk mengidentifikasi kebutuhan dan untuk menghasilkan spesifikasi SI PMB dalam model yang dapat dimengerti.

### 2.1 Object Oriented Analysis and Design (OOAD)

*Object oriented analysis* (OOA) adalah tahap awal dalam merancang perangkat lunak melalui proses pemahaman terhadap masalah dan kebutuhan yang diinginkan. Tujuan pelaksanaan OOA adalah menentukan kelas-kelas yang muncul sesuai dengan kebutuhan.

Desain merupakan tahapan berikutnya setelah analisis dilakukan. Desain selalu didasarkan pada hasil analisis. Desain adalah proses pemecahan masalah dan menemukan solusi konseptual dan belum diimplementasikan. *Object oriented design* (OOD) merupakan penguraian model analisis untuk menghasilkan spesifikasi implementasi [RSP01].

*Object oriented analysis and design* (OOAD) adalah sebuah pendekatan rekayasa perangkat lunak yang memodelkan sebuah sistem sebagai kumpulan objek yang saling berinteraksi. Setiap objek menggambarkan entiti dari sistem yang dimodelkan, dan digolongkan menurut kelas, atribut, dan tingkah laku [RSP01].

Berikut diuraikan tahapan yang dilakukan dalam merancang suatu perangkat lunak berorientasi objek.

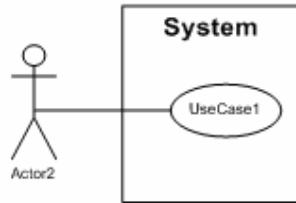


Gambar 1 Tahapan dalam merancang

#### 2.1.1 Menentukan use case diagram

*Use case diagram* menunjukkan kumpulan *use case* dan aktor. *Use case* menunjukkan rangkaian aksi yang dapat dilakukan sistem sedangkan aktor menunjukkan orang atau sistem lain yang berinteraksi dengan sistem yang sedang dimodelkan. Langkah-langkah

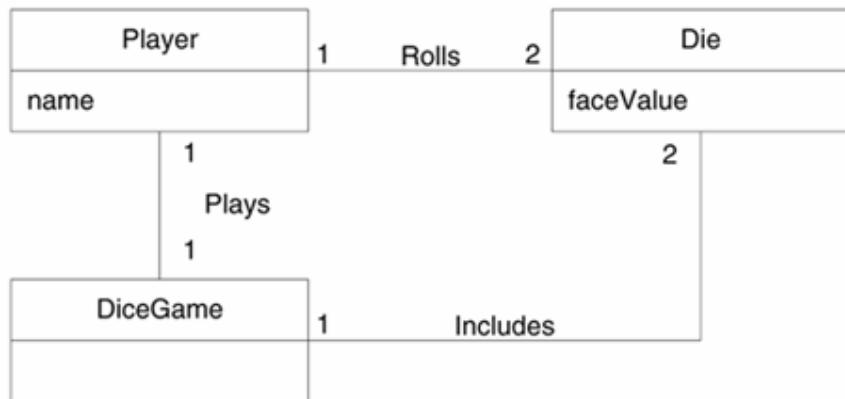
untuk mendefinisikan *use case diagram* adalah menentukan aktor, menentukan kumpulan *use case*, dan menggambarkan *use case diagram* [BES02].



Gambar 2 Model *use case*

### 2.1.2 Menentukan daerah pemodelan

OOA difokuskan pada pembuatan deskripsi suatu *domain* dari sudut pandang objek. Ada sebuah konsep pengenalan, atribut, dan hubungan yang perlu untuk dipertimbangkan. Berikut adalah contoh model *domain* [RSP01].

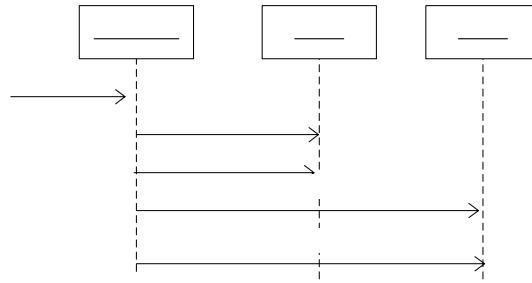


Gambar 3 Model *domain* untuk permainan dadu

Model ilustrasi pada Gambar 3 terdiri dari konsep-konsep yang penting antara lain Player, Die, dan DiceGame, dengan hubungan dan atributnya. Model *domain* merupakan sebuah konsep yang menggambarkan *domain* pada dunia nyata. Oleh sebab itu, model *domain* disebut sebagai model objek konseptual.

### 2.1.3 Menentukan interaksi diagram

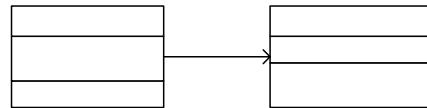
Interaksi diagram menggambarkan bagaimana sekumpulan objek berinteraksi. Diagram menunjukkan sejumlah objek dan pesan yang dilewatkan diantara objek tersebut. Berikut contoh *sequence diagram* yang mengilustrasikan sebuah desain perangkat lunak berorientasi objek melalui pengiriman pesan ke instansiasi kelas DiceGame dan Die [BJR99].



Gambar 4 Sequence diagram

#### 2.1.4 Menentukan *class diagram*

*Class diagram* adalah sebuah diagram yang menunjukkan sekumpulan kelas, *interface*, dan hubungan diantara kelas dan *interface*. Berikut adalah contoh *class diagram* [BJR99].



Gambar 5 Class diagram

## 2.2 Design pattern

*Design pattern* atau pola rancangan adalah sebuah istilah dalam rekayasa perangkat lunak yang mengacu kepada solusi umum yang dapat digunakan secara berulang kali untuk menyelesaikan masalah-masalah umum yang ditemukan dalam desain perangkat lunak.

*Design pattern* merupakan kumpulan dari aturan-aturan yang menentukan bagaimana menyelesaikan beberapa permasalahan dalam lingkungan pengembangan perangkat lunak khususnya dalam desain. *Design pattern* menggambarkan solusi sederhana dan sangat cocok untuk masalah tertentu dalam desain perangkat lunak berorientasi objek.

Menurut Chritoper Alexander [ERR95], “Setiap *pattern* menggambarkan suatu permasalahan yang terjadi berulang-ulang pada lingkungan kita, dan kemudian menggambarkan solusi dari masalah tersebut, sehingga solusi yang dihasilkan dapat digunakan secara berulang.

Penerapan *design pattern* pada suatu rancangan perangkat lunak tidak terlepas kaitannya dengan OOAD. *Design pattern* fokus pada desain, khususnya desain yang berorientasi objek. Desain berorientasi objek digambarkan dengan menggunakan metoda OOAD. *Design pattern* menggambarkan komunikasi antara objek dan kelas yang dibuat untuk memecahkan masalah desain secara umum dalam keadaan tertentu.

Suatu *design pattern* mengidentifikasi aspek utama suatu struktur desain agar *design pattern* tersebut berguna untuk membuat desain berorientasi objek yang mampu.

*Design pattern* dapat diklasifikasikan berdasarkan *purpose* yang mencerminkan apa yang dapat dilakukan oleh *pattern*. Klasifikasi *pattern* diuraikan pada Tabel 1 [ERR95].

Berdasarkan tujuannya, terdapat tiga *pattern* yaitu *creational*, *structural*, dan *behavioral* yang diuraikan sebagai berikut.

- Creational pattern*, fokus kepada proses pembuatan objek.
- Structural pattern*, menguraikan susunan objek atau kelas.
- Behavioral pattern*, menggolongkan kelas atau objek yang saling berhubungan.

**Tabel 1** *Design pattern* berdasarkan tujuan

Tujuan	Design pattern	Keterangan
<i>Creational</i>	<i>Abstract Factory</i>	Kumpulan objek.
	<i>Builder</i>	Gabungan objek yang dapat diciptakan.
	<i>Factory Method</i>	Sub kelas objek yang diinstansiasi.
	<i>Prototype</i>	Kelas objek yang diinstansiasi.
	<i>Singleton</i>	Instansiasi kelas.
<i>Structural</i>	<i>Adapter</i>	<i>Interface</i> untuk sebuah objek.
	<i>Bridge</i>	Implementasi sebuah objek.
	<i>Composite</i>	Struktur dan komposisi sebuah objek.
	<i>Decorator</i>	Tanggungjawab sebuah objek tanpa sub kelas.
	<i>Facade</i>	<i>Interface</i> untuk sebuah sub sistem.
	<i>Flyweight</i>	Penyimpanan objek.
	<i>Proxy</i>	Pengaksesan sebuah objek.
<i>Behavioral</i>	<i>Chain of Responsibility</i>	Objek yang memenuhi <i>request</i> .
	<i>Command</i>	Kapan dan bagaimana memenuhi <i>request</i> .
	<i>Interpreter</i>	<i>Grammar</i> dan <i>interpreter</i> tata bahasa.
	<i>Iterator</i>	Bagaimana kumpulan elemen di akses.
	<i>Mediator</i>	Bagaimana dan objek mana yang berinteraksi satu sama lain.
	<i>Memento</i>	Informasi private apa yang disimpan di luar objek dan kapan.
	<i>Observer</i>	Sekumpulan objek yang bergantung pada objek lain, dan bagaimana objek tersebut saling <i>meng-update</i> .
	<i>State</i>	<i>State</i> sebuah objek.
	<i>Strategy</i>	Sebuah algoritma.
	<i>Template Method</i>	Tahapan algoritma.
	<i>Visitor</i>	Operasi yang dapat digunakan objek tanpa mengganti kelas yang sudah ada.

### 2.2.1 Creational pattern

*Creational pattern* memisahkan proses instansiasi dan membuat sistem sendiri yang menentukan bagaimana objek diciptakan, disusun dan digambarkan. Kelas *creational*

*pattern* menggunakan *inheritance* untuk mengubah kelas yang diinstansiasi, sedangkan objek *creational pattern* akan memberikan intansiasi untuk objek yang lain. *Builder* dapat menggunakan salah satu dari *pattern* untuk mengimplemen komponen yang akan dibangun. *Prototype* dapat menggunakan *singleton* dalam implementasinya [ERR95].

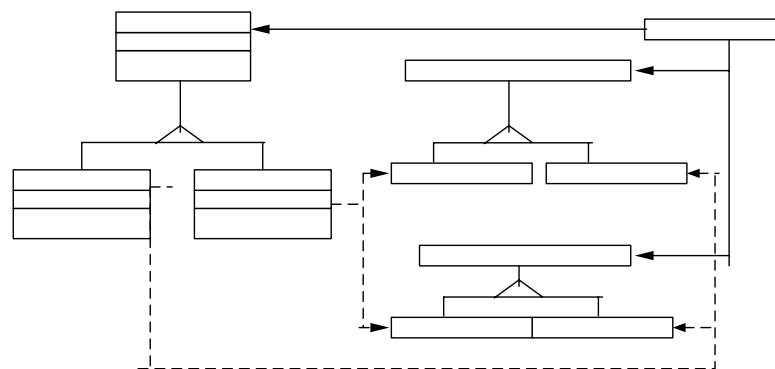
### 2.2.1.1 Abstract Factory Pattern

*Abstract factory pattern* menyediakan sebuah *interface* untuk menciptakan kumpulan objek yang berelasi dan bergantung tanpa menentukan kelas konkret [ERR95].

*Abstract Factory pattern* digunakan ketika [ERR95] :

1. Sebuah sistem seharusnya berdiri sendiri dan dapat dipandang dari hasil yang diciptakan, diubah dan digambarkannya.
2. Sistem seharusnya dikonfigurasi dengan salah satu kumpulan produknya.
3. Ingin menyediakan *library* kelas dari *product*.

Struktur *abstract factory pattern* terdapat pada Gambar 6.



Gambar 6 Struktur *abstarcet factory pattern*

Keterangan struktur *abstract factory pattern*

- a. *AbstractFactory* : mendeklarasikan *interface* untuk operasi yang menciptakan objek terpisah.
- b. *ConcreteFactory* : mengimplemen operasi untuk menciptakan objek konkret.
- c. *AbstractProduct* : mendefinisikan objek yang akan diciptakan dengan menyesuaikan *factory* konkret.
- d. *Client* : hanya menggunakan *interface* yang dideklarasikan oleh kelas *AbstractFactory* dan *AbstractProduct*.

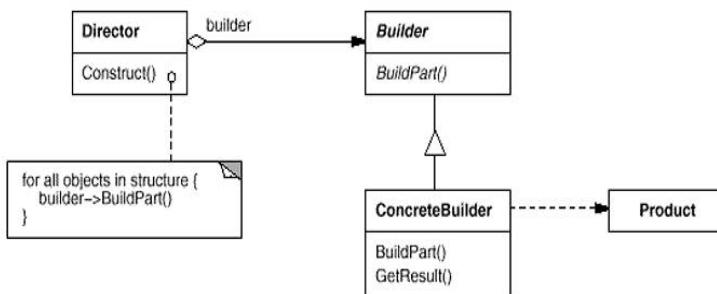
### 2.2.1.2 Builder Pattern

*Builder pattern* merupakan *pattern* yang bertujuan untuk menyusun objek yang kompleks karena gambarannya sehingga proses susunan yang sama dapat diciptakan pada gambaran yang berbeda [ERR95].

*Builder pattern* digunakan ketika [ERR95] :

- a. Algoritma untuk menciptakan sebuah objek yang kompleks harus bebas dari bagian yang membuat objek tersebut dan bebas dari bagaimana bagian-bagian tersebut disusun.
- b. Proses pembagunan harus mengizinkan gambaran yang berbeda untuk objek yang dibangun.

Struktur *builder pattern* terdapat pada Gambar 7.



Gambar 7 Struktur *builder pattern*

Keterangan struktur *builder pattern* :

- a. *Builder* : menentukan *interface abstract* untuk menciptakan objek *Product*.
- b. *ConcreteBuilder* : membuat dan menggunakan bagian dari implementasi *product interface* *Builder*, mendefinisikan dan membuat *track* untuk menggambarkannya, dan menyediakan *interface* untuk memperoleh *product*.
- c. *Director* : menyusun objek yang menggunakan *interface Builder*.
- d. *Product* : menggambarkan objek kompleks pada struktur. *ConcreteBuilder* membangun gambaran *product* internal, mendefinisikan proses melalui kumpulannya dan memasukkan kelas-kelas yang mendefinisikan unsur bagian utama, termasud *interface* untuk mengumpulkan *product* ke hasil akhir.

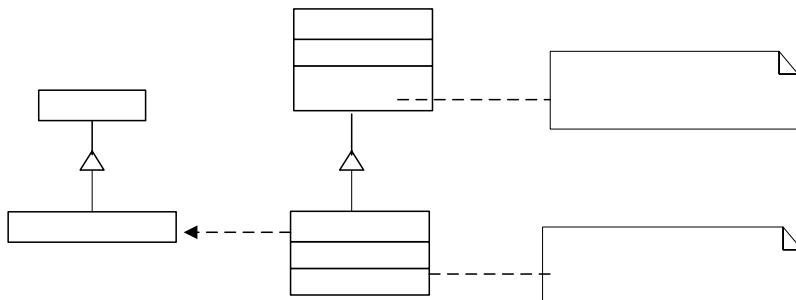
### 2.2.1.3 Factory Method Pattern

*Factory method pattern* menetapkan sebuah *interface* untuk menciptakan sebuah objek, tetapi mengizinkan sub kelas menentukan kelas mana yang diinstansiasi. *Factory method pattern* mengizinkan kelas membatalkan instansiasi ke sub kelas [ERR95].

*Factory method pattern* digunakan ketika [ERR95]:

- a. Kelas tidak dapat mendahului objek kelas yang harus diciptakan.
- b. Sebuah kelas menginginkan sub kelasnya untuk menentukan objek yang diciptakan.
- c. Kelas-kelas utusan bertanggung jawab terhadap salah satu dari beberapa *helper* sub kelas, dan untuk membatasi pengetahuan *helper* sub kelas mana sebagai delegasi.

Struktur *factory method pattern* dapat dilihat pada Gambar 8.



Gambar 8 Struktur *Factory Method Pattern*

Keterangan struktur *factory method pattern* :

- a. *Product* : mendefinisikan *interface* yang berasal dari penciptaan objek *factory method*.
- b. *ConcreteProduct* : implementasi dari *interface Product*.
- c. *Creator* : mendeklarasikan *factory method*, yang mengembalikan sebuah objek dari tipe *Product*. *Creator* juga mendefinisikan implementasi *default* dari *factory method* yang mengembalikan objek *default ConcreteProduct* dan akan memanggil *factory method* untuk menciptakan objek *Product*.
- d. Kelas *Concrete* : meng-*override* *factory method* untuk mengembalikan instansiasi dari *ConcreteProduct*.

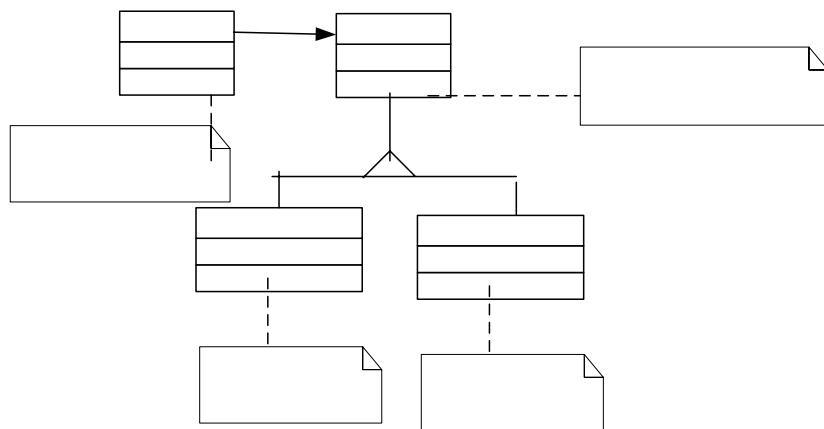
#### 2.2.1.4 Prototype Pattern

*Prototype pattern* bertujuan untuk menentukan jenis-jenis objek untuk menciptakan bentuk dasar dari instansiasi, dan menciptakan objek baru dengan menyalin *prototype* tersebut [ERR95].

*Prototype pattern* digunakan ketika [ERR95] :

- a. Sebuah sistem yang seharusnya bebas dari bagaimana produknya diciptakan, sehingga mengetahui bagaimana *productnya* diciptakan, disusun dan digambarkan.
- b. Kelas-kelas untuk instansiasi ditentukan pada saat *run-time*.
- c. Untuk menghindari pembuatan kelas hierarki dari *factory* yang paralel dari hierarki kelas *product*.
- d. Instansiasi kelas dapat memiliki salah satu dari kombinasi *state* yang berbeda.

Struktur *prototype pattern* dapat dilihat pada Gambar 9.



Gambar 9 struktur *prototype pattern*

Keterangan struktur *prototype pattern* :

- a. *Prototype* : mendeklarasikan *interface* untuk *cloning*.
- b. *ConcretePrototype* : mengimplementasi sebuah *Operation* untuk meng-*cloning* dirinya sendiri.
- c. *Client* : menciptakan sebuah objek baru dengan meminta *prototype* untuk meng-*cloning* dirinya sendiri.

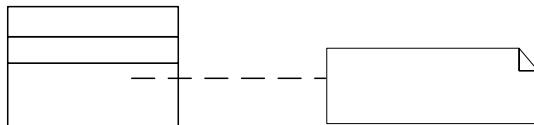
### 2.2.1.5 Singleton Pattern

*Singleton pattern* merupakan *pattern* yang menjamin kelas hanya memiliki satu instans, dan menyediakan sifat yang umum untuk mengaksesnya [ERR95].

*Singleton pattern* digunakan ketika [ERR95] :

- a. Harus ada satu instans dari sebuah kelas, dan mampu mengakses klien.
- b. Satu-satunya instans harus dapat diperluas melalui sub kelas, dan klien harus mampu menggunakan instans secara luas tanpa mengubah kode.

Struktur *singleton pattern* dapat dilihat pada Gambar 10.



Gambar 10 struktur *singleton pattern*

Keterangan struktur *singleton pattern* :

Berikut adalah dua tugas yang dimiliki kelas *singleton*.

- a. Mendefinisikan sebuah operasi *Instance* yang mengizinkan klien untuk mengakses instans yang unik.
- b. Harus bertanggung jawab untuk menciptakan *instance unique* sendiri.

### 2.2.2 Structural Pattern

*Structural pattern* diketahui melalui bagaimana kelas dan objek disusun untuk membentuk struktur yang lebih besar. Kelas *structural pattern* menggunakan *inheritance* untuk menyusun *interface* atau implementasi. Sebagai contoh sederhana, *structural pattern* mempertimbangkan bagaimana menggabungkan dua atau lebih kelas *inheritance* menjadi satu. Hasilnya adalah sebuah kelas yang menggabungkan properti dari kelas induknya. *Pattern* ini membuat kelas *library* secara *independent* bekerjasama [ERR95].

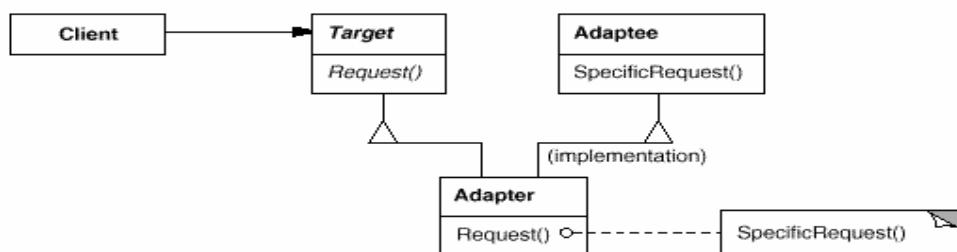
#### 2.2.2.1 Adapter Pattern

*Adapter pattern* merupakan *pattern* yang mengubah *interface* sebuah kelas menjadi *interface* *client* lain. *Adapter* mengizinkan kelas-kelas bekerjasama walaupun memiliki *interface* yang tidak sesuai [ERR95].

*Adapter pattern* digunakan ketika [ERR95] :

- a. Menggunakan kelas yang tersedia, dan *interface* kelas tersebut tidak sesuai dengan salah satu yang dibutuhkan.
- b. Menciptakan sebuah *reusable class* yang bekerjasama dengan kelas-kelas yang tidak berelasi.
- c. Membutuhkan penggunaan beberapa sub kelas yang ada. Sebuah objek *adapter* dapat menyesuaikan *interface* dengan kelas induknya.

Struktur *adapter pattern* dapat dilihat pada Gambar 11.



Gambar 11 Struktur *Adapter pattern*

Keterangan struktur *adapter pattern*:

- a. *Target* : mendefinisikan *domain interface* tertentu yang digunakan *Client*.
- b. *Client* : bekerjasama dengan objek menyesuaikan diri menjadi *interface Target*.
- c. *Adaptee* : mendefinisikan sebuah *interface* yang ada untuk mengadaptasikan.
- d. *Adapter* : menyesuaikan *interface* dari *Adaptee* ke *interface Target*.

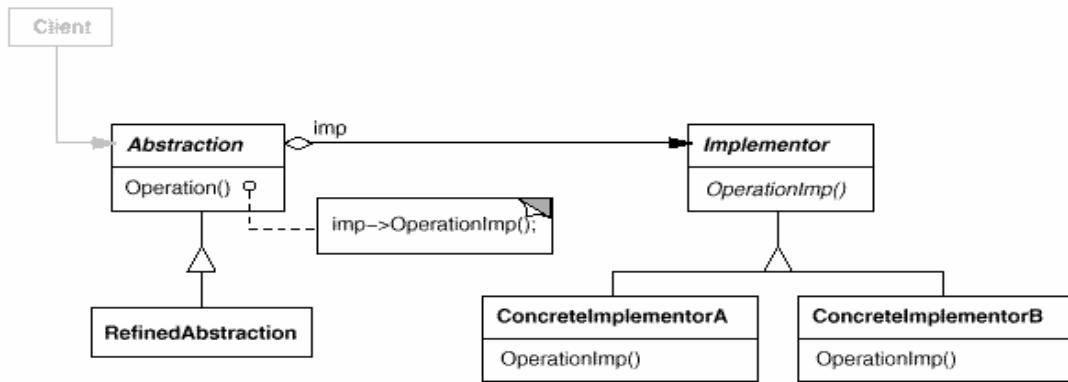
### 2.2.2.2 Bridge Pattern

*Bridge pattern* merupakan *pattern* yang memisahkan abstraksi objek dari implementasinya sehingga dapat mengubah objek dengan bebas [ERR95].

*Bridge pattern* digunakan ketika [ERR95] :

- a. Menghindari pengikatan permanen di antara abstraksi dan implementasi.
- b. Abstraksi dan implementasi harus dapat dikembangkan melalui sub kelas. Dalam hal ini, *bridge pattern* mengizinkan penggabungan abstraksi dan implementasi yang berbeda dan memperbanyak secara bebas.
- c. Mengganti implementasi dari abstraksi yang seharusnya tidak berpengaruh pada klien sehingga kodennya tidak harus di kompile ulang.
- d. Menyembunyikan implementasi dari abstraksi secara lengkap dari klien.
- e. Memperbanyak kelas.
- f. Membagi implementasi diantara banyak objek.

Struktur *bridge pattern* dapat dilihat pada Gambar 12.



Gambar 12 Struktur *Bridge Pattern*

Keterangan struktur *bridge pattern*:

- Abstraction* : mendefinisikan *interface* abstrak dan mempertahankan referensi untuk objek dengan tipe *Implementor*.
- RefinedAbstraction* : memberikan *interface* yang telah didefinisikan oleh *Abstraction*
- Kelas *Implementor* : mendefinisikan *interface* untuk kelas implementasi.
- ConcreteImplementor* : mengimplementasi *interface* *Implementor* dan mendefinisikan implementasi konkretnya.

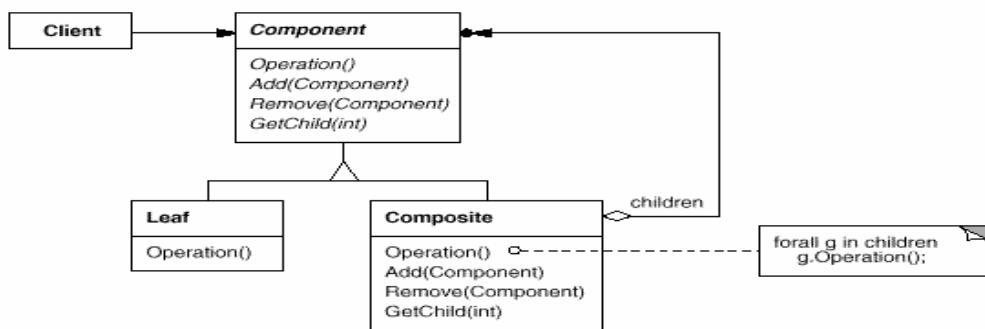
### 2.2.2.3 Composite Pattern

*Composite pattern* merupakan *pattern* yang menyusun objek menjadi tiga struktur untuk menggambarkan hierarki seluruh bagian. *Composite* mengizinkan klien menggunakan objek individu dan susunan dari objek yang seragam [ERR95].

*Composite pattern* digunakan ketika [ERR95] :

- Menggambarkan hierarki dari seluruh bagian objek.
- Client* mampu mengabaikan perbedaan antara objek komposisi dan individu.

Struktur *composite pattern* dapat dilihat pada Gambar 13.



Gambar 13 Struktur *Composite Pattern*

Keterangan struktur *composite pattern*:

- a. *Composite* : mendeklarasikan *interface* untuk objek, mengimplementasi tingkah laku dasar *interface* utama untuk seluruh kelas, mendeklarasikan *interface* untuk pengaksesan dan pengaturan sub komponen dan mendefinisikan sebuah *interface* untuk pengaksesan komponen induk dalam struktur yang berulang.
- b. *Leaf* : *leaf* tidak memiliki turunan dan mendefinisikan tingkah laku untuk objek primitif pada *composition*.
- c. *Composite* : mendefinisikan tingkah laku untuk komponen yang memiliki turunan, menyimpan komponen turunan, dan mengimplementasi *operation* turunan yang berelasi pada *interface Component*.
- d. *Client* : memanipulasi objek melalui *interface Component*.

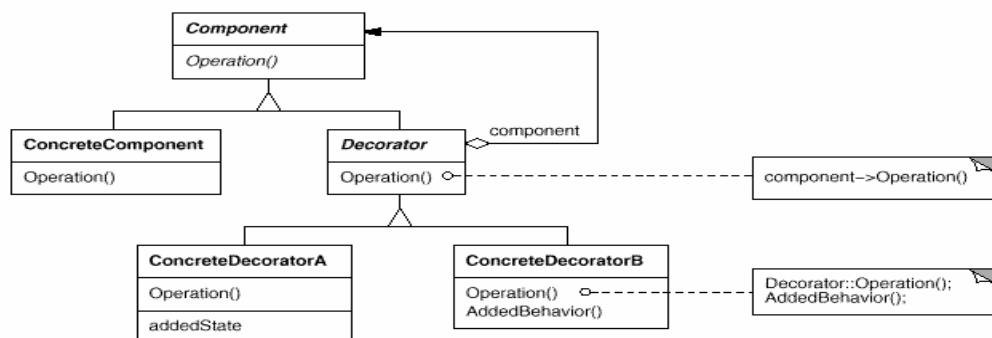
#### 2.2.2.4 Decorator Pattern

*Decorator pattern* merupakan *pattern* yang memberikan penambahan tanggung jawab pada sebuah objek secara dinamis. *Decorator* menyediakan alternatif yang sesuai untuk sub kelas selama peng-*extend*-an fungsi [ERR95].

Penggunaan *decorator pattern* adalah pada kondisi sebagai berikut [ERR95] :

- a. Menambah tanggung jawab pada objek individu secara dinamis dan jelas tanpa mempengaruhi objek yang lain.
- b. Tanggung jawab dapat diambil kembali.
- c. Ketika ekstensi melalui sub kelas tidak bermanfaat.

Struktur *decorator pattern* dapat dilihat pada Gambar 14.



Gambar 14 Struktur *Decorator Pattern*

Keterangan struktur *decorator pattern*:

- a. *Component* : mendefinisikan *interface* untuk objek yang dapat memiliki tanggung jawab yang ditambahkan secara dinamis.
- b. *ConcreteComponent* : mendefinisikan objek untuk menambahkan tanggung jawab yang dapat di berikan.
- c. *Decorator* : mempertahankan referensi sebuah objek *Component* dan mendefinisikan *interface* yang sesuai dengan *interface Component*.
- d. *ConcreteDecorator* : menambahkan tanggung jawab pada *component*.

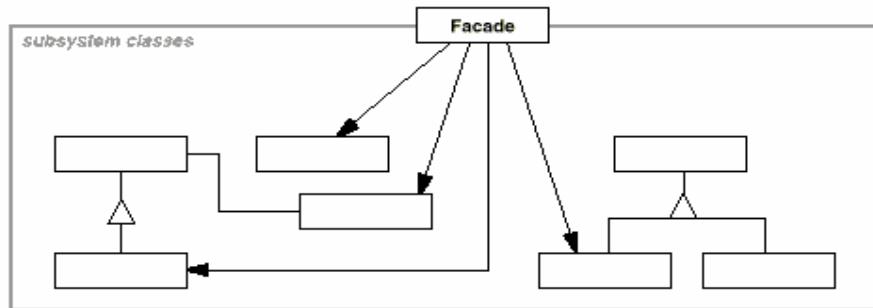
#### 2.2.2.5 Facade Pattern

*Facade pattern* menyediakan kumpulan *interface* dalam sebuah sub sistem. *Facade* mendefinisikan *interface* tingkat tinggi sehingga membuat sub sistem lebih mudah untuk digunakan [ERR95].

*Facade pattern* digunakan ketika [ERR95] :

- a. Menyediakan *interface* sederhana untuk sub sistem yang kompleks.
- b. Terdapat banyak ketergantungan di antara klien dan kelas implementasi dari sebuah abstraksi.
- c. Menggunakan *facade* untuk mendefinisikan masukan *point* pada setiap level sub sistem.

Struktur *facade pattern* dapat dilihat pada Gmbar 15



Gambar 15 Struktur Facade Pattern

Keterangan struktur *facade pattern*:

- a. *Facade* : mengetahui kelas sub sistem mana yang bertanggung jawab untuk sebuah permintaan dan mendelegasikan permintaan klien untuk disesuaikan dengan objek sub sistem.
- b. *Subsystem* : mengimplementasi fungsi sub sistem, menangani tugas yang diberikan oleh objek *Facade*, dan tidak memiliki pengetahuan *Facade* sehingga tetap menyimpan yang bukan referensi untuk *facade*.

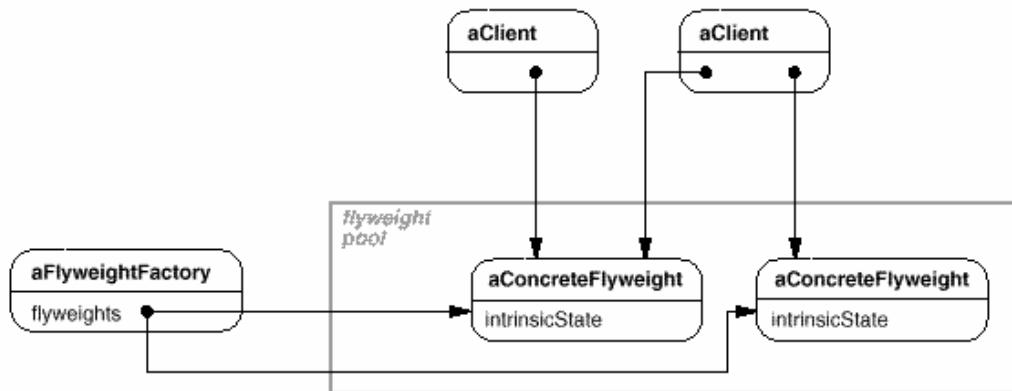
#### 2.2.2.6 Flyweight Pattern

*Flyweight pattern* menggunakan *sharing* untuk beberapa objek yang disatukan dengan baik secara efisien [ERR95].

*Flyweight pattern* akan efektif jika digunakan pada kondisi sebagai berikut [ERR95] :

- Sebuah aplikasi menggunakan objek yang banyak.
- Menghabiskan biaya yang tinggi karena jumlah objek yang banyak.
- Sebagian besar *state* dapat dibuat akibat dari *state* lain.
- Banyak pengelompokan objek yang seharusnya dipindahkan setiap kali *state* dipindahkan.
- Aplikasi tidak bergantung pada identitas objek.

Struktur *flyweight pattern* dapat dilihat pada Gambar 16.



Gambar 16 Struktur *Flyweight Pattern*

Keterangan struktur *flyweight pattern*:

- Flyweight* : mendeklarasikan sebuah *interface* melalui *flyweight* yang dapat menerima dan berperan pada *state* yang di sebabkan oleh keadaan lain.
- ConcreteFlyweight* : mengimplementasi *interface* sub kelas *Flyweight* untuk dibagi.
- UnsharedConcreteFlyweight* : tidak seluruh sub kelas *Flyweight* harus dibagi.
- FlyweightFactory* : menciptakan dan mengatur objek *flyweight* dan menjamin bahwa *flyweight* dibagi dengan baik. Ketika klien meminta *flyweight*, maka objek *FlyweightFactory* menyediakan instansiasi atau menciptakan satu instansiasi jika tidak ada.
- Client* : mempertahankan sebuah referensi untuk *flyweight* dan memperhitungkan atau menyimpan *state* dari *flyweight*.

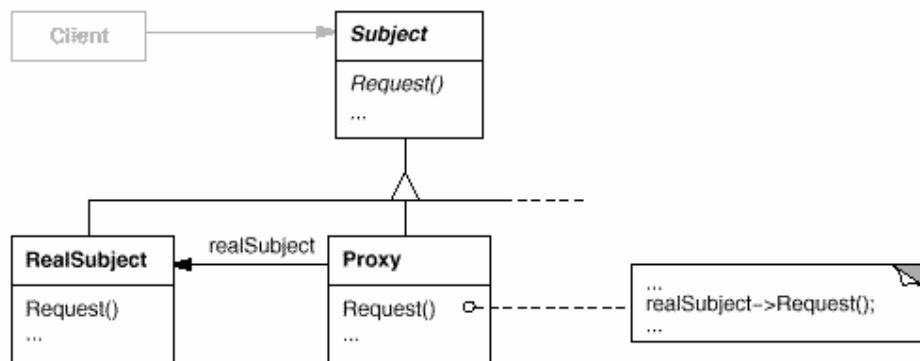
#### 2.2.2.7 Proxy Pattern

*Proxy pattern* menyediakan pengganti objek lain untuk mengontrol pengaksesan. *Proxy* dapat digunakan kapan pun ketika membutuhkan kecanggihan referensi ke sebuah objek daripada *pointer* sederhana [ERR95].

Berikut diuraikan beberapa kondisi utama dalam menggunakan *proxy pattern* [ERR95]

- :
  - a. Sebuah *proxy remote* menyediakan perwakilan lokal untuk objek pada bagian alamat yang berbeda.
  - b. *Proxy virtual* menciptakan objek pada permintaan.
  - c. *Proxy protection* mengontrol pengaksesan pada objek.
  - d. *Smart reference* adalah pergantian untuk *pointer* yang menunjukkan penambahan peran ketika objek diakses.

Struktur *proxy pattern* dapat dilihat pada Gambar 17.



Gambar 17 Struktur *Proxy Pattern*

Keterangan struktur *proxy pattern*:

- a. *Proxy* : memelihara referensi yang mengizinkan pengaksesan *proxy* pada *RealSubject*, menyediakan *interface* untuk *Subject* sehingga *proxy* dapat disubstitusi, mengontrol pengaksesan dan bertanggung jawab untuk pembuatan dan penghapusan *interface*.
- b. *Subject* : mendefinisikan *interface* utama untuk *RealSubject* dan *Proxy* sehingga *Proxy* dapat digunakan dimana *RealSubject* diharapkan.

### 2.2.3 Behavioral Pattern

*Behavioral pattern* dihubungkan dengan algoritma dan penetapan tanggung jawab diantara objek. *Behavioral pattern* tidak hanya menentukan *pattern* untuk objek atau kelas tetapi *pattern* untuk komunikasi diantara objek dan kelas [ERR95].

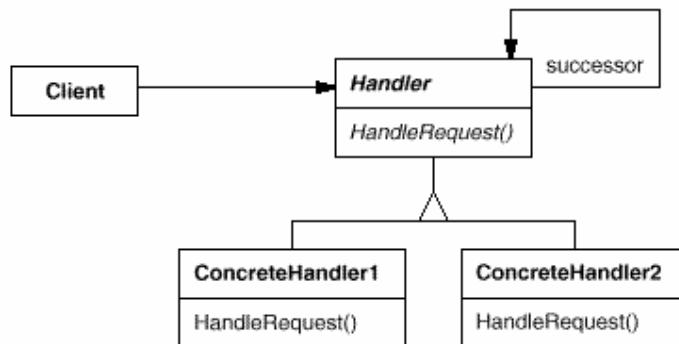
#### 2.2.3.1 Chain of Responsibility Pattern

*Chain of Responsibility Pattern* merupakan *pattern* yang menghindari penggabungan pengirim permintaan ke penerimanya dengan memberikan lebih dari satu objek yang mencoba untuk menangani permintaan [ERR95].

*Chain of responsibility* digunakan ketika [ERR95] :

- Lebih dari satu objek harus menangani permintaan.
- Menginginkan hasil permintaan untuk beberapa objek tanpa menentukan penerima secara eksplisit.
- Terdapat sekumpulan objek sehingga dapat menangani permintaan yang ditentukan secara dinamis.

Struktur *Chain of responsibility* dapat dilihat pada Gambar 18.



Gambar 18 Struktur *Chain of Responsibility Pattern*

Keterangan struktur chain of responsibility pattern:

- Handler* : mendefinisikan *interface* untuk menangani permintaan dan implementasi dari hubungan *successor*.
- ConcreteHandler* : dapat mengakses *successor* dan jika *ConcreteHandler* dapat menangani permintaan, maka sebaliknya *ConcreteHandler* meneruskan permintaan ke *successor*.
- Client* : menginisiasi permintaan ke objek *ConcreteHandler* pada *chain*.

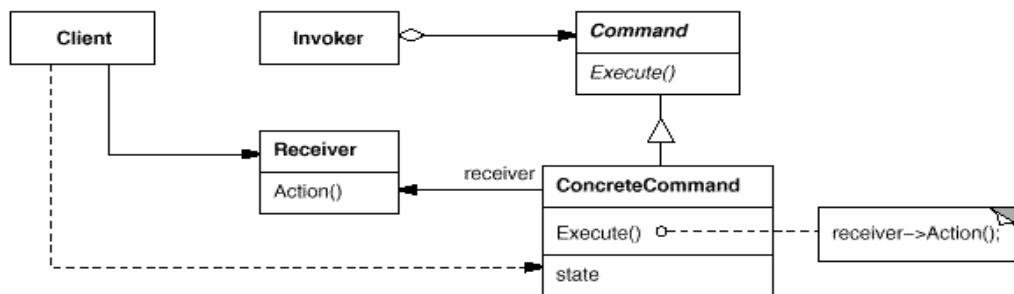
### 2.2.3.2 Command Pattern

*Command pattern* merupakan *pattern* yang membungkus permintaan sebagai objek, dengan demikian mengizinkan parameter klien dengan permintaan yang berbeda, mengantri atau mencatat permintaan, dan mendukung kemampuan membuka operasi [ERR95].

*Command pattern* digunakan ketika [ERR95] :

- a. Objek parameter melalui peran yang ditampilkan sebagai objek *MenuItem*.
- b. Menentukan, antrian dan pengeksekusian permintaan pada waktu yang berbeda.
- c. Mendukung *undo*.
- d. Mendukung pencatatan perubahan sehingga dapat digunakan dalam kasus sistem yang rusak
- e. Struktur sistem disekitar operasi level tinggi dibangun pada operasi primitif.

Struktur *command pattern* dapat dilihat pada Gambar 19.



Gambar 19 Struktur *Command Pattern*

Keterangan struktur *command pattern*:

- a. *Command* : mendeklarasikan *interface* untuk menjalankan operasi.
- a. *ConcreteCommand* : mendefinisikan keterkaitan diantara objek *Receiver* dan *action*, dan mengimplemen *Execute* melalui pemanggilan operasi pada *Receiver*.
- b. *Client* : menciptakan objek *ConcreteCommand* dan mengatur *receiver*.
- c. *Invoker* : menanyakan *command* untuk membawa *request*.

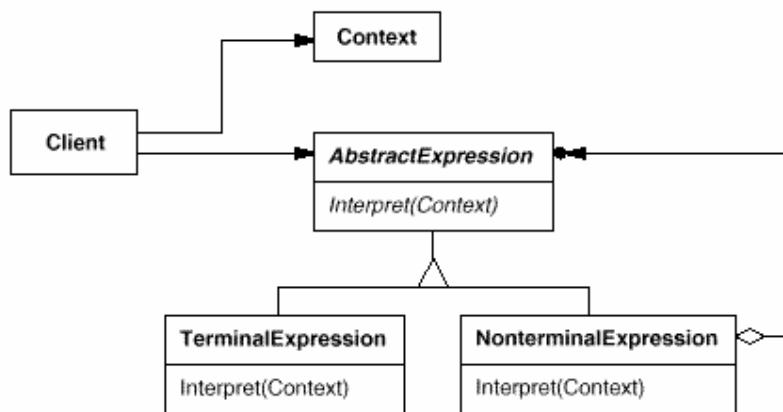
### 2.2.3.3 Interpreter Pattern

Diberikan sebuah bahasa, mendefinisikan representasi untuk *grammar* (tata bahasa) dan *interpreter* yang menggunakan repsentasi tata bahasa tersebut untuk menginterpretasi kalimat dalam bahasa tersebut [ERR95].

*Interpreter pattern* digunakan ketika [ERR95] :

- Tata bahasa sederhana.
- Efisiensi tidak kritis.

Struktur *Interpreter pattern* dapat dilihat pada Gambar 20.



Gambar 20 Struktur *Interpreter Pattern*

Keterangan struktur *interpreter pattern*:

- AbstractExpression* : mendeklarasikan sebuah operasi *Interpreter* abstrak sehingga menjadi seluruh *node* dalam sintaks pohon abstrak.
- TerminalExpression* : mengimplementasi sebuah operasi *Interpreter* yang berhubungan dengan simbol terminal dalam tata bahasa dan instansiasi dibutuhkan untuk setiap simbol terminal dalam kalimat.
- NonterminalExpression* : salah satu contoh kelas yang dibutuhkan untuk setiap aturan  $R ::= R_1 \ R_2 \dots R_n$  pada tata bahasa dan mempertahankan instansiasi variabel dari tipe *AbstractExpression* untuk setiap simbol  $R$  sampai  $R_n$ .
- Context* : terdiri dari informasi umum *interpreter*.
- Client* : membuat sebuah sintaks pohon abstrak yang menggabarkan kalimat khusus dalam bahasa yang mendefinisikan tata bahasa dan memanggil operasi *Interpret*.

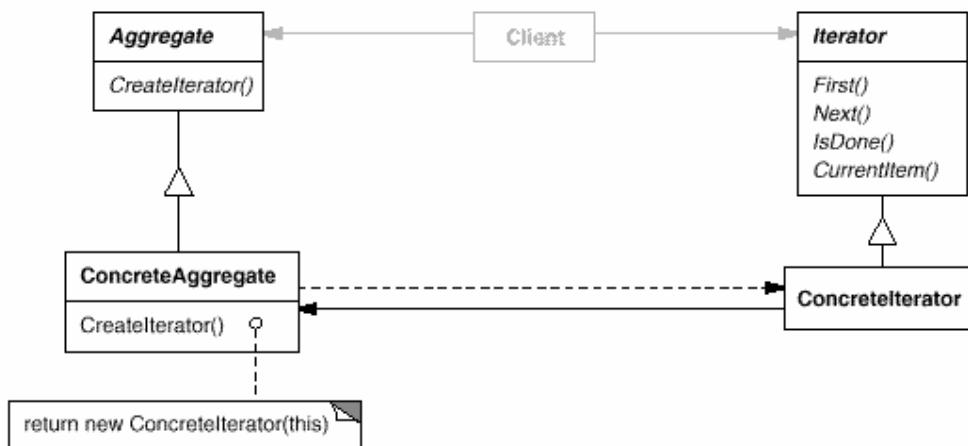
#### 2.2.3.4 Iterator Pattern

*Iterator pattern* menyediakan cara untuk mengakses elemen dari kumpulan objek secara *sequence* [ERR95].

*Iterator pattern* digunakan untuk [ERR95] :

- Mengakses sekumpulan isi objek tanpa menampilkan gambaran internalnya.
- Mendukung banyak lintasan dari sekumpulan objek.
- Menyediakan *interface* yang sama untuk melintasi sekumpulan struktur yang berbeda.

Struktur *Iterator pattern* dapat dilihat pada Gambar 21.



Gambar 21 Struktur *Iterator Pattern*

Keterangan struktur *iterator pattern*:

- Iterator* : mendefinisikan sebuah *interface* untuk pengaksesan dan pelintasan elemen.
- ConcreteIterator* : mengimplementasi *interface* *Iterator* dan mempertahankan posisi dalam *aggregate traversal* .
- Aggregate* : mendefinisikan sebuah *interface* untuk menciptakan objek *Iterator*.
- ConcreteAggregate* : mengimplementasi *Iterator* dalam menciptakan *interface* untuk mengembalikan instansiasi dari *ConcreteIterator*.

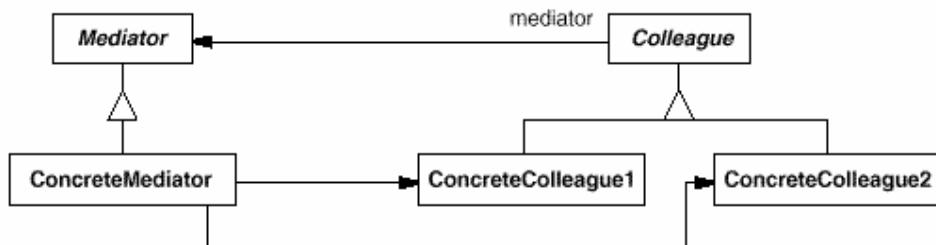
### 2.2.3.5 Mediator Pattern

*Mediator pattern* mendefinisikan sebuah objek yang membungkus sekumpulan interaksi objek. *Mediator* mempertimbangkan hilangnya penggabungan melalui penyimpanan objek dan mengizinkan mengubah interaksi secara *independent* [ERR95].

*Mediator pattern* digunakan ketika [ERR95] :

- Sekumpulan objek berkomunikasi tetapi dengan cara yang rumit.
- Menggunakan objek berulang sulit dilakukan karena mengarah kepada komunikasi dengan banyak objek.
- Tingkah laku yang didistribusikan diantara beberapa kelas seharusnya mampu di *customize* tanpa banyak sub kelas.

Struktur *Mediator pattern* dapat dilihat pada Gambar 22.



Gambar 22 Struktur *Mediator Pattern*

Keterangan struktur *mediator pattern*:

- Mediator* : mendefinisikan sebuah *interface* untuk berkomunikasi dengan objek *Colleague*.
- ConcreteMediator*: mengimplementasi tingkah laku melalui koordinasi objek *Colleague*.
- Kelas *Colleague* : setiap kelas *Colleague* mengetahui objek *Mediator*nya.

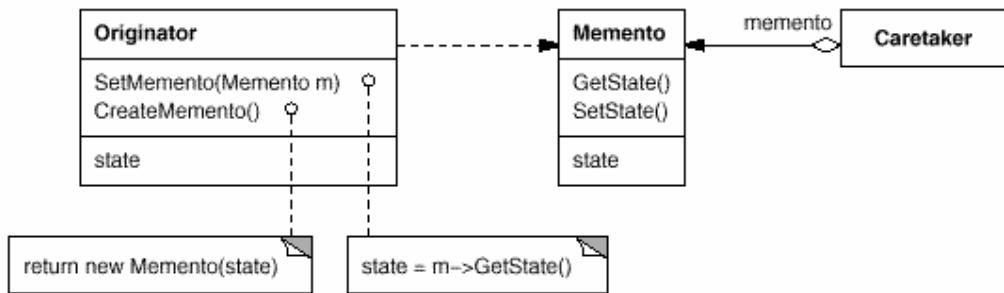
### 2.2.3.6 Memento Pattern

*Memento pattern* merupakan *pattern* tanpa mengganggu pembungkusan, mengambil dan mengadakan *state* objek *internal* sehingga objek dapat disimpan ke *state* selanjutnya [ERR95].

*Memento pattern* digunakan ketika [ERR95] :

- Gambaran sebuah *state* objek harus disimpan sehingga dapat dihapus pada *state* berikutnya.
- Menunjukkan *interface* untuk menghasilkan *state* yang akan menunjukkan *detail* implementasi dan menghentikan pembungkusan objek.

Struktur *Memento pattern* dapat dilihat pada Gambar 23.



Gambar 23 Struktur Memento Pattern

Keterangan struktur *memento pattern*:

- Memento* : menyimpan *state internal* dari objek *Originator*.
- Originator* : menciptakan sebuah memento yang terdiri dari gambaran *state internal* dan menggunakan *memento* untuk menghapus *state internal*.
- Caretaker* : bertanggung jawab untuk penyimpanan *memento* dan tidak pernah mengoperasikan atau menguji isi dari *memento*.

#### 2.2.3.7 Observer Pattern

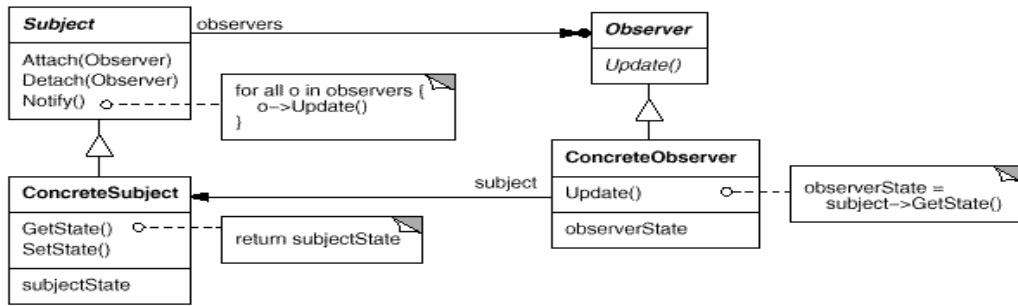
*Observer pattern* mengasumsikan bahwa objek yang berisi data, terpisah dari objek yang menampilkan data [ERR95].

*Observer* menetapkan satu ke banyak kebergantungan antar objek, sehingga ketika satu objek melakukan perubahan status, semua yang bergantung kepada objek akan diberitahukan dan dibaharu secara otomatis [ERR95].

*Observer pattern* digunakan ketika [ERR95] :

1. *Abstraction* mempunyai dua aspek, aspek yang satu bergantung pada yang lain. *Encapsulation* aspek pada pemisahan objek mengizinkan terjadinya pertukaran dan penggunaan kembali secara bebas.
2. Perubahan terhadap suatu objek membutuhkan perubahan objek yang lain, dan tidak mengetahui berapa banyak objek yang dibutuhkan untuk diubah.
3. Suatu objek seharusnya mampu untuk memberitahukan objek lain tanpa membuat asumsi mengenai objek tersebut dan tidak saling terikat.

Struktur *observer pattern* dapat dilihat pada Gambar 24.



Gambar 24 Struktur *Observer Pattern*

Keterangan struktur *observer pattern*:

- Subject* : mengetahui *Observer*nya. Beberapa objek *observer* memungkinkan menjalankan sebuah subyek dan menyediakan *interface* untuk memasang dan melepaskan objek *Observer*.
- Observer* : mendefinisikan perubahan *interface* untuk objek sehingga dapat diketahui perubahan pada subyek.
- ConcreteSubject* : menyimpan *state* yang penting untuk objek *ConcreteObserver* dan mengirimkan pemberitahuan ke *observer* ketika *state*-nya berubah.
- ConcreteObserver* : memelihara referensi ke objek *ConcreteSubject*, menyimpan *state* sehingga tetap konsisten dengan subyek dan mengimplementasi perubahan *interface* *Observer* untuk menunjukkan *state* yang konsisten.

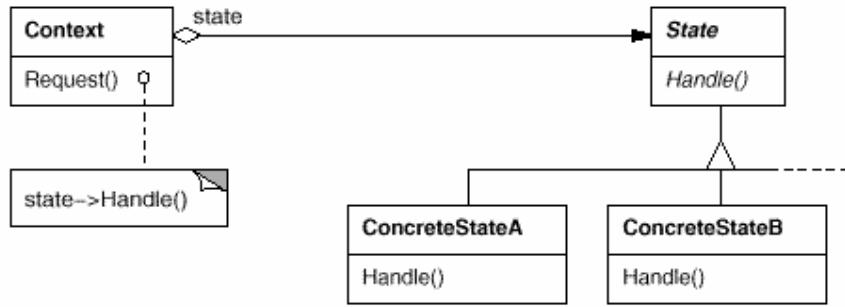
#### 2.2.3.8 State Pattern

*State pattern* mengizinkan sebuah objek untuk mengubah tingkah lakuanya ketika *state* internalnya berubah. Objek akan menampilkan perubahan kelasnya[ERR95].

*State pattern* digunakan pada saat salah satu kasus berikut ini[ERR95]:

- Sebuah tingkah laku objek bergantung pada *state*, dan harus mengubah tingkah lakuanya pada saat *run-time* berdasarkan pada *state*.
- Memiliki operasi yang banyak, menyatakan banyak kondisi yang bergantung pada *state* objek.

Struktur *state pattern* dapat dilihat pada Gambar 25.



Gambar 25 Struktur *State Pattern*

Keterangan struktur *state pattern* :

- Context* : mendefinisikan *interface* ke *client*, memelihara sebuah instansiasi dari sub kelas *ConcreteState* sehingga mendefinisikan *state* saat ini.
- State* : mendefinisikan sebuah *interface* untuk pembungkusan tingkah laku yang berhubungan dengan *state* khusus dari *Context*.
- Sub kelas *ConcreteState* : setiap sub kelas mengimplementasi sebuah tingkah laku yang berhubungan dengan *state* pada *Context*.

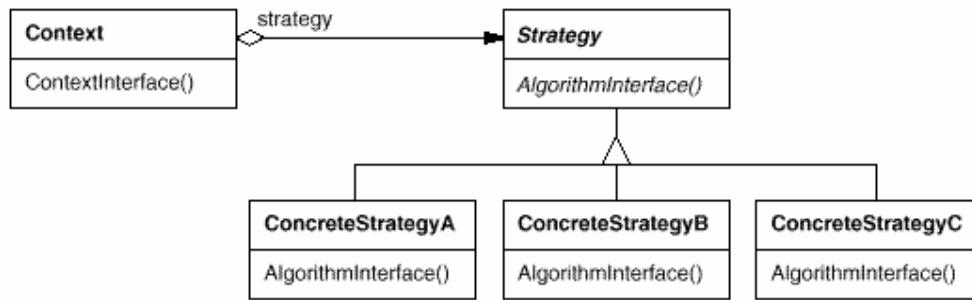
#### 2.2.3.9 Strategy Pattern

*Strategy pattern* mendefinisikan kumpulan algoritma, *encapsulate* dan memampukan algoritma dan *encapsulate* saling bertukar. *Strategy* mengizinkan algoritma berubah secara bebas dari *client* yang menggunakannya [ERR95].

*Strategy* digunakan ketika [ERR95] :

- Banyak relasi kelas berbeda hanya pada tingkah lakunya. *Strategy* menyediakan cara untuk mengkonfigurasi kelas dengan salah satu atau banyak tingkah laku.
- Membutuhkan perbedaan bermacam-macam algoritma. Sebagai contoh, mendefinisikan algoritma yang menggambarkan perbedaan ruang atau waktu.
- Sebuah algoritma menggunakan data sehingga *client* tidak perlu mengetahuinya.
- Sebuah kelas mendefinisikan banyak tingkah laku, dan hal ini menampilkan banyak kondisi dalam operasinya.

Struktur *strategy pattern* dapat dilihat pada Gambar 26.



Gambar 26 Struktur *Strategy Pattern*

Keterangan struktur *startegy pattern* :

- a. *Strategy* : mendeklarasikan sebuah *interface* utama untuk mendukung seluruh algoritma.
- b. *ConcreteStrategy* : mengimplementasi algoritma menggunakan *interface* *Strategy*.
- c. *Context* : dikonfigurasi dengan objek *ConcreteStrategy*, mempertahankan referensi ke objek *Strategy* dan mendefinisikan sebuah *interface* yang mengizinkan *Strategy* mengakses datanya.

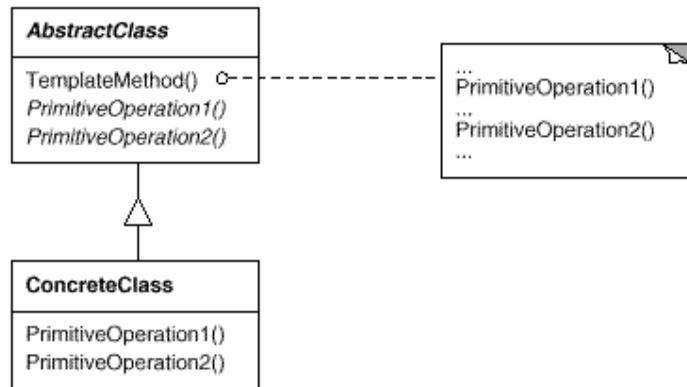
#### 2.2.3.10 Template Method Pattern

*Template method pattern* mendefinisikan kerangka dari algoritma pada operasi, menunda beberapa tahapan untuk sub kelas. *Template method* mengizinkan sub kelas mendefinisikan ulang beberapa tahapan pada sebuah algoritma tanpa mengubah struktur algoritma [ERR95].

*Template method pattern* akan digunakan ketika [ERR95] :

- a. Untuk mengimplementasi bagian invariant dari sebuah algoritma dan mewariskannya kepada sub kelas untuk mengimplementasi tingkah laku yang dapat berubah.
- b. Ketika tingkah laku umum diantara sub-sub kelas seharusnya difaktorkan dan dialokasikan pada kelas utama untuk menghindari duplikasi kode.
- c. Untuk mengontrol keberadaan sub-sub kelas.

Struktur *template method pattern* dapat dilihat pada Gambar 27.



Gambar 27 Struktur *Template Method Pattern*

Keterangan struktur template method pattern:

- AbstractClass* : mendefinisikan abstrak *PrimitiveOperation* sehingga sub kelas konkret mendefinisikan tahap implemen pada algoritma dan mengimplemen *template method* pada kumpulan algoritma.
- ConcreteClass* : mengimplemen *PrimitiveOperation* untuk membawa tahapan sub kelas tertentu pada algoritma.

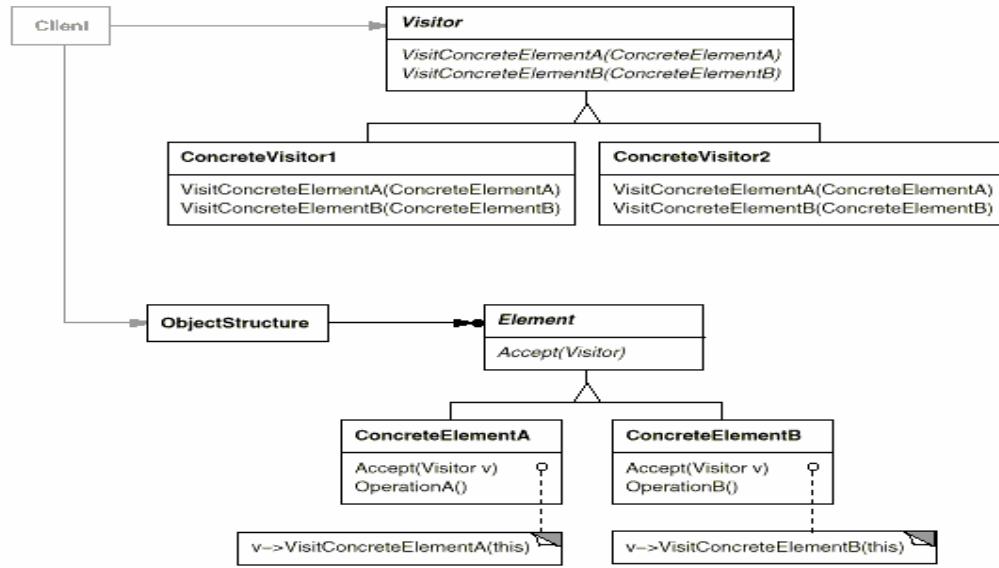
#### 2.2.3.11 Visitor Pattern

*Visitor pattern* menggambarkan sebuah operasi untuk menampilkan elemen struktur objek. *Visitor* mengizinkan mendefinisikan operasi baru tanpa mengubah kelas-kelas dari elemen yang dioperasikannya [ERR95].

*Visitor pattern* digunakan ketika [ERR95] :

- Struktur objek terdiri dari banyak kelas objek dengan *interface* yang berbeda, dan menunjukkan operasi pada objek yang bergantung pada kelas-kelas konkret.
- Banyak operasi yang berbeda dan tidak berelasi harus ditampilkan objek dalam struktur objek.
- Kelas-kelas mendefinisikan struktur objek yang tidak selalu berubah, tetapi mendefinisikan operasi yang baru diatas struktur.

Struktur *visitor pattern* dapat dilihat pada Gambar 28.



Gambar 28 Struktur *Visitor Pattern*

Keterangan visitor pattern:

- Visitor* : mendeklarasikan operasi *visitor* untuk setiap kelas *ConcreteElement* pada struktur objek.
- ConcreteVisitor* : mengimplementasi setiap operasi yang dideklarasikan oleh *Visitor*.
- Element* : mendefinisikan operasi *Accept* yang mengambil *visitor* sebagai argumen.
- ConcreteElement* : mengimplementasi operasi *Accept* yang mengambil *visitor* sebagai argumen.
- ObjectStructure* : menyediakan *interface* tingkat tinggi untuk mengizinkan *visitor* mengunjungi elemennya dan memungkinkan menjadi *composite*.

### 2.3 Studi Aplikasi PMB

Pada subbab ini dijelaskan studi yang dilakukan terhadap aplikasi SI PMB PI Del dan SI PMB PA2, yang menjadi penerapan *design pattern* pada SI PMB TA2008.

Aplikasi SI PMB PI Del dibuat tanpa menggunakan dokumentasi. Aplikasi tersebut telah digunakan dalam membantu proses penerimaan mahasiswa baru di PI Del. Kemudian aplikasi tersebut dikembangkan melalui penggerjaan proyek akhir 2 (PA2) dengan dokumen yang dibuat berdasarkan aplikasi SI PMB PA2. Aplikasi SI PMB PA2

tidak digunakan karena tidak selesai dikembangkan. Tetapi struktur tabel pada dokumen SI PMB PA2 digunakan sebagai inspirasi dalam menyusun tabel pada SI PMB TA2008.

### 2.3.1 Studi Aplikasi SI PMB PI Del

Studi aplikasi SI PMB PI Del dilakukan dengan mempelajari aplikasi melalui *user interface* dan menjalankan setiap fungsi yang disediakan pada aplikasi. Hal tersebut dilakukan karena SI PMB PI Del tidak memiliki *source code* dan dokumentasi yang dapat diacu. Hasil yang dicapai dalam pelaksanaan studi aplikasi SI PMB PI Del adalah daftar fungsi yang tersedia dan karakteristik *user* dan tanggung jawabnya. Berikut diuraikan daftar fungsi yang tersedia dan karakteristik user SI PMB PI Del.

**Tabel 2 Daftar spesifikasi fungsi SI PMB PI Del**

No.	Fungsi/Fitur Utama	Kemampuan Software
1	<i>Login</i>	Melakukan autentikasi <i>user</i> .
2	<i>Logout</i>	Memutuskan koneksi <i>user</i> ke database.
3	<i>ViewEvent</i>	Menampilkan <i>event</i> .
4	<i>ViewBerita</i>	Menampilkan berita.
5	<i>ViewFaq</i>	Menampilkan faq.
6	<i>ViewAgenda</i>	Menampilkan agenda.
7	<i>AddFaq</i>	Menambahkan faq.
8	<i>MakeRegistration</i>	Menerima data yang dientry oleh <i>user</i> .
9	<i>EditDataPeserta</i>	Menerima data yang telah diubah oleh <i>user</i> .
10	<i>ViewPesertaYangLulus</i>	Menampilkan data peserta yang lulus.
11	<i>ViewDataPendaftar</i>	Menampilkan detail data pendaftar.
12	<i>AddBerita</i>	Menerima data yang di entry jika isi berita ditambah.
13	<i>EditBerita</i>	Menerima data berita yang di edit.
14	<i>AddAgenda</i>	Menerima data agenda jika terdapat penambahan agenda.
15	<i>EditAgenda</i>	Menerima data agenda yang di edit.
16	<i>AddEvent</i>	Menerima data <i>event</i> jika terdapat penambahan <i>event</i> .
17	<i>EditEvent</i>	Menerima data <i>event</i> yang di edit.
18	<i>AddJadwalUjian</i>	Menerima data jadwal ujian jika terdapat penambahan jadwal ujian.
19	<i>EditJadwalUjian</i>	Menerima data jadwal ujian yang di jadwal ujian.
20	<i>AddLokasiUjian</i>	Menerima data lokasi ujian jika terdapat penambahan lokasi ujian.
21	<i>EditLokasiUjian</i>	Menerima data lokasi ujian yang di lokasi ujian.
22	<i>AnswerFaq</i>	Menerima data faq yang di entry.

No.	Fungsi/Fitur Utama	Kemampuan Software
23	EditFaq	Menerima data <i>entry</i> yang telah di <i>edit</i> .

**Tabel 3 Karakteristik pengguna SI PMB PI Del**

No.	Peran	Deskripsi Tugas
1	Administrator	Maintain aplikasi Menambah data (berita, event dan agenda) Update data Menjawab <i>FAQ</i>
2	Operator (staf administrasi)	Mencetak data pendaftar. Mengubah data pendaftar. Melihat berita, event, agenda, dan <i>FAQ</i> .
3	Visitor	Meng- <i>entry</i> data diri (melakukan pendaftaran) Melihat data pendaftar Melihat berita, event, agenda, dan <i>FAQ</i> . Menambah <i>FAQ</i> .

### 2.3.2 Studi Aplikasi SI PMB PA2

Studi aplikasi SI PMB PA2 dilakukan dengan melihat struktur tabel aplikasi (daftar tabel) sebagai inspirasi dalam penyusunan tabel dan kelas yang dibutuhkan dalam melakukan rancang ulang. Hasil yang dicapai dalam pelaksanaan studi aplikasi SI PMB PA2 adalah daftar tabel aplikasi SI PMB PA2. Berikut diuraikan daftar tabel aplikasi tersebut. Deskripsi rinci dari masing-masing tabel dapat dilihat pada Lampiran A.

**Tabel 4 Daftar tabel SI PMB PA2**

Nama Tabel	Primary Key	Deskripsi Isi
tgroup	GroupID	Data <i>group</i> pengguna aplikasi.
tuser	Email	Data pengguna aplikasi yang telah memiliki <i>account</i> (dapat melakukan <i>login</i> ).
tcalonmahasiswa	Email	Data calon mahasiswa.
tberita	ID_Berita	Data berita yang terdapat yang diumunkan oleh <i>administrator</i> .
tFAQ	ID_FAQ	Data pertanyaan pengunjung.
tevent	ID_Event	Data peristiwa yang berhubungan dengan Politeknik Informatika Del.
thn_pend	Tahun_Akademik	Data tahun akademik

## Bab 3 Analisis

Pada bab ini diuraikan kegiatan analisis yang dilakukan pada SI PMB TA2008 terhadap *design pattern* yang diterapkan dan tidak diterapkan pada kelas diagram.

### 3.1 Analisis dan Spesifikasi Kebutuhan

Berdasarkan hasil studi yang telah dilakukan pada SI PMB PI Del dan SI PMB PA2 maka diuraikan hasil analisis terhadap SI PMB TA2008.

#### 1. Analisis kelompok pengguna sistem informasi

Pengguna sistem informasi dikelompokkan berdasarkan peran dan deskripsi tugas. Berikut diuraikan gambaran setiap pengguna sistem.

- a. *User*, merupakan generalisasi dari keempat pengguna yang lain. *User* dapat melihat *event*, berita, *FAQ* dan agenda.
- b. *Guest*, merupakan semua pengguna yang mengunjungi SI PMB tetapi hanya dapat melakukan beberapa fungsi saja selain apa yang telah digeneralisasi. *Guest* dapat melakukan registrasi pendaftaran dan menambah *FAQ*.
- c. *Peserta*, merupakan semua *guest* yang telah melakukan pendaftaran dan statusnya telah diubah menjadi peserta oleh *operator*. Setelah status pendaftar diubah menjadi peserta ujian, peserta memiliki *account* untuk menggunakan aplikasi. *Account* tersebut diberitahukan oleh *operator*, dengan mengirimkan konfirmasi ke email peserta.
- d. *Operator*, merupakan bagian administrasi PI Del. *Operator* memiliki *account* sendiri dalam menggunakan sistem. *Operator* dapat melihat data pendaftar, peserta yang lulus, mengirimkan konfirmasi, mencetak kartu ujian dan mengubah status calon pendaftar.
- e. *Committee*, merupakan ketua panitia PMB.

*Committee* memiliki hak akses penuh terhadap aplikasi dan memiliki *account* khusus dalam menggunakan sistem. *Committee* dapat melihat, mengedit, menambah data *event*, berita, *FAQ*, agenda, jadwal ujian, mata pelajaran, lokasi ujian. *Committee* juga memiliki akses untuk meng-*entry* nilai, menjawab *FAQ* dan menjawab tanggapan.

## 2. Spesifikasi kebutuhan

Pada hasil analisis terhadap aplikasi SI PMB PI Del dan SI PMB PA2 terdapat penambahan fungsi. Berikut diuraikan seluruh daftar fungsional pada SI PMB TA2008.

**Tabel 5 Daftar spesifikasi fungsi SI PMB TA2008**

No.	Fungsi/Fitur Utama	Kemampuan Software
1	<i>Login</i>	Melakukan autentikasi <i>user</i> .
2	<i>Logout</i>	Memutuskan koneksi <i>user</i> ke database.
3	<i>ViewEvent</i>	Menampilkan <i>event</i> .
4	<i>ViewBerita</i>	Menampilkan berita.
5	<i>ViewFAQ</i>	Menampilkan FAQ.
6	<i>ViewAgenda</i>	Menampilkan agenda.
7	<i>AddFAQ</i>	Menambahkan FAQ.
8	<i>MakeRegistration</i>	Menerima data yang dientry oleh <i>user</i> .
9	<i>EditDataPeserta</i>	Menerima data yang telah diubah oleh <i>user</i> .
10	<i>BeriTanggapan</i>	Menerima data yang dientry oleh <i>user</i> .
11	<i>ViewHasilUjian</i>	Menampilkan nilai ujian.
12	<i>ViewPesertaYangLulus</i>	Menampilkan data peserta yang lulus.
13	<i>ViewDataPendaftar</i>	Menampilkan detail data pendaftar.
14	<i>PrintKartuUjian</i>	Mencetak kartu ujian.
15	<i>ChangeStatus</i>	Menerima data perubahan untuk status pendaftar.
16	<i>SendKonfirmasi</i>	Mengirimkan pemberitahuan.
17	<i>ViewStatistik</i>	Menampilkan statistik.
18	<i>AddUser</i>	Menerima data <i>user</i> jika terdapat penambahan <i>user</i> .
19	<i>EditUser</i>	Menerima data <i>user</i> yang di <i>edit</i> .
20	<i>AddBerita</i>	Menerima data yang di <i>entry</i> jika isi berita ditambah.
21	<i>EditBerita</i>	Menerima data berita yang di <i>edit</i> .
22	<i>AddAgenda</i>	Menerima data agenda jika terdapat penambahan agenda.
23	<i>EditAgenda</i>	Menerima data agenda yang di <i>edit</i> .
24	<i>AddEvent</i>	Menerima data <i>event</i> jika terdapat penambahan <i>event</i> .
25	<i>EditEvent</i>	Menerima data <i>event</i> yang di <i>edit</i> .
26	<i>AddMataUjian</i>	Menerima data mata ujian jika terdapat penambahan mata ujian.
27	<i>EditMataUjian</i>	Menerima data mata ujian yang di <i>edit</i> .
28	<i>AddJadwalUjian</i>	Menerima data jadwal ujian jika terdapat penambahan jadwal ujian.

No.	Fungsi/Fitur Utama	Kemampuan Software
29	<i>EditJadwalUjian</i>	Menerima data jadwal ujian yang di jadwal ujian.
30	<i>AddLokasiUjian</i>	Menerima data lokasi ujian jika terdapat penambahan lokasi ujian.
31	<i>EditLokasiUjian</i>	Menerima data lokasi ujian yang di lokasi ujian.
32	<i>InsertNilai</i>	Menerima data nilai yang di <i>entry</i> .
33	<i>AnswerFAQ</i>	Menerima data FAQ yang di <i>entry</i> .
34	<i>EditFAQ</i>	Menerima data <i>entry</i> yang telah di <i>edit</i> .
35	<i>AnswerTanggapan</i>	Menerima data tanggapan yang di <i>entry</i> .

Berdasarkan hasil analisis terhadap spesifikasi fungsi maka fungsi-fungsi tersebut dapat dikategorikan berdasarkan kesamaan prilaku. Pengkategorian dilakukan karena *design pattern* dilakukan dengan melihat kesamaan fungsi.

1. Kategori menampilkan data terdiri dari :
  - a. *ViewAgenda*
  - b. *ViewBerita*
  - c. *ViewDataPendaftar*
  - d. *ViewEvent*
  - e. *ViewFAQ*
  - f. *ViewHasilUjian*
  - g. *ViewPesertaYangLulus*
  - h. *ViewStatistik*
2. Kategori mengedit data terdiri dari :
  - a. *EditAgenda*
  - b. *EditBerita*
  - c. *EditDataPeserta*
  - d. *EditEvent*
  - e. *EditFAQ*
  - f. *EditJadwalUjian*
  - g. *EditLokasiUjian*
  - h. *EditMataPelajaran*
  - i. *EditUser*

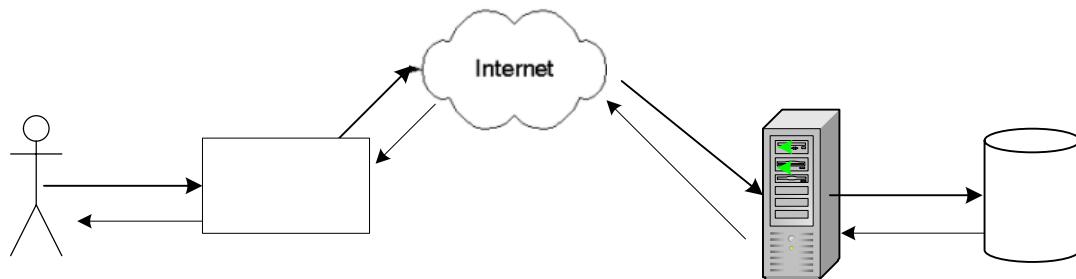
3. Kategori menambah data terdiri dari :

- a. *AddAgenda*
- b. *AddBerita*
- c. *AddEvent*
- d. *AddFAQ*
- e. *AddJadwalUjian*
- f. *AddLokasiUjian*
- g. *AddMataPelajaran*
- h. *AddUser*

Pengelompokan user dan daftar fungsi yang telah diuraikan pada bab 3.1 mengenai analisis terhadap SI PMB TA2008 telah dimodelkan dengan UML yaitu *usecase diagram* dan dapat dilihat pada Lampiran B.

### 3.2 Arsitektur Aplikasi Web

SI PMB TA2008 yang dirancang adalah aplikasi yang berbasis web. Pada bab ini dijelaskan arsitektur aplikasi web yang secara umum akan diterapkan dalam perancangan SI PMB TA2008. Berikut adalah gambar arsitektur aplikasi web.



Gambar 29 Arsitektur aplikasi web

Aplikasi web adalah sebuah aplikasi yang diakses dengan menggunakan *browser* dan dikirim dalam bentuk *form* melalui jaringan Internet atau Intranet.

Skenario penggunaan aplikasi web.

1. *User* melakukan aksi terhadap aplikasi melalui web *browser*. Fungsi utama *interface* adalah menterjemahkan *task and result* untuk menjadi sesuatu yang dapat dimengerti oleh *user*.
2. Web *browser* mengirimkan *request* ke *application server* melalui jaringan seperti internet atau intranet dan melayani request tersebut.

3. *Application server* melayani *request* tersebut dengan membuat *query* dan *update* ke database.
4. Hasil *query* dan *update* dikembalikan ke *application server*.
5. Hasil *query* dan *update* dikirimkan ke web browser dan men-generate *form*
6. *Form* ditampilkan pada web browser sehingga dapat dimengerti oleh *user*.

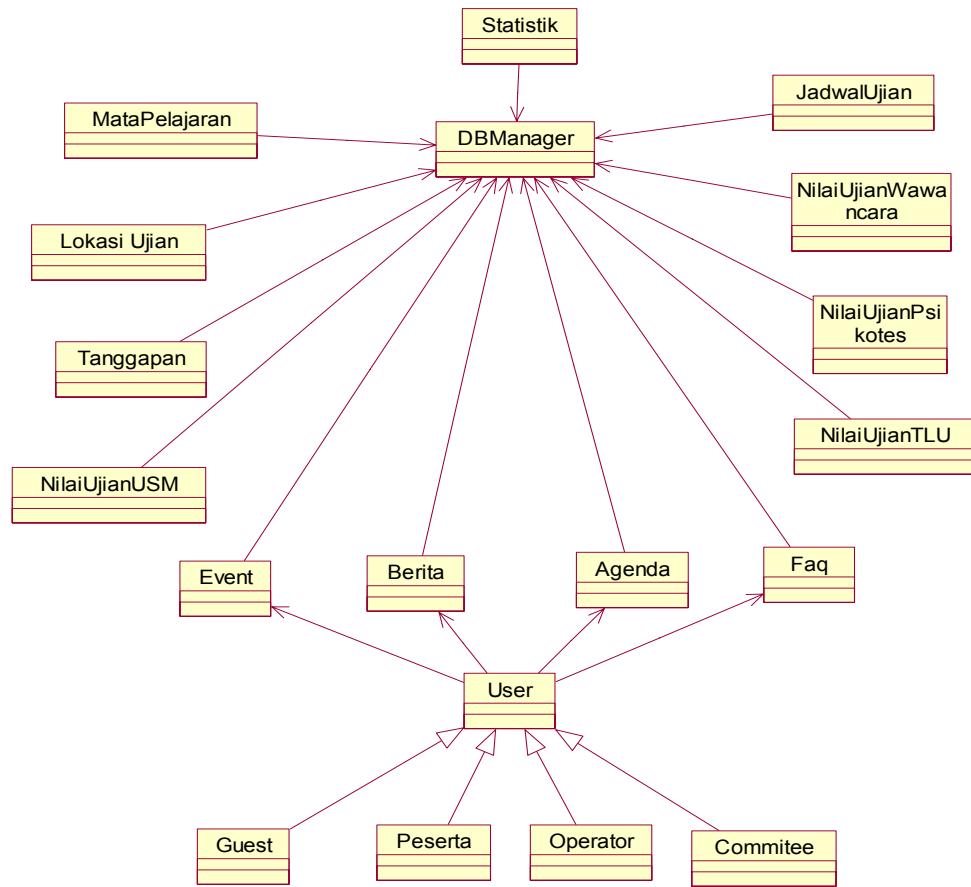
Berdasarkan skenario diatas, web *browser* membentuk koneksi ke database *server* yang harus ditampilkan oleh aplikasi. Database *server* diwakili oleh kelas `DbManager` pada rancangan kelas SI PMB TA2008.

### **3.3 Rancangan Kelas Tanpa Pattern**

Rancangan kelas dihasilkan setelah mempelajari sistem *existing*. Kelas dibentuk berdasarkan daftar fungsi yang *didefine* untuk SI PMB TA 2008. Fungsi yang telah *didefine* adalah fungsi-fungsi yang memiliki potensi untuk dibentuk menjadi tabel.

Pada fungsi *AddEvent*, *EditEvent* dan *ViewEvent* diidentifikasi memiliki atribut yang menggambarkan fungsi tersebut. Dengan demikian, fungsi tersebut dibentuk menjadi sebuah kelas entiti yaitu kelas `Event`. Demikian halnya dengan fungsi-fungsi yang telah diuraikan pada subbab 3.1 analisis dan spesifikasi kebutuhan. Kelas-kelas entiti yang terbentuk yaitu `MataUjian`, `LokasiUjian`, `Tanggapan`, `NilaiUjianUSM`, `Berita`, `Agenda`, `FAQ`, `JadwalUjian`, `NilaiUjianWawancara`, `NilaiUjianPsikotes`, `NilaiUjianTLU` dan `User`.

Berikut adalah diagram kelas SI PMB TA 2008 berdasarkan daftar fungsi yang telah diuraikan pada Tabel 2.



Gambar 30 Rancangan kelas SI PMB TA2008

### 3.4 Penerapan Design Pattern SI PMB TA 2008

Analisis *design pattern* dilakukan dengan menyesuaikan setiap pola desain dengan relasi antar kelas yang terdapat pada diagram kelas SI PMB TA2008.

Berikut adalah hasil analisis terhadap *design pattern* untuk menetapkan *design pattern* yang dapat diterapkan pada diagram kelas SI PMB TA2008.

### 3.4.1 Creational Patterns

Setiap pola pada *creational patterns* diuraikan sebagai berikut.

Nama Pattern	Keterangan	Status	Alasan
Abstract Factory	Menciptakan sekumpulan objek yang berhubungan atau saling tergantung, dengan menyediakan suatu antar muka.	Tidak digunakan	Tidak ada kondisi penciptaan objek dan semua kelas yang ada pada rancangan SI PMB adalah kelas konkret.
Builder	Memisahkan objek yang kompleks dari bagian-bagiannya sehingga proses pembangunan yang sama dapat menciptakan gambaran yang berbeda.	Tidak digunakan	Tidak terdapat suatu objek yang kompleks sehingga tidak ada kondisi <i>builder pattern</i> diterapkan pada rancangan kelas SI PMB TA 2008.
Factory Method	Menciptakan objek-objek dengan menyediakan suatu antarmuka yang memungkinkan subkelas-subkelas untuk memutuskan objek mana yang akan dicitakan.	Tidak digunakan	Tidak ada kondisi penciptaan objek sehingga tidak ada kondisi <i>factory method pattern</i> diterapkan pada rancangan kelas SI PMB TA 2008.
Prototype	Menentukan jenis-jenis objek untuk menciptakan bentuk dasar dari instansiasi dan menciptakan objek baru.	Tidak digunakan	Tidak ada kondisi penciptaan objek baru sehingga tidak ada kondisi <i>prototype pattern</i> diterapkan pada rancangan kelas SI PMB TA 2008.
Singleton	Tidak menciptakan objek, tetapi memastikan sebuah kelas hanya memiliki satu instansiasi dan menyediakan satu titik akses global kepadanya.	Digunakan	Method pada kelas DbManager akan digunakan secara berulang dan menyebabkan instansiasi yang terus-menerus ketika kelas DbManager dipanggil. Maka diterapkan singleton pattern agar kelas DbManager hanya memiliki satu instansiasi yang menyediakan akses global sehingga dapat diakses oleh <i>clients</i> .

### 3.4.2 Structural Patterns

Setiap pola pada *structural patterns* diuraikan sebagai berikut.

Nama Pattern	Keterangan	Status	Alasan
Adapter	Mengubah antarmuka sebuah kelas kepada kelas lain yang diharapkan klien, agar memungkinkan kelas-kelas tersebut bekerjasama .	Tidak digunakan	Setiap kelas dapat bekerjasama karena memiliki antarmuka yang cocok sehingga tidak ada kondisi <i>adapter pattern</i> diterapkan pada rancangan kelas SI PMB TA 2008.

Nama Pattern	Keterangan	Status	Alasan
Bridge	Memisahkan sebuah abstraksi dan implementasinya sehingga keduanya berbeda dan tidak saling tergantung.	Tidak digunakan	Pada rancangan SI PMB TA 2008 semua kelas dibuat konkret sehingga tidak ada kelas abstraksi yang harus dipisahkan dengan implementasinya.
Composite	Menyusun objek-objek kedalam struktur pohon untuk mewakili hierarki seluruh-sebagian.	Tidak digunakan	Perbedaan antara klien memperlakukan objek tunggal dan klien memperlakukan komposisi tidak terlalu kelihatan perbedaannya pada SI PMB TA 2008.
Decorator	Secara dinamis memberikan tambahan tanggungjawab kepada sebuah objek.	Digunakan	Pada SI PMB TA 2008 ada beberapa kelas yang memiliki tanggung jawab yang sama sehingga dibutuhkan suatu mekanisme perluasan yang mudah disesuaikan
Facade	Memberikan sebuah kesatuan antarmuka kepada suatu himpunan antarmuka didalam sebuah subsistem.	Tidak digunakan	Antarmuka yang disediakan pada rancangan kelas SI PMB TA 2008 tidak rumit sehingga tidak diperlukan penetapan antarmuka yang lebih tinggi.
Flyweight	Penggunaan objek secara bersama-sama untuk memungkinkan sejumlah besar objek-objek yang berukuran kecil dapat ditangani secara tepatgunya.	Tidak digunakan	Pada SI PMB TA 2008 objek-objek kecil dikontrol secara langsung oleh DbManager.
Proxy	Menyediakan pengganti objek lain untuk mengontrol pengaksesan.	Tidak digunakan	Pada SI PMB yang mengontrol pengaksesan adalah kelas DbManager yang memiliki <i>method connect()</i> , sehingga tidak perlu menyediakan pengganti objek untuk mengontrol pengaksesan.

### 3.4.3 Behavioral Patterns

Setiap pola pada *behavioral patterns* diuraikan sebagai berikut.

Nama Pattern	Keterangan	Status	Alasan
Chain of Responsibility	Memisahkan pengirim dari permintaan kepada penerima dengan memberikan kesempatan kepada lebih dari satu objek untuk menangani permintaan tersebut.	Tidak digunakan	Pada SI PMB TA 2008 setiap permintaan langsung ditangani oleh DbManager, sehingga tidak perlu memberikan kesempatan kepada objek lain untuk menangani permintaan tersebut.

Nama Pattern	Keterangan	Status	Alasan
Command	Memparameterkan klien dengan permintaan-permintaan yang berbeda dengan cara membungkus sebuah permintaan di dalam sebuah objek.	Tidak digunakan	Klien dikenali menurut role yang diberikan sehingga tidak perlu harus memparameterkan klien dan hanya melalui GUI.
Interpreter	Memberikan sebuah bahasa, mendefinisikan gambaran untuk tatabahasanya dengan sebuah interprete sehingga menggunakan gambaran untuk menerjemahkan kalimat dalam bahasa.	Tidak digunakan	Pada rancangan SI PMB TA 2008 tidak terdapat proses mendefinisikan <i>grammar</i> untuk bahasa sederhana ( <i>interface</i> ).
Iterator	Mengakses unsur-unsur dari sekumpulan objek secara berurutan. Iterator memberikan jalan untuk mengakses unsur-unsur tanpa menyingkap implementasi yang mendasarinya.	Tidak digunakan	Pada SI PMB TA 2008 elemen-elemen dapat diakses secara langsung.
Mediator	Mendefinisikan sebuah objek yang membungkus sekumpulan interaksi objek	Digunakan	Pada SI PMB TA 2008 terdapat elemen sama pada form untuk memasukkan nilai sehingga mediator digunakan untuk membungkus elemen yang sama.
Memento	Menangkap dan mengeluarkan <i>state</i> pada suatu objek sehingga <i>state</i> tersebut dapat dikembalikan diakhir tanpa melanggar <i>encapsulation</i> .	Tidak digunakan	Tidak ada <i>state</i> yang harus di <i>return</i> .
Observer	Semua objek yang bergantung diberitahukan serta diperbarui ketika suatu objek berganti <i>state</i> .	Tidak digunakan	Tidak ada kondisi suatu objek yang berganti <i>state</i> atau keadaan.
State	Mengizinkan sebuah objek untuk mengubah tingkah laku ketika <i>state</i> internalnya berubah.	Tidak digunakan	Tidak ada perubahan <i>state</i> , dan tingkah laku <i>state</i> pada setiap objek sudah ditetapkan.
Strategy	Membuat algoritma-algoritma dari suatu kumpulan algoritma dapat dipertukarkan dengan cara membungkus masing-masing algoritma.	Tidak digunakan	Algoritma tidak digunakan pada rancangan SI PMB TA 2008 karena setiap kelas akan dikode sendiri.
Template	Menunda langkah-langkah	Tidak	Algoritma tidak digunakan

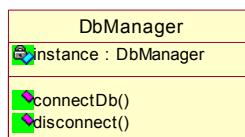
<b>Nama Pattern</b>	<b>Keterangan</b>	<b>Status</b>	<b>Alasan</b>
Method	suatu algoritma ke subkelas-subkelas dengan cara menetapkan rangka dari suatu algoritma di satu operasi.	digunakan	sehingga tidak ada penundaan langkah-langkah suatu algoritma pada rancangan SI PMB TA 2008.
Visitor	Menghadirkan sebuah operasi yang akan dijalankan didalam elemen-elemen dari sebuah struktur objek, tanpa mengubah kelas dari elemen-elemen tersebut.	Tidak digunakan	Tidak ada operasi yang dihadirkan untuk dijalankan pada elemen-elemen.

## Bab 4 Penerapan Design Pattern

Pada bab rancangan diuraikan mengenai rancangan kelas SI PMB TA 2008 yang telah menerapkan *design pattern*. Berdasarkan hasil analisis terhadap *design pattern* yang telah diuraikan pada sub bab 3.3 analisis penerapan *design pattern*, ada 3 *pattern* yang diterapkan pada diagram kelas SI PMB TA 2008. Berikut diuraikan penerapan *design pattern* terhadap diagram kelas SI PMB TA 2008.

### 4.1 Singleton pattern

Berikut adalah gambar diagram kelas dengan menerapkan *singleton pattern*.



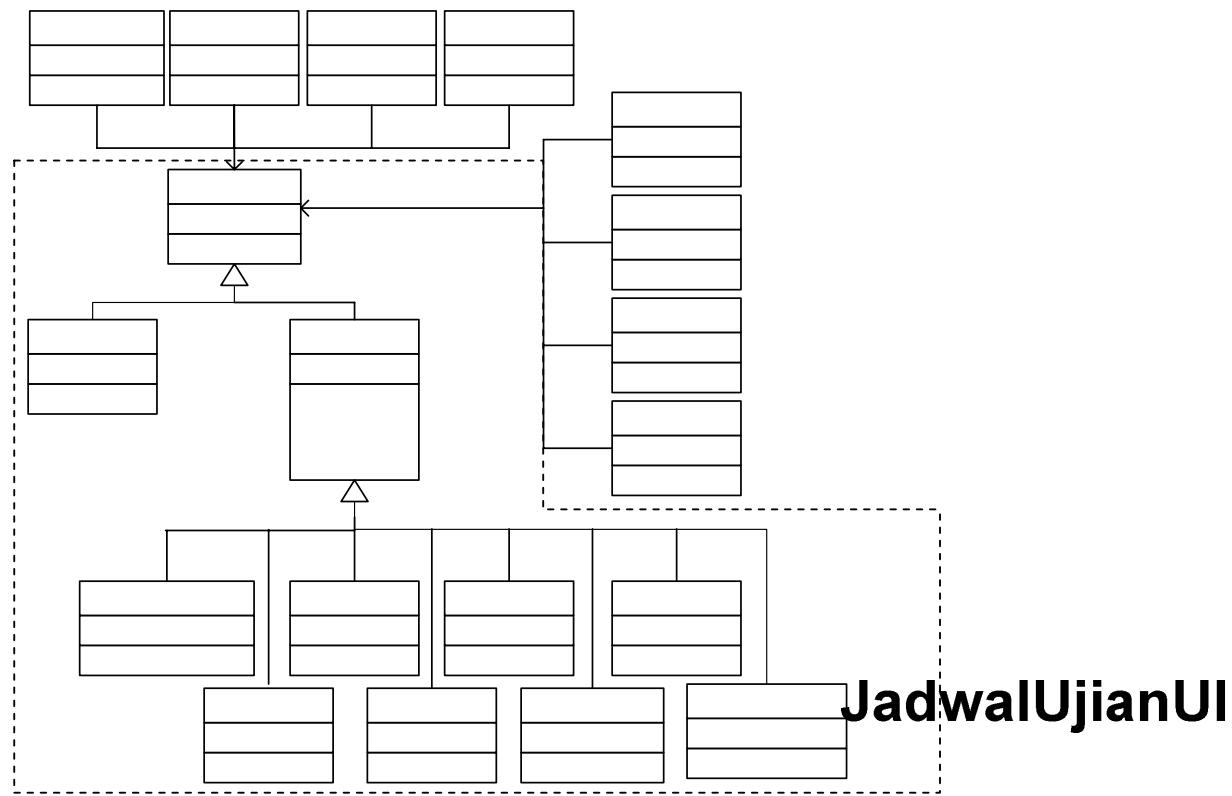
Gambar 31 Diagram kelas yang menerapkan *singleton pattern*

Proses penerapan *singleton pattern* diawali dengan mempelajari relasi antar kelas. Pada kelas FAQ, NilaiUSM, NilaiITLU, NilaiPsikotest, NilaiWawancara, Commitee, Pendaftar, LokasiUjian, MataUjian, Tanggapan, Event, Berita, Agenda, dan JadwalUjian terhubung dengan kelas DbManager. Kelas DbManager memiliki *method* koneksi ke database yaitu `connectDb()` yaitu membuka koneksi ke database untuk mengelola data.

Koneksi yang terjadi secara terus-menerus mengakibatkan terjadinya instansiasi yang berulang pada seluruh kelas yang terhubung ke kelas DbManager. Penginstansiasian secara berulang dapat diatasi dengan menerapkan *singleton pattern*. *Pattern* ini menjamin sebuah kelas hanya memiliki satu instansiasi. Cara penerapannya adalah dengan menciptakan sebuah objek pada kelas DbManager dan objek tersebut merupakan instansiasi dirinya sendiri. Instansiasi objek dapat diakses secara global oleh kelas-kelas yang membutuhkan koneksi ke database.

#### 4.2 Decorator pattern

Berikut adalah gambar diagram kelas dengan menerapkan *decorator pattern*.



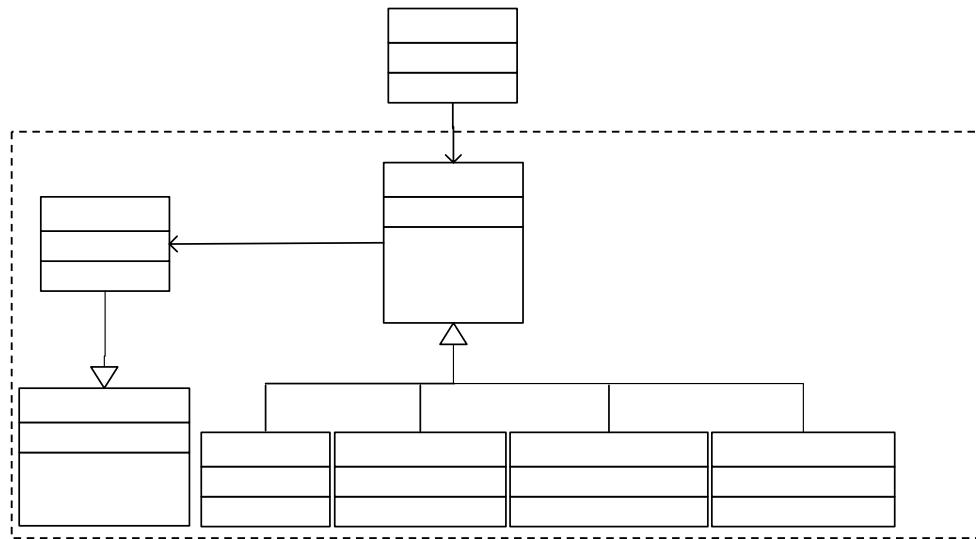
Gambar 32 Diagram kelas yang menerapkan *decorator pattern*

Berdasarkan hasil analisis pada rancangan kelas, terdapat beberapa kelas yang memiliki *method* yang sama. Kelas *form* terdiri dari *FormJadwalUjian*, *FormBerita*, *FormFAQ*, *FormUser*, *FormEvent*, *FormMataUjian*, *FormAgenda* dan *FormLokasiUjian*. Kelas *form* adalah kelas-kelas yang memiliki *method* yang sama seperti *submit()*, *cancel()*, *getData()* dan *displayData()*. Cara penerapan *decorator pattern* adalah dengan menciptakan kelas *Component*, *Decorator* dan *ShowForm*. Seluruh *method* yang telah disebutkan akan *didefine* pada kelas *Decorator*, yang berfungsi untuk memelihara referensi ke objek *Component*. Kemudian kelas *BeritaUI*, *MataPelajaranUI*, *LoaksiUjianUI*, *AgendaUI*, *FAQUI*, *JadwalUjianUI*, *EventUI* dan *ManagementUserUI* merupakan kelas-kelas yang memiliki *method* yang sama yaitu *displayForm()*. *Method* *displayForm()* *didefine* kedalam kelas *Component*.

#### 4.3 Mediator pattern

Berikut adalah gambar diagram kelas dengan menerapkan *mediator pattern*.

ShowForm



Gambar 33 Diagram kelas yang menerapkan *mediator pattern*

Proses penerapan *mediator pattern* adalah menciptakan kelas *AbstractFormNilai* yang mengimplementasikan *method* yang sama seperti *submit()*, *displayFrom()* dan *displayData()* yang dimiliki oleh kelas-kelas *FormNilaiWawancara*, *FormNilaiAkademik*, *FormNilaiPsikotes* dan *FormNilaiTlu*. Kelas *AbstracFormNilai* berperan mendefinisikan *interface* untuk berkomunikasi dengan kelas *FormNilaiUjian*.

Gambar keseluruhan diagram kelas *form* sebelum dan setelah menerapkan *design pattern* dapat dilihat pada Lampiran C.

## **FormNilai**

### ***AbstractFormNilai***

-noUjian

+submit()

+displayForm()

+displayData()

## Bab 5 Kesimpulan dan Saran

Dengan memperhatikan proses penggerjaan tugas akhir dan melihat hasil rancangan pada bab 4 maka dapat diuraikan kesimpulan dan saran yang berguna sebagai sarana perbaikan di masa yang akan datang.

### 5.1 Kesimpulan

Pada akhir penulisan tugas akhir ini diperoleh beberapa kesimpulan, yaitu:

1. Penerapan *design pattern* pada rancangan kelas SI PMB TA2008 dapat dikatakan kurang maksimal. Hal ini disebabkan karena ada banyak *pattern* yang disediakan namun sulit dipahami. Pada tugas akhir ini ada 3 *design pattern* yang telah diidentifikasi.
2. Pada rancangan SI PMB TA2008 telah diterapkan 3 *pattern*. Pada dasarnya dapat diterapkan banyak *pattern* pada SI PMB TA2008, namun ketika *pattern* tersebut ingin diterapkan terdapat beberapa kondisi pada rancangan kelas yang tidak sesuai dengan *applicability* yang disediakan oleh *pattern*.
3. Penerapan *design pattern* terhadap rancangan SI PMB TA2008 menyebabkan penambahan kelas. Penambahan kelas tidak memberikan pengaruh yang besar terhadap rancangan kelas yang lain.

### 5.2 Saran

Untuk memperoleh hasil yang maksimal dalam pengembangan SI PMB TA2008, penulis menyarankan agar hasil rancangan terhadap SI PMB TA2008 dengan penerapan *design pattern* dapat diimplementasikan menjadi kode pada suatu bahasa pemrograman. Bahasa yang disarankan adalah bahasa pemrograman yang mendukung *object oriented* seperti java, php, c# dan sebagainya.