

MODUL PRAKTIKUM

RPL : DESAIN DAN IMPLEMENTASI

S1 INFORMATIKA



Created By Jati Hiliamsyah Husein
Published By School Of Computing



LEMBAR PENGESAHAN

Saya yang bertanda tangan di bawah ini:

Nama : Eko Darwiyanto, S.T., M.T.
NIP : 13861144-1
Koordinator Mata Kuliah : Desain dan Implementasi Perangkat Lunak
Prodi : S1 Informatika

Menerangkan dengan sesungguhnya bahwa modul ini digunakan untuk pelaksanaan praktikum di Semester Genap Tahun Ajaran 2021/2022 di Laboratorium Informatika Fakultas Informatika Universitas Telkom.

Bandung, 15 Februari 2022

Mengesahkan,
Koordinator MK Berpraktek Desain dan Implementasi
Perangkat Lunak



Eko Darwiyanto, S.T., M.T.

Mengetahui,
Kaprodi S1 Informatika



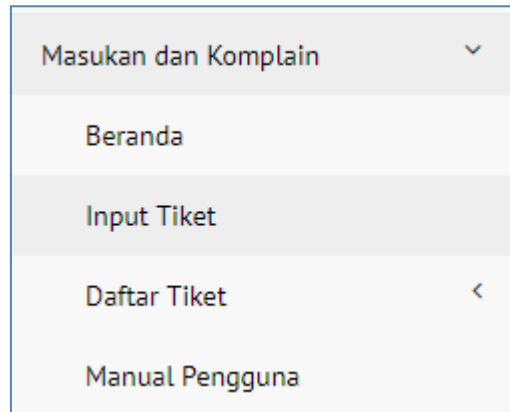
Dr. Erwin Budi Setiawan, S.Si., M.T.

Peraturan Praktikum Laboratorium Informatika 2021/2022

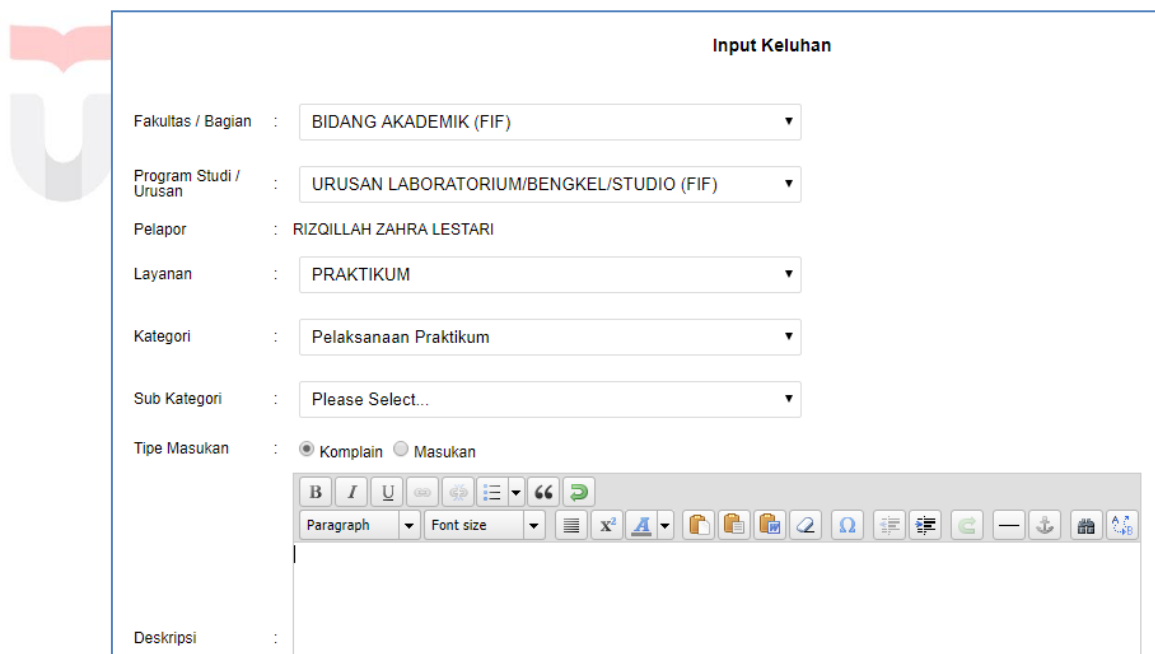
1. Praktikum diampu oleh dosen kelas dan dibantu oleh asisten laboratorium dan asisten praktikum.
2. Praktikum dilaksanakan di Gedung TULT Lantai 6 (IFLAB 1 s/d IFLAB 5) dan Gedung Kultubai Utara (IFLAB 6 s/d IFLAB 7) sesuai jadwal yang ditentukan.
3. Praktikan wajib membawa modul praktikum, kartu praktikum, dan alat tulis.
4. Praktikan wajib mengisi daftar hadir *rooster* dan BAP praktikum dengan bolpoin bertinta hitam.
5. Durasi kegiatan praktikum S-1 = 2 jam (100 menit).
6. Jumlah pertemuan praktikum:
 - 14 kali di lab (praktikum rutin)
 - 2 kali di luar lab (terkait Pemantauan dan Presentasi Tugas Besar)
7. Praktikan wajib hadir minimal 75% dari seluruh pertemuan praktikum di lab. Jika total kehadiran kurang dari 75% maka nilai UAS/ Tugas Besar = 0.
8. Praktikan yang datang terlambat :
 - ≤ 5 menit : diperbolehkan mengikuti praktikum tanpa tambahan praktikum
 - > 5 menit : tidak diperbolehkan mengikuti praktikum
9. Saat praktikum berlangsung, asisten praktikum dan praktikan:
 - Wajib menggunakan seragam sesuai aturan institusi.
 - Wajib mematikan/ mengkondisikan semua alat komunikasi.
 - Dilarang membuka aplikasi yang tidak berhubungan dengan praktikum yang berlangsung.
 - Dilarang mengubah pengaturan *software* maupun *hardware* komputer tanpa ijin.
 - Dilarang membawa makanan maupun minuman di ruang praktikum.
 - Dilarang memberikan jawaban ke praktikan lain.
 - Dilarang menyebarkan soal praktikum.
 - Dilarang membuang sampah di ruangan praktikum.
 - Wajib meletakkan alas kaki dengan rapi pada tempat yang telah disediakan.
10. Setiap praktikan dapat mengikuti praktikum susulan maksimal dua modul untuk satu mata kuliah praktikum.
 - Praktikan yang dapat mengikuti praktikum susulan hanyalah praktikan yang memenuhi syarat sesuai ketentuan institusi, yaitu: sakit (dibuktikan dengan surat keterangan medis), tugas dari institusi (dibuktikan dengan surat dinas atau dispensasi dari institusi), atau mendapat musibah atau keduakaan (menunjukkan surat keterangan dari orangtua/wali mahasiswa.)
 - Persyaratan untuk praktikum susulan diserahkan sesegera mungkin kepada asisten laboratorium untuk keperluan administrasi.
 - Praktikan yang diijinkan menjadi peserta praktikum susulan ditetapkan oleh Asman Lab dan Bengkel Informatika dan tidak dapat diganggu gugat.

Tata Cara Komplain Praktikum IFLab Melalui IGracias

1. Login IGracias
2. Pilih Menu **Masukan dan Komplain**, pilih **Input Tiket**



3. Pilih Fakultas/Bagian: **Bidang Akademik (FIF)**
4. Pilih Program Studi/Urusan: **Urusan Laboratorium/Bengkel/Studio (FIF)**
5. Pilih Layanan: **Praktikum**
6. Pilih Kategori: **Pelaksanaan Praktikum**, lalu pilih **Sub Kategori**.
7. Isi **Deskripsi** sesuai komplain yang ingin disampaikan.



Input Keluhan

Fakultas / Bagian : **BIDANG AKADEMIK (FIF)**

Program Studi / Urusan : **URUSAN LABORATORIUM/BENGKEL/STUDIO (FIF)**

Pelapor : **RIZQILLAH ZAHRA LESTARI**

Layanan : **PRAKTIKUM**

Kategori : **Pelaksanaan Praktikum**

Sub Kategori : **Please Select...**

Tipe Masukan : ☒ **Komplain** ☐ **Masukan**

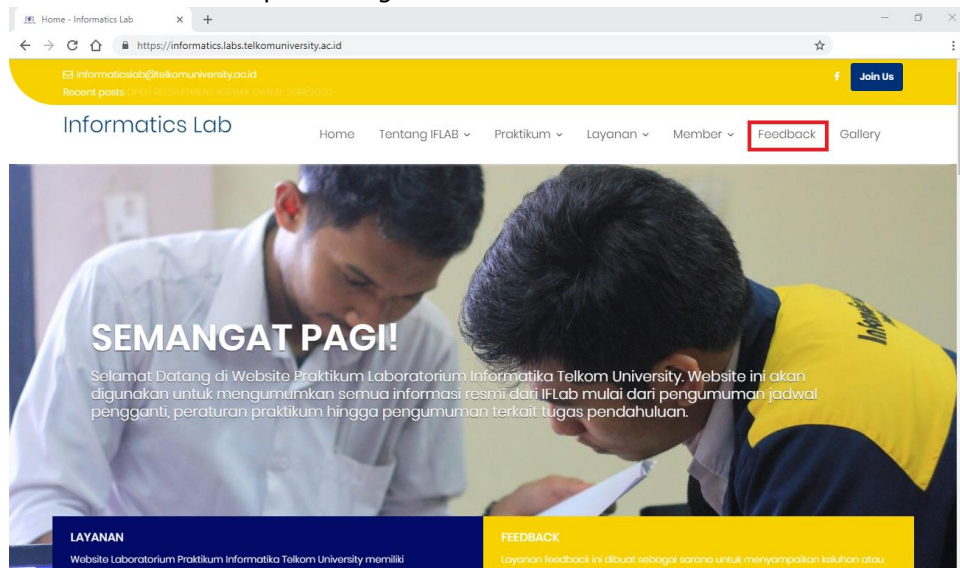
Deskripsi :

[Rich text editor toolbar with buttons for Bold, Italic, Underline, Bulleted List, Numbered List, Indent, Outdent, Link, Unlink, Image, Table, etc.]

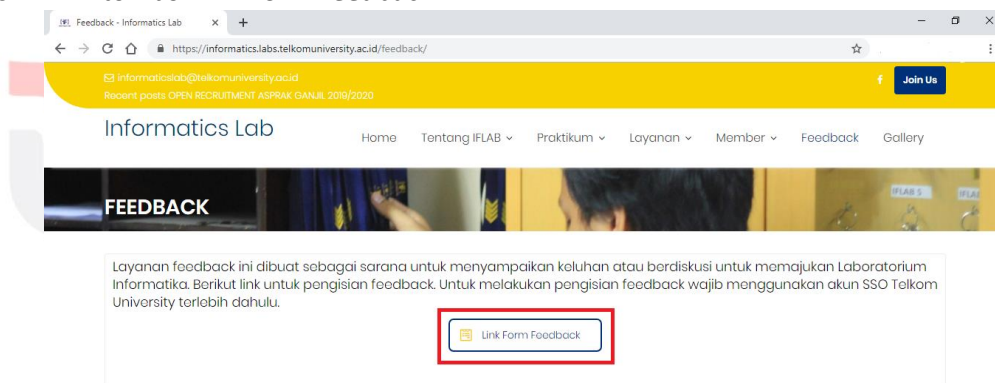
Lampirkan *file* jika perlu. Lalu klik Kirim.

Tata Cara Komplain Praktikum IFLAB Melalui Website

1. Buka website <https://informatics.labs.telkomuniversity.ac.id/> melalui browser.
2. Pilih menu **Feedback** pada *navigation bar website*.



3. Pilih tombol **Link Form Feedback**.



4. Lakukan *login* menggunakan akun **SSO Telkom University** untuk mengakses *form feedback*.
5. Isi *form* sesuai dengan *feedback* yang ingin diberikan.

DAFTAR ISI

LEMBAR PENGESAHAN	i
Peraturan Praktikum Laboratorium Informatika 2021/2022	ii
Tata Cara Komplain Praktikum IFLAB Melalui <i>Website</i>	iv
DAFTAR ISI	v
DAFTAR GAMBAR	i
Modul 1 PENGANTAR PRAKTIKUM DESAIN DAN IMPLEMENTASI PERANGKAT LUNAK	1
1.1 Definisi	1
1.2 Garis Besar Praktikum	1
Modul 2 REVIEW SKPL DAN ALTERNATIF STUDI KASUS	2
2.1 Definisi	2
2.2 Public Software Requirements Specification Dataset	2
Modul 3 INTEGRATED DEVELOPMENT ENVIRONMENT	5
3.1 Definisi	5
3.1.1 Eclipse	5
Modul 4 GITHUB.....	7
4.1 Definisi	7
4.2 Instalasi	7
4.3 Git Commands.....	8
Modul 5 GUI BUILDER.....	10
5.1 Definisi	10
5.2 Implementasi WindowBuilder pada eclipse	10
5.2.1 Membuat Project Baru	10
5.2.2 Menambahkan Windows	11
5.2.3 Menambahkan Komponen Pada Windows	12
Modul 6 MYSQL/MS ACCESS	16
6.1 ER Diagram.....	16
6.2 Primary Key	16
6.3 Normalisasi I, II dan III	17
6.4 Implementasi Basis Data pada MySQL	18
Modul 7 IMPLEMENTASI CLASS DIAGRAM	22
7.1 Implementasi Class Tunggal	22
7.2 Implementasi Relasi Antar Class	23
7.2.1 Association	23
7.2.2 Aggregation	24
7.2.3 Composition	26
Modul 8 IMPLEMENTASI SEQUENCE DIAGRAM	28
8.1 Sequence Diagram	28
8.2 Implementasi Sequence Diagram.....	28
8.3 Contoh Implementasi Sequence Diagram	29
Modul 9 IMPLEMENTASI ROBUSTNESS ANALYSIS	32
9.1 Robustness analysis	32
9.2 Implementasi Robustness analysis.....	32
Modul 10 DOKUMENTASI PERANCANGAN PERANGKAT LUNAK, SOURCE CODE DAN USER MANUAL	36
10.1 Dokumentasi Perangkat Lunak.....	36

10.2	Template Dokumen Perancangan Perangkat Lunak.....	36
10.3	Dokumen <i>Source Code</i>	37
10.4	Dokumen <i>User Manual</i>	37
Modul 11	DESIGN PATTERN	39
11.1	Singleton	40
11.2	Adapter	41
11.3	Command.....	43
Modul 12	REFACTORING	47
12.1	Refactoring.....	47
12.2	Code smells	47
12.3	Refactoring Strategy.....	47
Modul 13	UNIT TESTING.....	52
13.1	Membuat Unit Test	52
Modul 14	PRESENTASI.....	56
14.1	Pengantar.....	56
14.2	Persiapan Presentasi	56
DAFTAR PUSTAKA	57



DAFTAR GAMBAR

Gambar 2.1 Tampilan website hosting dataset PURE	3
Gambar 2.2 Mendownload file requirement.zip.....	3
Gambar 2.3 Extract file requirement.zip.....	4
Gambar 4.1 Tampilan menu option pada Github desktop.....	7
Gambar 4.2 Tampilan sign in akun Github	8
Gambar 4.3 Tampilan input nama dan email pada konfigurasi Git.....	8
Gambar 5.1 Tampilan membuat project baru pada Eclipse	11
Gambar 5.2 Tampilan input nama project pada Eclipse	11
Gambar 5.3 Tampilan window baru dengan WindowBuilder Eclipse	12
Gambar 5.4 Tampilan input nama window baru pada WindowBuilder Eclipse	12
Gambar 5.5 Tampilan daftar komponen yang bisa ditambahkan ke window	13
Gambar 5.6 Tampilan menambahkan komponen ke dalam window.....	13
Gambar 5.7 Tampilan menambahkan parameter komponen yang ditambahkan	14
Gambar 5.8 Tampilan menambahkan event listener pada komponen	14
Gambar 6.1 Contoh gambar ERD Chen	16
Gambar 6.2 Contoh gambar ERD yang melibatkan Primary Key dan Foreign Key	17
Gambar 6.3 Contoh ERD untuk implementasi pada MySQL.....	18
Gambar 6.4 Tabel yang dibuat pada MySQL.....	19
Gambar 6.5 Detail tabel transaksi.....	19
Gambar 6.6 Detail tabel produk.....	20
Gambar 6.7 Detail tabel member	20
Gambar 6.8 Detail tabel detail transaksi.....	21
Gambar 7.1 Contoh class diagram dan implementasinya	22
Gambar 7.2 Contoh class diagram dengan association.....	23
Gambar 7.3 Implementasi kelas Department	23
Gambar 7.4 Implementasi kelas Employee.....	23
Gambar 7.5 Implementasi kelas penghubung Department-Employee	24
Gambar 7.6 Contoh class diagram dengan aggregation.....	24
Gambar 7.7 Implementasi kelas Team.....	25
Gambar 7.8 Implementasi kelas Player	25
Gambar 7.9 Implementasi kelas penghubung Team-Player.....	25
Gambar 7.10 Contoh class diagram dengan composition.....	26
Gambar 7.11 Implementasi kelas Book.....	26
Gambar 7.12 Implementasi kelas Chapter	27
Gambar 8.1 Contoh Sequence Diagram.....	29
Gambar 8.2 Class UI.....	30
Gambar 8.3 Class Segitiga.....	31
Gambar 9.1 Kategori objek pada robusiness analysis.....	32
Gambar 9.2 Sequence diagram add book to cart.....	33
Gambar 9.3 Sequence diagram (Entity Object).....	33
Gambar 9.4 Menambahkan GUI ke Sequence Diagram.....	34
Gambar 9.5 Menambahkan 2 fungsi pada Sequence Diagram.....	34

Gambar 11.1 Struktur Singleton.	40
Gambar 11.2 Hasil code.	43
Gambar 12.1 Contoh <i>dirty code</i>	48
Gambar 12.2 Contoh hasil extract method.	48
Gambar 12.3 Contoh <i>dirty code</i>	49
Gambar 12.4 Class AddressInformation.	50
Gambar 12.5 Class Student.	50
Gambar 13.1 Class Calculator.	52
Gambar 13.2 JUnit Test Case.	53
Gambar 13.3 Class CalculatorTest.	54
Gambar 13.4 Package Explorer window.	54



Fakultas Informatika
School of Computing
Telkom University



Modul 1 PENGANTAR PRAKTIKUM DESAIN DAN IMPLEMENTASI PERANGKAT LUNAK

TUJUAN PRAKTIKUM

1. Memahami posisi praktikum pada MK RPL:Desain dan Implementasi.
2. Memahami garis besar isi kegiatan praktikum MK RPL:Desain dan Implementasi.

1.1 Definisi

Pada praktikum mata kuliah Desain dan Implementasi Perangkat Lunak mahasiswa akan mempelajari bagaimana desain dan implementasi saling berkaitan dalam proses pengembangan perangkat lunak. Desain dan implementasi adalah dua aktivitas yang tidak dapat dihilangkan dalam aktivitas rekayasa perangkat lunak. Keduanya berperan penting dalam menghasilkan perangkat lunak yang berkualitas.

Desain perangkat lunak pada level atas umumnya dibagi menjadi tiga bagian: arsitektur, basis data, dan antarmuka pengguna. Desain arsitektur membahas bagaimana struktur dari perangkat lunak pada level yang tinggi, dimana perangkat lunak dibagi menjadi bagian-bagian yang saling berkomunikasi. Desain basis data memfasilitasi bagaimana struktur basis data yang akan digunakan untuk menyimpan data hasil pengolahan dari perangkat lunak yang dibangun. Terakhir, desain antarmuka pengguna berfokus pada perancangan tampilan dan interaksi dari perangkat lunak yang dapat memudahkan pengguna dalam mengoperasikan perangkat lunak.

Selain desain pada level atas, perangkat lunak juga perlu didesain pada level kode. Penggunaan model UML seperti class dan sequence diagram dapat membantu menggambarkan desain perangkat lunak pada level kode. Selain model UML, penerapan robustness analysis juga dapat diterapkan untuk memastikan desain level kode yang dibuat sudah komplet.

Seluruh jenis desain tersebut akan diimplementasikan pada praktikum ini menjadi sebuah perangkat lunak yang utuh. Selain itu, mahasiswa dituntut untuk dapat menunjukkan keterkaitan antara desain yang sudah dihasilkan dan kode program hasil implementasi yang sudah diterapkan. Untuk memfasilitasi kegiatan implementasi, mahasiswa akan dibekali dengan beberapa tools dan teknik yang dapat digunakan untuk mendesain dan mengimplementasikan perangkat lunak.

1.2 Garis Besar Praktikum

Praktikum yang dilakukan akan membahas topik-topik sebagai berikut:

- 1) Penggunaan IDE dalam implementasi.
- 2) Penggunaan Github sebagai version control dari kode program.
- 3) Penggunaan GUI Builder untuk mengimplementasikan desain antarmuka.
- 4) Penggunaan MySQL/MSAccess untuk mengimplementasikan desain basis data.
- 5) Penerapan desain kode.
- 6) Pendokumentasian desain perangkat lunak.
- 7) Penggunaan design pattern untuk menghasilkan desain yang baik.
- 8) Penerapan refactoring untuk menghasilkan kode yang dapat dipelihara.
- 9) Penerapan unit testing untuk memastikan kebenaran dari kode yang dibuat.

Modul 2 REVIEW SKPL DAN ALTERNATIF STUDI KASUS

TUJUAN PRAKTIKUM
1. Memahami kriteria dari SKPL yang layak untuk digunakan dalam perancangan dan implementasi.

2.1 Definisi

Desain dan implementasi yang baik harus didasari dari kebutuhan perangkat lunak yang sudah dipahami oleh tim pengembang. Setiap keputusan perancangan dan implementasi harus dilakukan mengikuti kebutuhan yang sudah terspesifikasi. Tanpa adanya kebutuhan yang dapat diikuti, desain dan implementasi yang sudah dilakukan tidak dapat dipertanggungjawabkan kebenarannya.

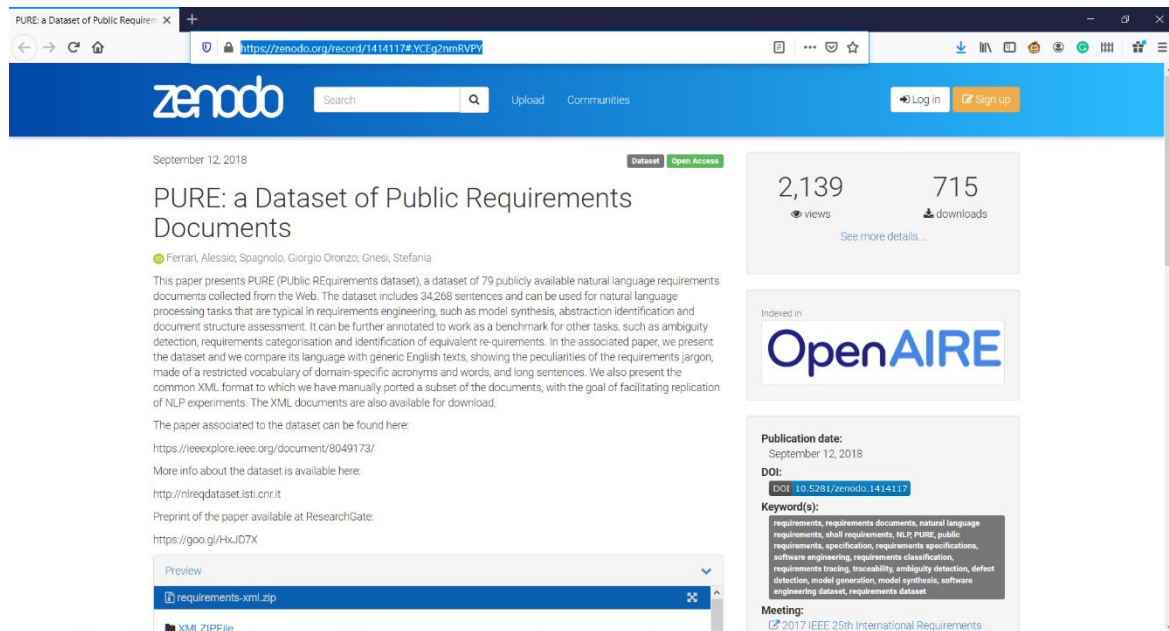
Pada seluruh kegiatan praktikum mata kuliah Desain dan Implementasi Perangkat Lunak mahasiswa diharuskan melakukan desain dan implementasi sesuai dengan kebutuhan. Oleh karena itu, pada praktikum ini mahasiswa diharuskan menggunakan SKPL yang sudah dibuat pada mata kuliah Analisis Kebutuhan Perangkat Lunak atau memilih SKPL yang tersedia secara publik. Jika diperlukan, SKPL yang sudah dibuat dilakukan penyempurnaan dengan kebutuhan untuk menambahkan :

- Jumlah role pengguna aplikasi : diusahakan sampai 3 (tiga) peran yang berbeda
- Jumlah use case dalam Use Case Diagram : diusahakan sampai setidaknya 9 (sembilan) use case
- aspek Kecerdasan Buatan.

2.2 Public Software Requirements Specification Dataset

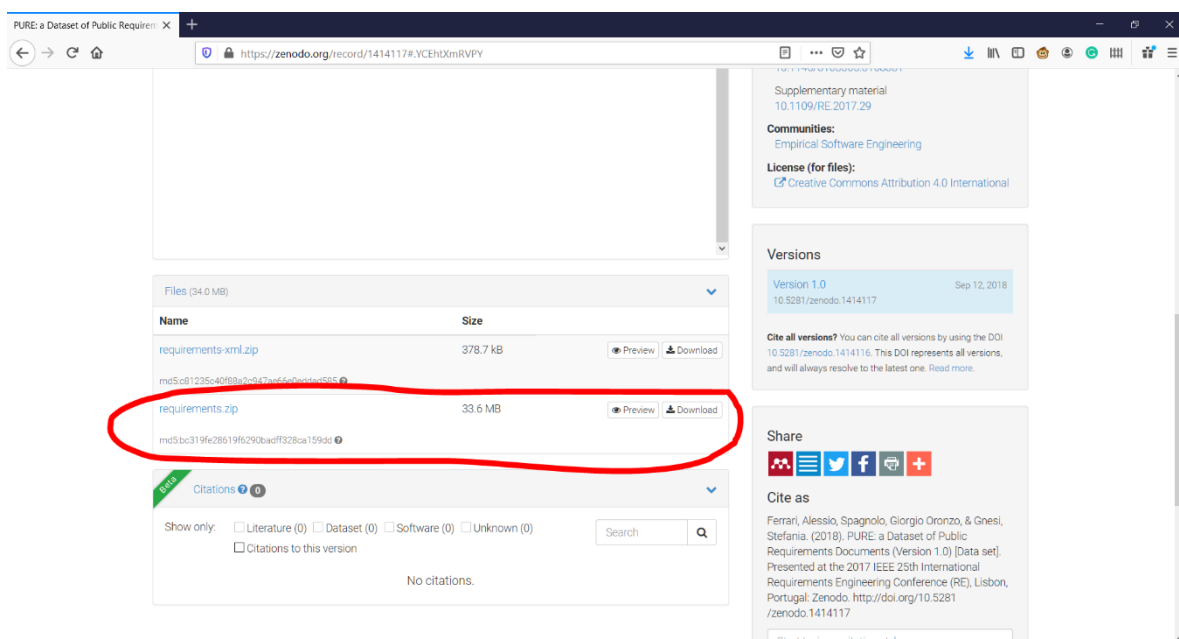
Terdapat sebuah dataset yang berisikan dokumen-dokumen SKPL yang tersedia secara publik di internet. Dataset itu diberi nama PURE (Public REquirements dataset) yang terdiri dari 79 dokumen SKPL. Untuk mengakses dataset tersebut dapat mengikuti langkah-langkah sebagai berikut:

1. Buka website hosting dataset PURE pada <https://zenodo.org/record/1414117#.YCEhtXmRVPY>



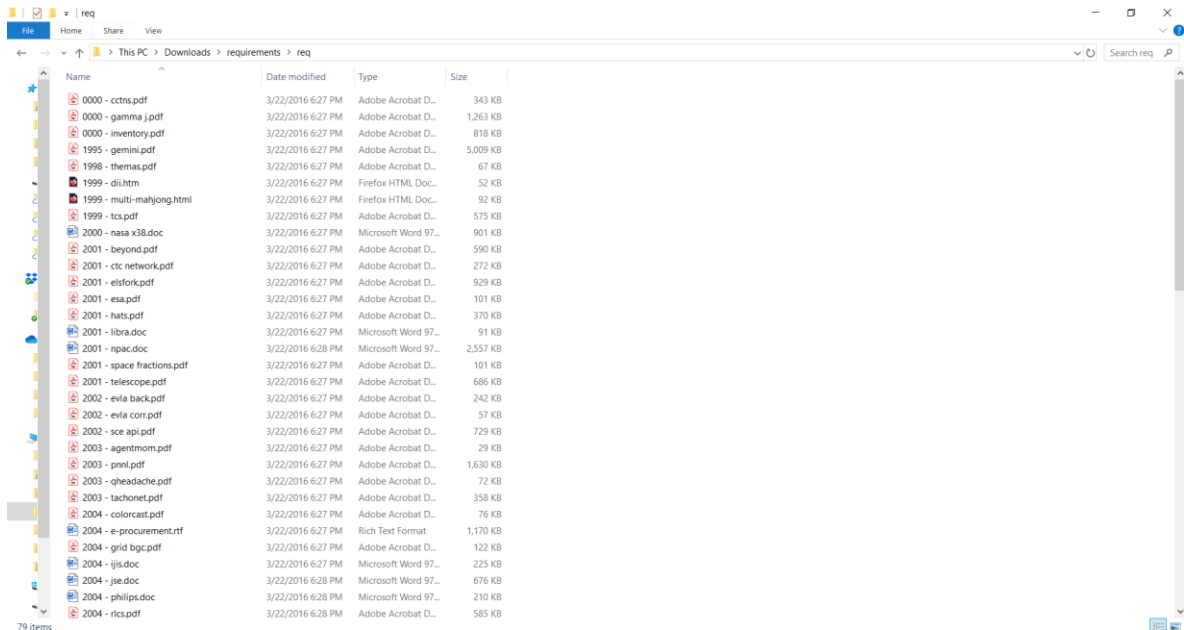
Gambar 2.1 Tampilan website hosting dataset PURE

2. Download file requirements.zip yang tersedia pada laman tersebut.



Gambar 2.2 Mendownload file requirement.zip

3. Extract file requirements.zip yang ada, kemudian buka folder tempat anda mengextract.



Gambar 2.3 Extract file requirement.zip

3. File SKPL yang tersedia dapat dibuka menggunakan aplikasi pengolah kata yang sesuai.

KEGIATAN PRAKTIKUM

1. Pilih salah satu dokumen requirement di <https://zenodo.org/record/1414117#.YCEhtXmRVpY> yang topiknya paling dekat dengan dokumen SKPL Anda
2. Lebih kompleks mana antara dokumen SKPL Anda dengan dokumen requirement yang Anda pilih?
Jelaskan dengan membandingkan keduanya dari sisi:
 - a. jumlah "use case" dalam Use Case Diagram
(Jika Use Case Diagram tidak muncul di dokumen, buatlah)
 - b. jumlah class yang perlu dibuat
(Jika Class Diagram tidak muncul di dokumen, buatlah)

#Selamat berdiskusi

#dan menyampaikan hasilnya saat jam praktikum berakhir

Modul 3 INTEGRATED DEVELOPMENT ENVIRONMENT

TUJUAN PRAKTIKUM
1. Mampu melakukan instalasi IDE. 2. Mampu mengoperasikan IDE.

3.1 Definisi

Integrated development environment (IDE) adalah sebuah perangkat lunak yang memberikan sekumpulan fasilitas kepada programmer dalam mengembangkan perangkat lunak. Pemilihan IDE yang tepat dapat meningkatkan efisiensi dan kualitas dari kode yang ditulis. Selain untuk menulis dan mengcompile code, IDE biasanya memiliki fitur-fitur yang umum diantaranya:

- Highlight syntax
- Code completion
- Refactoring support
- Debugging tool
- Code Search
- Dll.

Terdapat banyak IDE yang tersedia baik yang gratis maupun berbayar. Pemilihan IDE yang tepat haruslah didasari oleh kebutuhan dari tim pengembang terutama dari segi bahasa pemrograman yang akan digunakan, terlebih lagi jika tim akan menggunakan IDE berbayar. Beberapa IDE yang umum digunakan antara lain:

- Eclipse
- Netbeans
- Microsoft Visual Studio
- IntelliJ IDEA
- XCode
- Dll.

3.1.1 Eclipse

Contoh-contoh pada modul ini ditulis pada bahasa pemrograman Java menggunakan IDE Eclipse. Mahasiswa tidak diwajibkan menggunakan Java maupun Eclipse, tetapi jika ingin menggunakan IDE Eclipse dapat mengakses installer eclipse melalui <https://www.eclipse.org/downloads/>

KEGIATAN PRAKTIKUM

1. Setiap kelompok mendownload IDE untuk Implementasi TUBESnya.

Menginstallnya, dan :

- a). screenshot halaman awal IDE,
- b). cari fitur code completion
- c). cari fitur code search

2. Setelah membandingkan dengan 1 dari 79 Dokumen requirements internasional, lengkapi requirement di Dokumen SKPL Anda. Usahakan jumlah peran pengguna aplikasi Kelompok Anda setidaknya 3 (tiga) peran, dan use case dalam Use Case Diagram setidaknya 9 (sembilan). Lebih banyak lebih baik.

3. Lihat kembali Dokumen Perancangan Perangkat Lunak (DPPL) Kelompok Anda dengan desain tambahan akibat tambahan requirements tersebut, terutama pada :

- a. Use Case Diagram
- b. Use Case Description
- c. Class Diagram

#Laporkan hasilnya di akhir kegiatan Lab

#Selamat bekerja



Modul 4 GITHUB

TUJUAN PRAKTIKUM

1. Mampu mengoperasikan GitHub.

4.1 Definisi

GitHub adalah layanan hos web bersama untuk proyek pengembangan perangkat lunak yang menggunakan sistem kendali versi Git dan layanan hosting internet. Hal ini banyak digunakan untuk kode komputer. Ini memberikan kontrol akses dan beberapa fitur kolaborasi seperti pelacakan bug, permintaan fitur, manajemen tugas, dan wiki untuk setiap proyek.

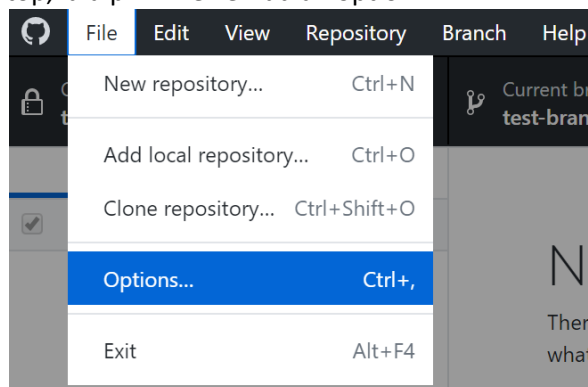
GitHub menawarkan paket repositori pribadi dan gratis pada akun yang sama dan digunakan untuk proyek perangkat lunak sumber terbuka. Pada bulan April 2017, GitHub melaporkan bahwa mereka mempunyai lebih dari 20 juta pengguna dan lebih dari 57 juta repositori, menjadikannya layanan terbesar dari kode sumber di dunia.

GitHub mempunyai sebuah maskot yang bernama Octocat, seekor kucing dengan lima tentakel dan wajah seperti manusia.

4.2 Instalasi

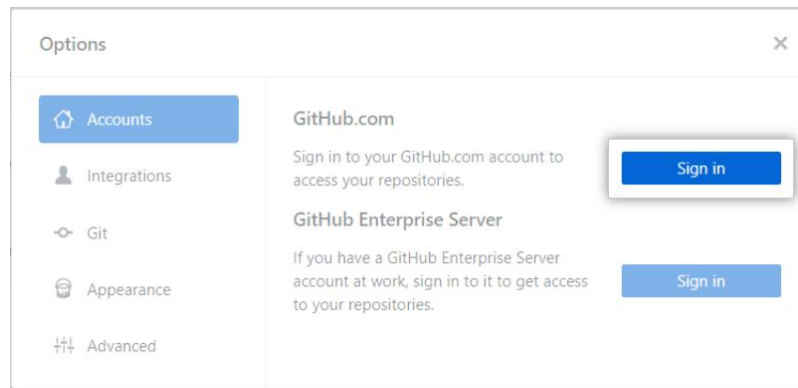
Berikut ini cara menginstall github:

1. Download git pada laman <https://git-scm.com/downloads>
2. Lakukan instalasi git dengan mengikuti default installation.
3. Download github desktop pada laman <https://desktop.github.com/>
4. Lakukan instalasi github desktop dengan membuka file installer yang tadi di download.
5. Buka Github desktop, lalu pilih file kemudian option.



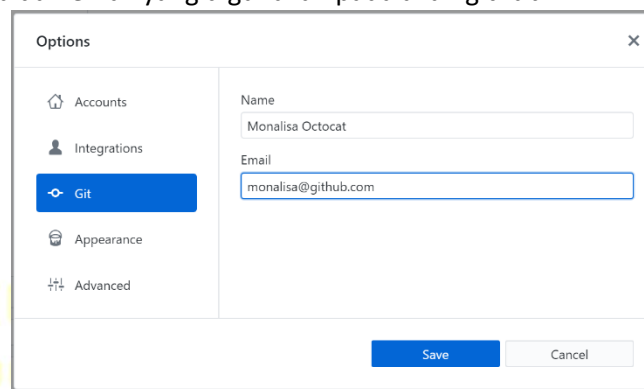
Gambar 4.1 Tampilan menu option pada Github desktop

6. Pada windows options pilih accounts.
7. Klik button sign in pada kanan "GitHub.com".



Gambar 4.2 Tampilan sign in akun Github

8. Lakukan Sign in akun github
9. Pada windows options pilih Git
10. Masukkan nama dan email yang digunakan pada akun github



Gambar 4.3 Tampilan input nama dan email pada konfigurasi Git

11. Klik button save.

4.3 Git Commands

Command	Fungsi
git init	membuat repository baru.
git clone	melakukan cloning repository berdasarkan url githubnya.
git add	menambahkan file ke dalam repository lokal
git commit	menyimpan perubahan pada repository lokal
git push	menyimpan commit pada online repository
git pull	mengecek dan menggabungkan update pada online repository dengan lokal repository
git fetch	mengecek update pada online repository

KEGIATAN PRAKTIKUM

1. Setiap kelompok membuat akun Github.

Dan memanfaatkannya untuk:

a). upload SKPL terakhir

b). upload DPPL terakhir

Dan melaporkan daftar akun Github kelompoknya

2. Dari arsitektur perangkat lunak yang Kelompok Anda ketahui,

(Dari slide kuliah - Pressman, dan

dari video : <https://www.youtube.com/watch?v=iyES7UwJfw>, dll)

lakukan analisis mana arsitektur perangkat lunak terbaik untuk perangkat lunak

Tugas Besar Kelompok Anda.

#Laporkan hasilnya di akhir kegiatan Lab

#Selamat bekerja



Modul 5 GUI BUILDER

TUJUAN PRAKTIKUM

1. Mampu mengimplementasikan desain antarmuka menggunakan GUI Builder.

5.1 Definisi

Graphical User Interface Builder (GUI builder) yang juga dikenal sebagai GUI designer adalah alat pengembangan perangkat lunak yang menyederhanakan pembuatan GUI dengan memungkinkan perancang untuk mengatur elemen kontrol grafis (widget) menggunakan drag-and-drop WYSIWYG editor. Tanpa adanya GUI Builder, GUI harus dibangun dengan secara manual menentukan parameter setiap widget dalam source-code, tanpa umpan balik visual sampai program dijalankan.

User interfaces biasanya diprogram menggunakan arsitektur event-driven architecture, jadi GUI builders juga menyederhanakan pembuatan kode berbasis event-driven. Kode pendukung ini menghubungkan widget dengan input dan output yang memicu fungsi yang menyediakan logika aplikasi.

Beberapa graphical user interface builders, seperti Glade Interface Designer, secara otomatis menghasilkan semua source-code untuk elemen kontrol grafis. Lainnya, seperti Interface Builder, menghasilkan instance serialized object yang kemudian dimuat oleh aplikasi.

Berikut ini beberapa contoh GUI Builders:

1. GTK+ / Glade Interface Designer
2. XForms (toolkit)
3. UWP / Windows Presentation Foundation / WinForms
4. Microsoft Visual Studio XAML Editor, XAML based GUI layout
5. SharpDevelop
6. Xamarin Studio
7. C++Builder / VCL (Visual Component Library)
8. Qt Creator / Qt
9. FLTK
10. wxWidgets
11. wxGlade
12. wxFormBuilder
13. wxCrafter (plugin for CodeLite)
14. Projucer
15. Android Studio, XML based GUI layout
16. NetBeans GUI design tool
17. Apache Cordova / PhoneGap

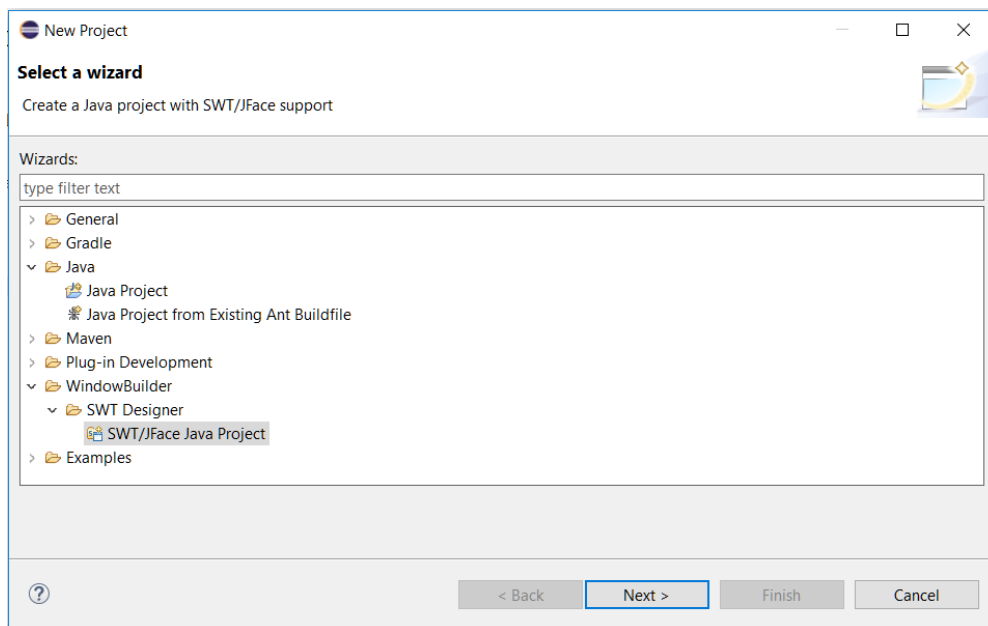
5.2 Implementasi WindowBuilder pada eclipse

Sebelum memulai, pastikan package WindowBuilder sudah terinstall pada eclipse anda. Kunjungi <https://www.eclipse.org/windowbuilder/download.php> untuk mendapatkan file instalasi terbaru.

5.2.1 Membuat Project Baru

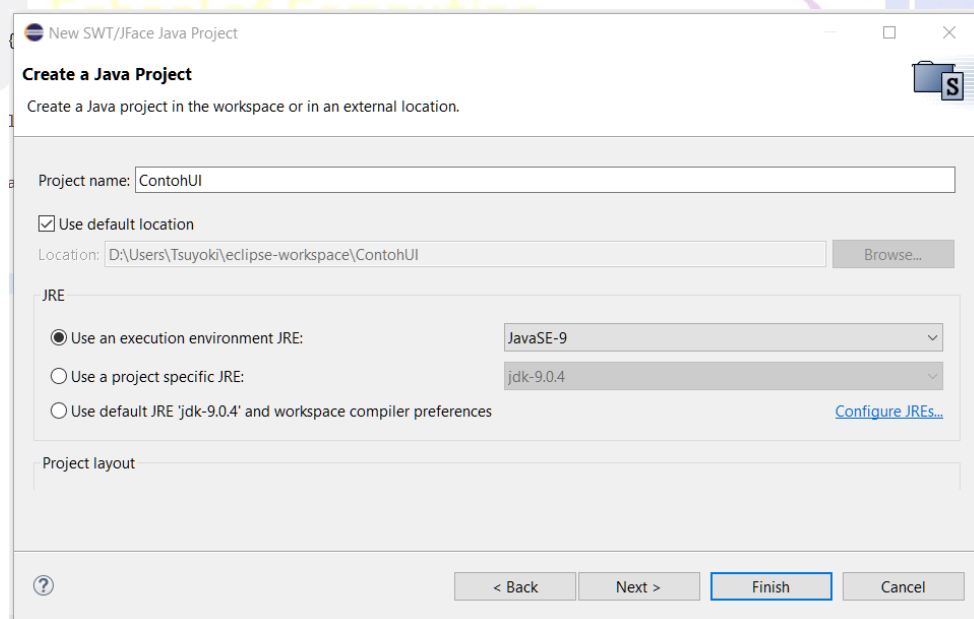
1. Buka Eclipse.

2. Pada menu File, pilih “New” lalu “Project...”
3. Pada window yang terbuka carilah “WindowBuilder” kemudian “SWT Designer” dan pilih “SWT/JFace Java Project” lalu klik “Next >”



Gambar 5.1 Tampilan membuat project baru pada Eclipse

4. Beri nama project sesuai dengan kebutuhan lalu klik “Finish”

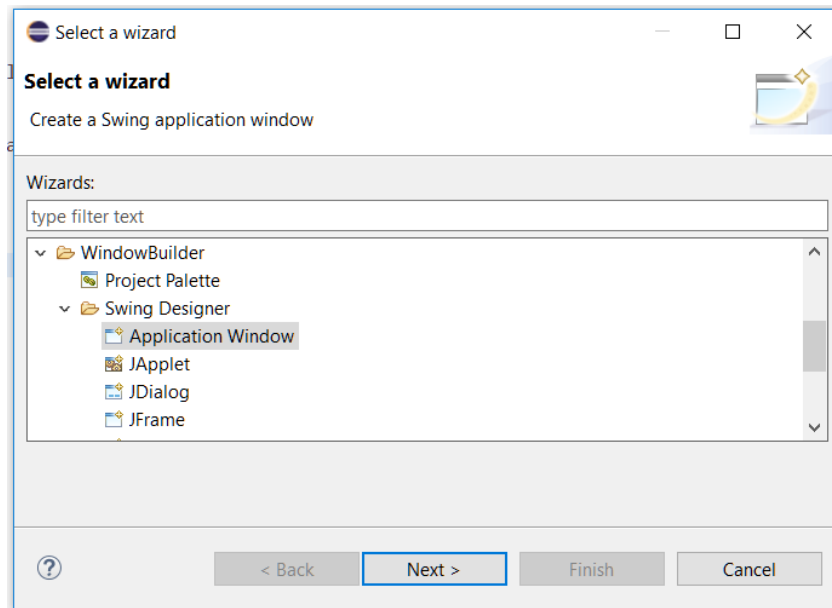


Gambar 5.2 Tampilan input nama project pada Eclipse

5.2.2 Menambahkan Windows

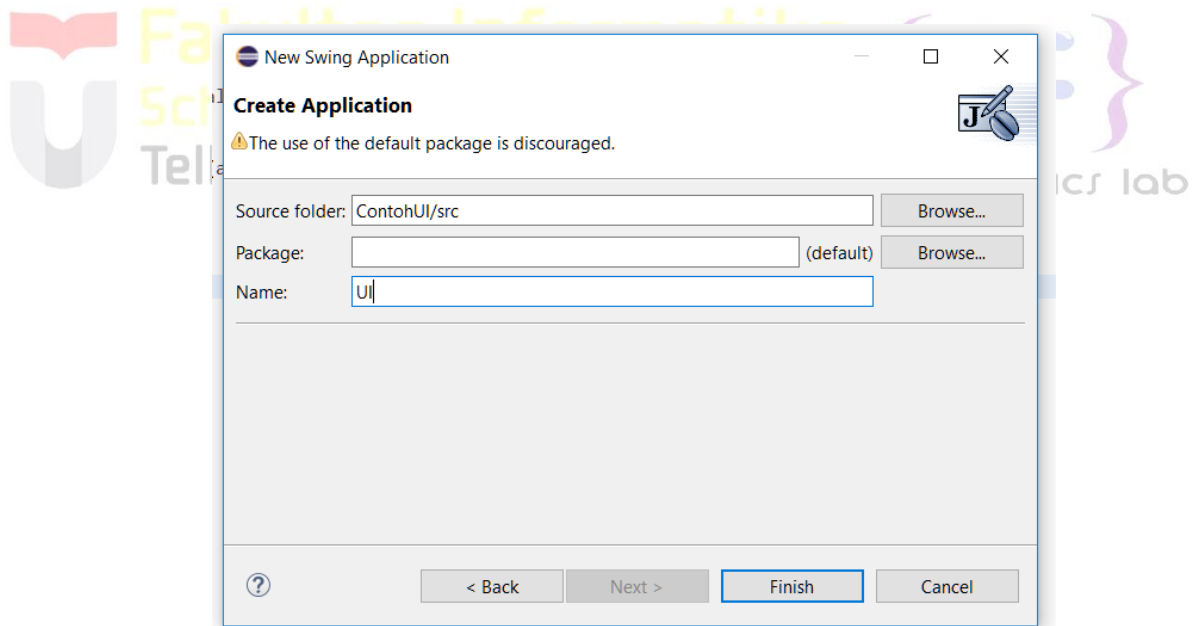
1. Klik kanan pada package dimana class windows akan disimpan.
2. Pilih “New” kemudian “Other...”

3. Cari “WindowBuilder” kemudian “Swing Designer” lalu pilih “Application Window”, klik “Next >”



Gambar 5.3 Tampilan window baru dengan WindowBuilder Eclipse

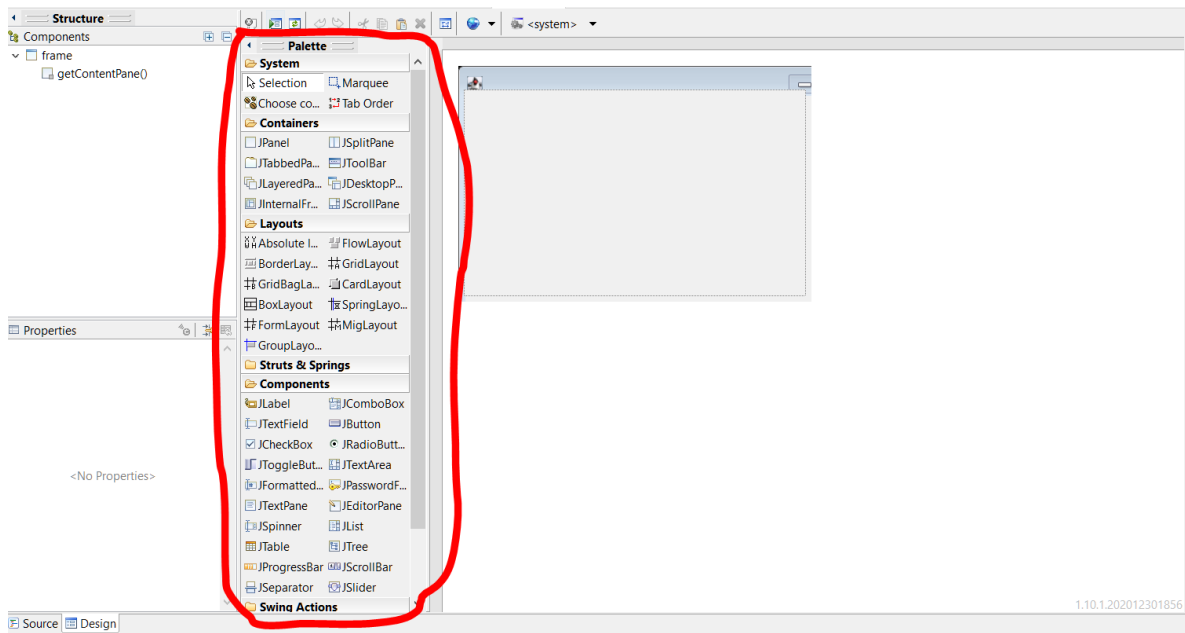
4. Beri nama sesuai kebutuhan lalu klik finish.



Gambar 5.4 Tampilan input nama window baru pada WindowBuilder Eclipse

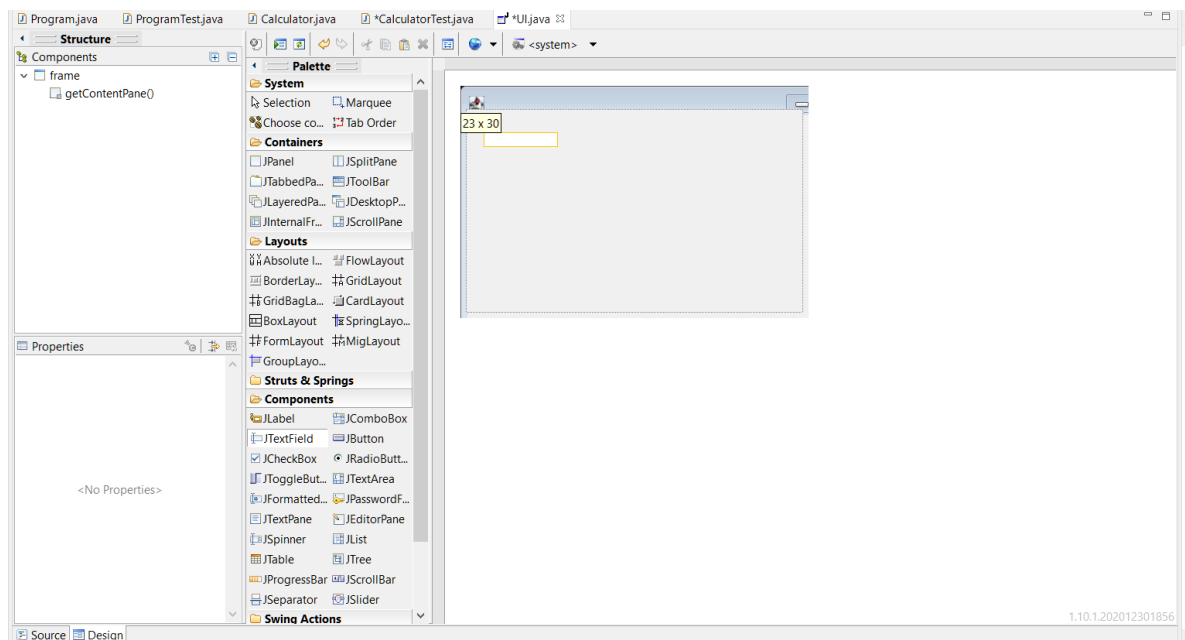
5.2.3 Menambahkan Komponen Pada Windows

1. Klik komponen yang diinginkan pada Palette



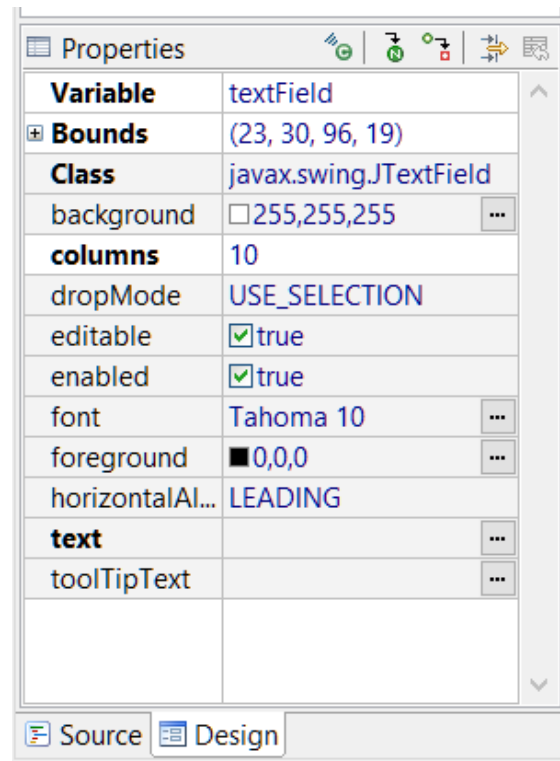
Gambar 5.5 Tampilan daftar komponen yang bisa ditambahkan ke window

2. Klik pada posisi yang diinginkan



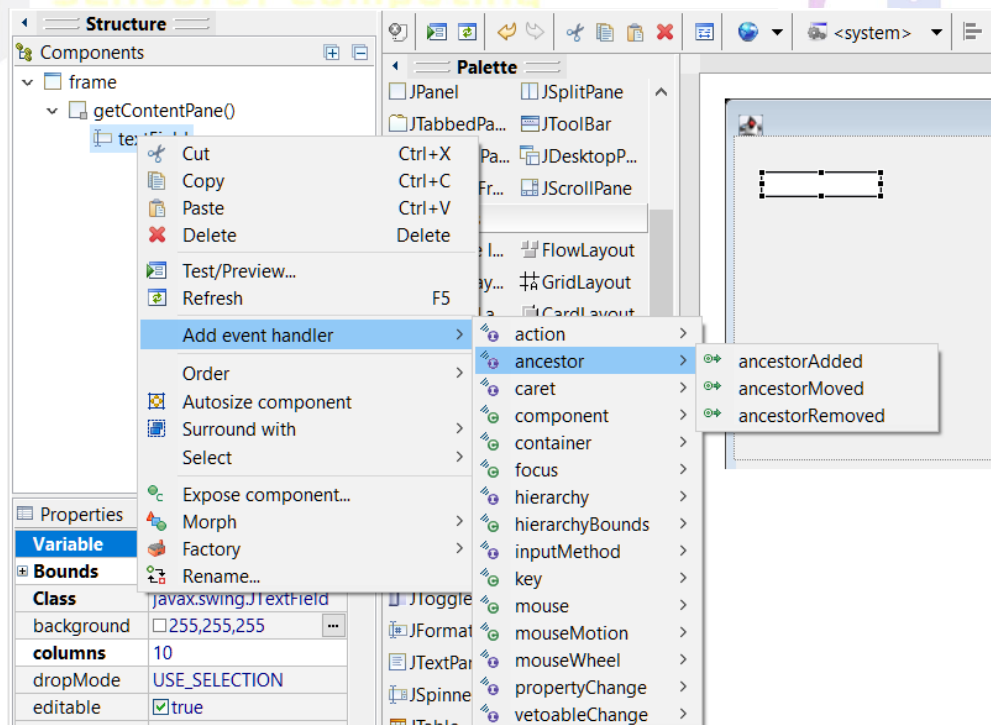
Gambar 5.6 Tampilan menambahkan komponen ke dalam window

3. Sesuaikan parameter sesuai yang dibutuhkan pada panel properties



Gambar 5.7 Tampilan menambahkan parameter komponen yang ditambahkan

4. Tambahkan event listener jika ada code yang akan dieksekusi untuk komponen tersebut untuk event tertentu.



Gambar 5.8 Tampilan menambahkan event listener pada komponen

KEGIATAN PRAKTIKUM

1. a. Jelaskan profile setiap kategori pengguna perangkat lunak Kelompok Anda.
b. Apa yang dilakukan masing-masing kategori pengguna di aplikasi Anda.
2. Buat implementasi semua tampilan *user interface* aplikasi Kelompok Anda:
 - *User Interface* untuk Input Data
 - *User Interface* untuk Output Informasimenggunakan IDE yang Anda pilih sebelumnya,
untuk masing-masing kategori pengguna tersebut.
3. Simpan di GitHub

#Laporkan hasilnya di akhir kegiatan Lab
#Selamat mengerjakan



Modul 6 MYSQL/MS ACCESS

TUJUAN PRAKTIKUM

1. Mampu mengimplementasikan desain basis data.

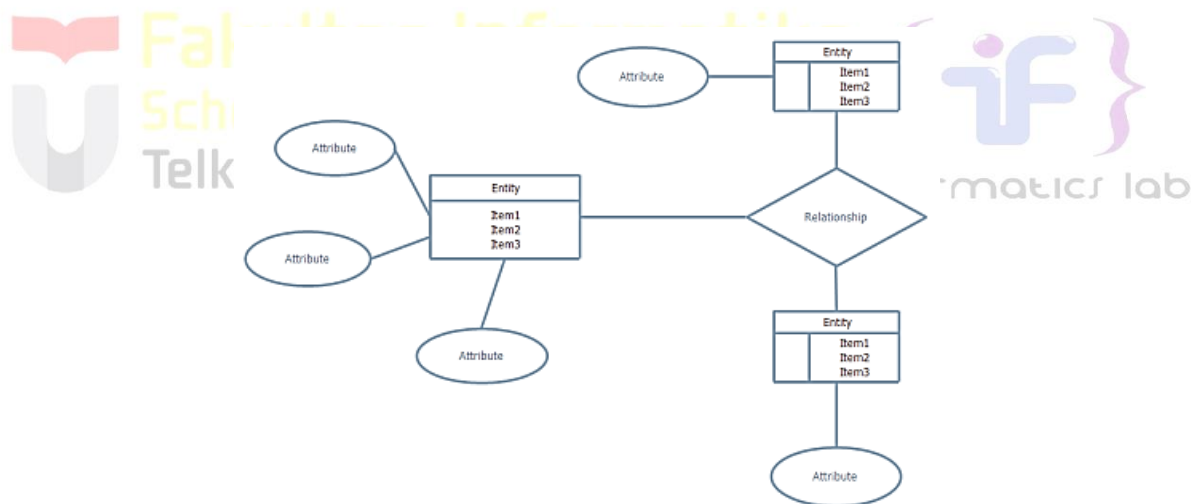
6.1 ER Diagram

ER atau Entity-relationship merupakan pemodelan data berdasarkan persepsi pada dunia nyata yang terdiri dari objek yang disebut entitas, dan hubungan antar entitas yang biasa disebut relasi. Sebagai contoh, mahasiswa, dosen dan mata kuliah merupakan entitas yang diambil dari lingkungan kampus.

Entitas dideskripsikan pada database dengan kumpulan atribut. Misalnya, NIM, Nama, Tempat dan Tanggal Lahir. Dari atribut diatas kita dapat menyimpulkan bahwa atribut diatas merupakan atribut milik entitas mahasiswa.

Pada entitas yang ada diperlukan atribut yang unik, yang dapat menunjuk suatu individu secara jelas, yang nantinya digunakan sebagai primary key. Seperti pada kasus mahasiswa diatas, yang menjadi primary key adalah NIM, dikarenakan NIM setiap mahasiswa pasti berbeda-beda. Sedangkan atribut lain seperti nama, tempat tanggal lahir terdapat kemungkinan kesamaan antara satu mahasiswa dengan mahasiswa yang lain.

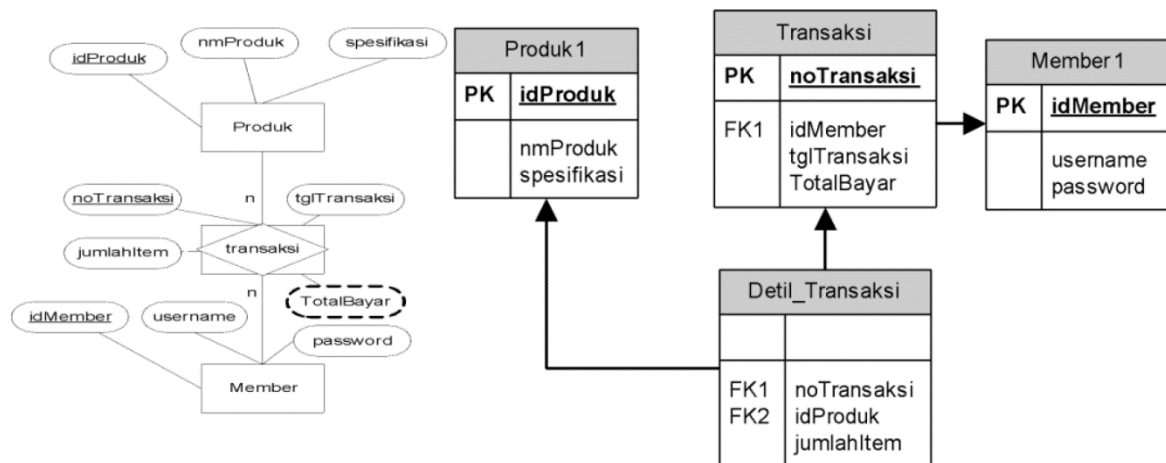
Blok bangunan dasar dari ER Diagram adalah entitas, atribut dan relasi. Selain itu, ada pula agregasi dan generalisasi-spesialisasi sebagai blok bangunan tambahan pada ERD.



Gambar 6.1 Contoh gambar ERD Chen

6.2 Primary Key

Atribut kunci dari suatu entitas kuat yang unik (seperti nim, id) sehingga membedakan dengan lainnya. 6.4 Foreign Key Attribute (atau satu set attribute) yang melengkapi satu hubungan yang menunjukkan ke induknya. Foreign key berguna untuk mendefinisikan kolom-kolom pada suatu tabel yang nilainya mengacu ke tabel lain, jadi kolom foreign key nilainya harus diambil dari nilai kolom pada tabel lain.



Gambar 6.2 Contoh gambar ERD yang melibatkan Primary Key dan Foreign Key

6.3 Normalisasi I, II dan III

Sebuah tabel relasional dikatakan memenuhi bentuk normal tertentu jika memenuhi satu set kondisi tertentu. Derajat normalisasi ditentukan menggunakan normal form. Secara umum ada 6 bentuk normal, namun yang akan dibahas pada bahasan ini hanya 4 dari 6, yaitu normal bentuk 1 (1 NF), normal bentuk 2 (2 NF), normal bentuk 3 (3 NF), dan terakhir adalah BCNF (Boyce-Codd Normal Form). Setiap normal form merupakan suatu set kondisi pada skema yang menjamin sifat tertentu yang berkaitan dengan redundansi dan memperbaiki anomali.

Normal Bentuk Pertama (1NF)

- Jika dan hanya jika semua kolomnya atomik (tidak ada kelompok berulang atau composit)
- Setiap record harus unik (tidak ada pengulangan)
- Tidak ada atribut yang muncul berulang atau atribut bernilai ganda
- Tiap atribut hanya memiliki satu pengertian.

Normal Bentuk Kedua (2NF)

- Telah memenuhi bentuk normal pertama
- Atribut bukan kunci harus bergantung penuh secara fungsional pada candidate key (CK)

Normal Bentuk Ketiga (3NF)

- Telah memenuhi bentuk normal kedua
- Semua atribut bukan himpunan bagian CK tidak mempunyai hubungan yang transitif antar kolom dengan CK.

Contoh:

CK = AB

AB → C

AB → D

$CD \rightarrow A$

hal ini tidak membuat penggalan terhadap 3NF, karena A merupakan himp bagian dari CK

$CD \rightarrow E$

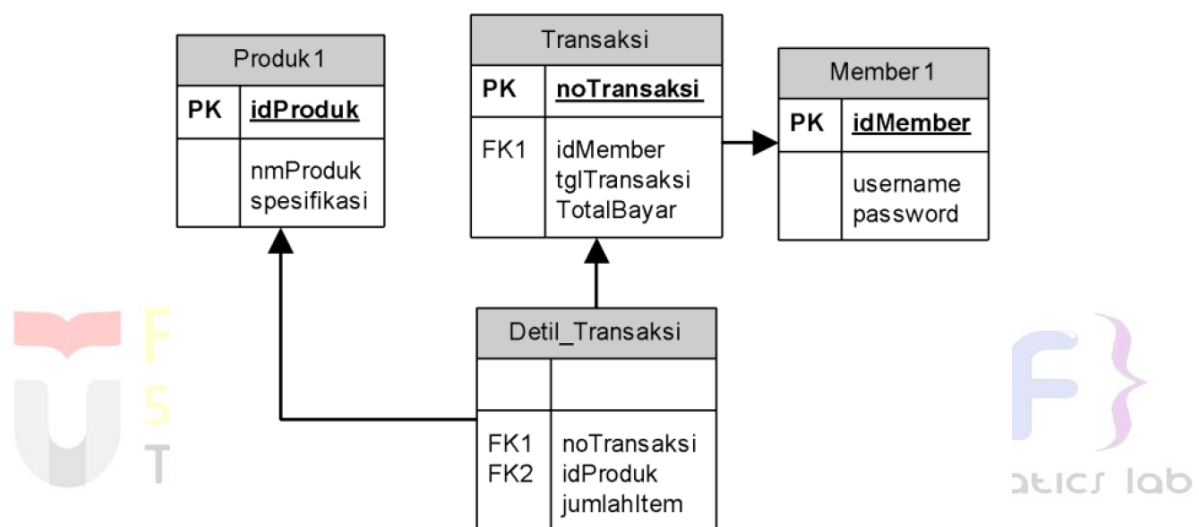
hal ini membuat jadi tidak 3NF, karena E bukan merupakan himp bagian dari CK

BCNF

Telah memenuhi bentuk normal ketiga

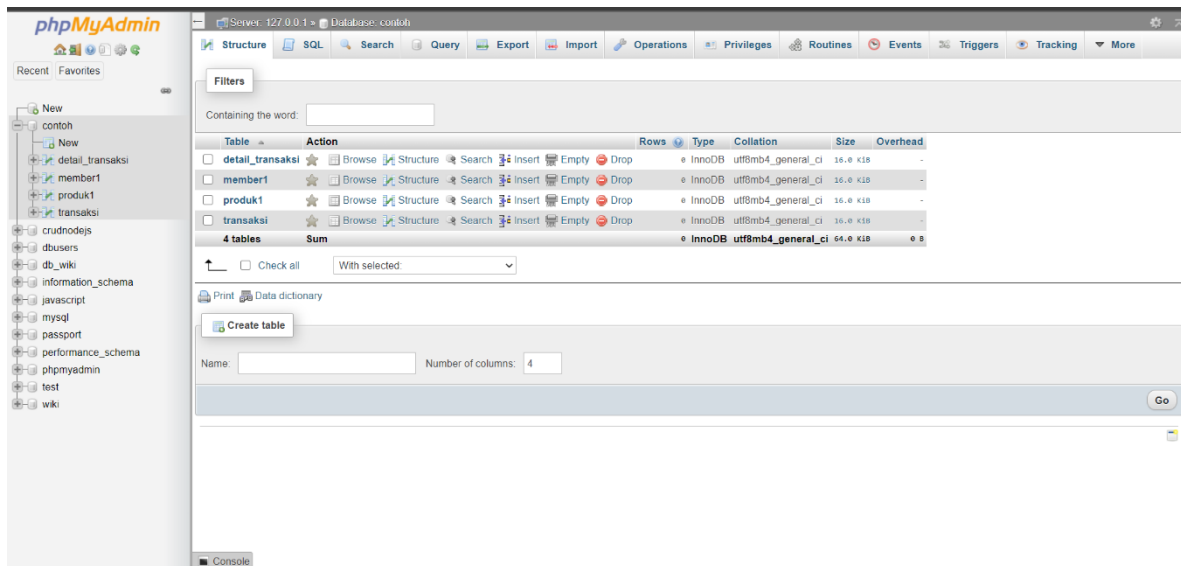
Setiap determinan harus menjadi candidate key (CK). Dengan kata lain, setiap ketergantungan fungsional yang terbentuk, ruas kirinya merupakan CK.

6.4 Implementasi Basis Data pada MySQL



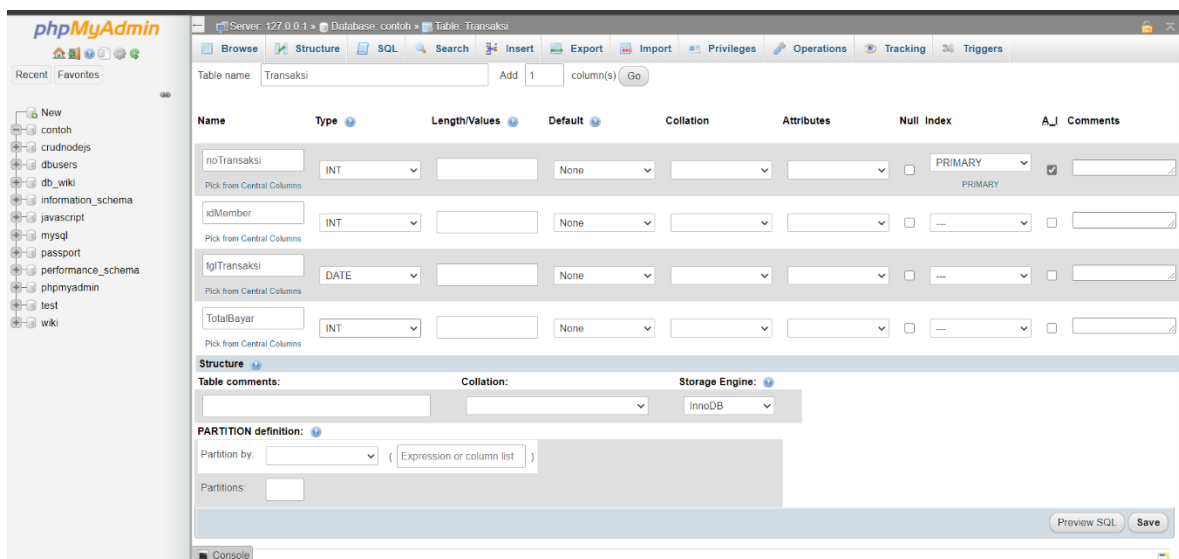
Gambar 6.3 Contoh ERD untuk implementasi pada MySQL

Karena terdapat empat entitas pada ERD, maka kita harus menerapkan empat table pada database yang kita bangun. Kemudian untuk tiap table kita tambahkan attribute sesuai dengan yang tertera pada ER Diagram. Terakhir, pastikan primary key dari tiap table sudah ditandai. Seperti berikut:

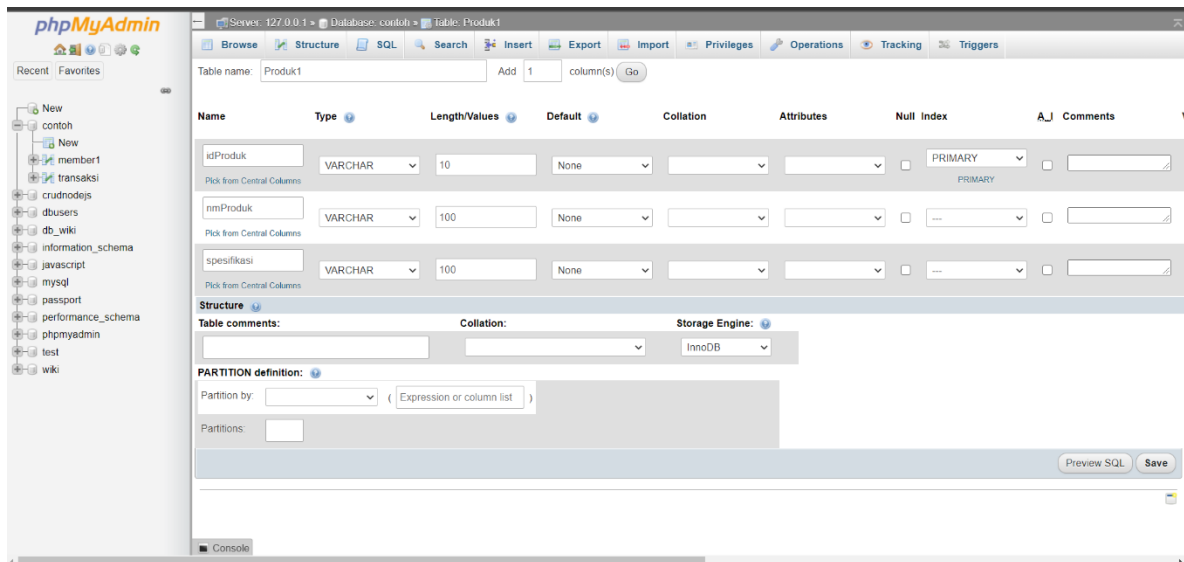


Gambar 6.4 Tabel yang dibuat pada MySQL

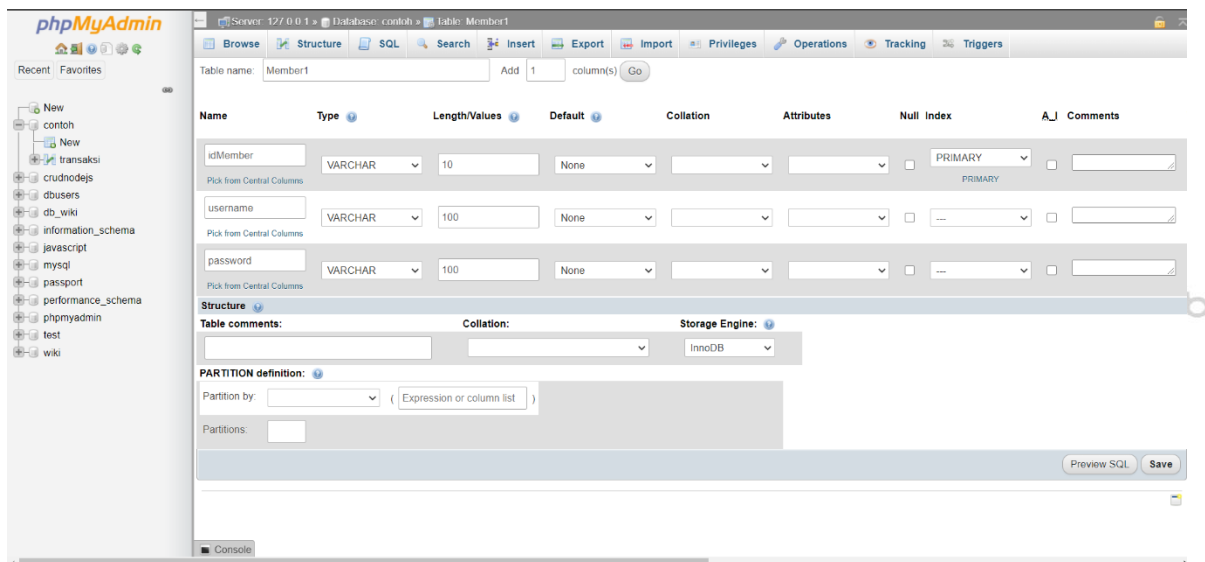
Detail dari tiap table dapat dilihat sebagai berikut:



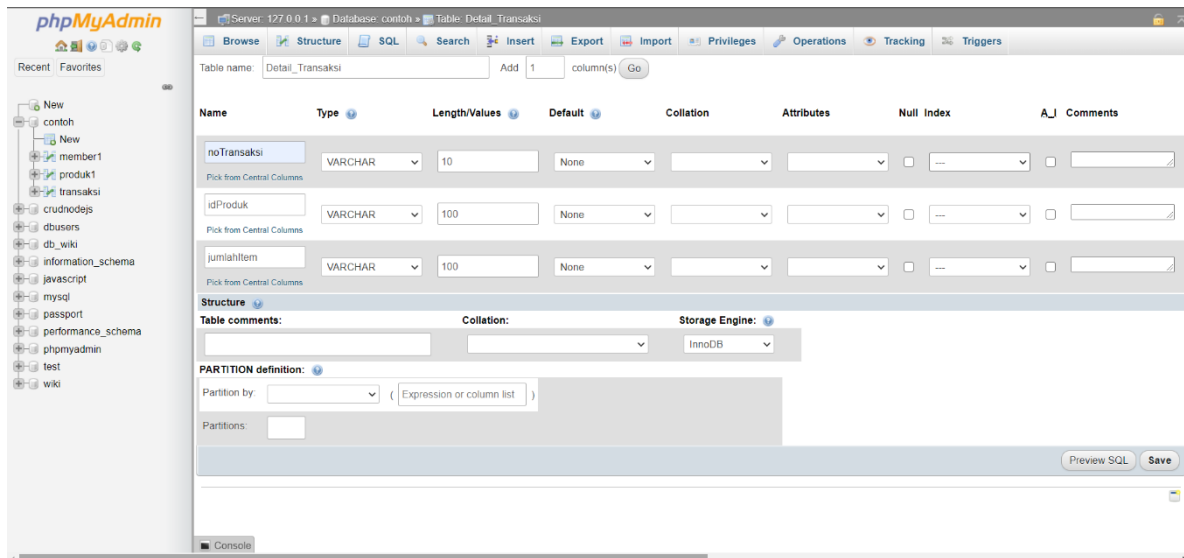
Gambar 6.5 Detail tabel transaksi



Gambar 6.6 Detail tabel produk



Gambar 6.7 Detail tabel member



Gambar 6.8 Detail tabel detail transaksi

TUGAS PENDAHULUAN

(dikumpulkan sebelum masuk Kelas Praktikum)

Dari Class Diagram atau Entity Relationship Diagram dalam dokumen DPPL Anda:
lakukan desain tabel-tabel database

(lengkap dengan kolom, tipe data, ukuran, primary key/foreign key)

KEGIATAN PRAKTIKUM

(Dikerjakan pada saat melaksanakan Praktikum)

1. a. Pilih software DBMS untuk implementasi database TUBES. Jelaskan alasan pemilihan.
b. Download DBMS pilihan kelompok Anda, dan install.

2. Implementasikan tabel-tabel database hasil Tugas Pendahuluan dalam DBMS yang sudah Anda install

#Selamat bekerja

Modul 7 IMPLEMENTASI CLASS DIAGRAM

TUJUAN PRAKTIKUM

1. Mampu mengimplementasikan desain class.

7.1 Implementasi Class Tunggal

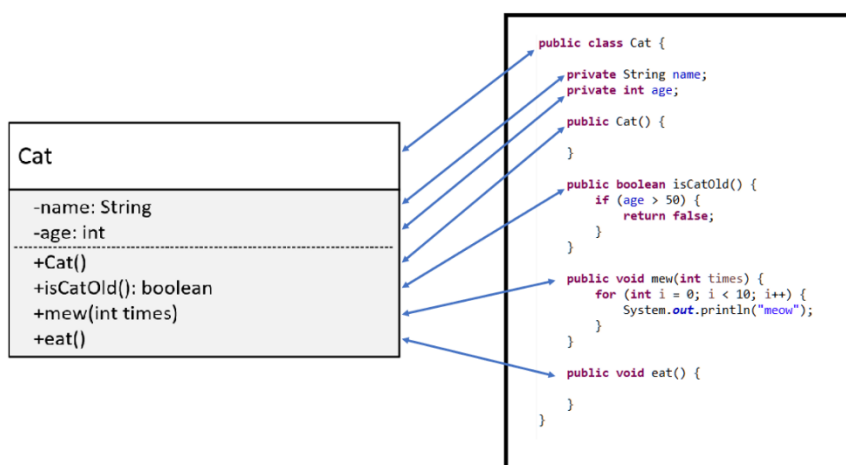
Sebuah class terdiri dari tiga bagian, yaitu:

1. Nama kelas, ditulis dengan nama benda tunggal. Sebuah kelas dapat ditulis dengan hanya menuliskan nama dari class seperti "Cat" atau diberi prefix dari *package* dimana kelas tersebut berada seperti "Animal Package::Cat".
2. Attribute, merupakan nilai yang dapat dimiliki oleh kelas tersebut.
3. Operasi, merupakan implementasi dari layanan yang dapat diminta dari beberapa *object* dan *class* yang mempengaruhi behavior.

Attribute dan operasi memiliki visibility yang menentukan apakah sebuah attribute atau operasi dapat diakses oleh kelas lain atau tidak. Terdapat tiga buah tingkat visibility yang umumnya dapat digunakan seperti pada tabel

Visibility	Keterangan
Public (+)	Dapat diakses oleh <i>class</i> lain.
Protected (#)	Hanya dapat diakses oleh class itu sendiri dan class turunannya.
Private (-)	Hanya dapat diakses oleh class itu sendiri

Ketika sebuah class diagram diimplementasikan dalam kode setiap nama, operasi dan attribute beserta visibility nya harus diterapkan sesuai dengan nama yang tercantum. Setiap attribute harus bertipe sesuai dengan apa yang tercantum pada class diagram. Begitu pula dengan operasi, harus menerima parameter input dan mengembalikan nilai sesuai yang tercantum pada class diagram.



Gambar 7.1 Contoh class diagram dan implementasinya

7.2 Implementasi Relasi Antar Class

Relasi atau relationship menghubungkan beberapa objek sehingga memungkinkan terjadinya interaksi dan kolaborasi diantara objek-objek yang terhubung. Relasi antar class antara lain:

7.2.1 Association

Relasi asosiasi merupakan relasi structural yang menspesifikasikan bahwa satu objek terhubung dengan objek lainnya. Dalam penerapannya, association dilakukan dengan menyimpan pointer objek dari salah satu kelas di objek dari kelas lainnya. Contoh dari penerapan association sebagai berikut:



Gambar 7.2 Contoh class diagram dengan association

```
public class Department {
    String name;

    public Department(String name) {
        this.name = name;
    }
}
```

Gambar 7.3 Implementasi kelas Department

```
public class Employee {
    Department department;
    String name;
    // Other attributes

    public Employee(String name) {
        this.name = name;
    }

    public void SetDepartment(Department newDepartment) {
        department = newDepartment;
    }
}
```

Gambar 7.4 Implementasi kelas Employee


```

public class MainClass {

    public static void main(String[] args) {

        Department departmentPerbelanjaan = new Department("Perbelanjaan");
        Employee employeeUjang = new Employee("Ujang");

        employeeUjang.SetDepartment(departmentPerbelanjaan);

    }

}

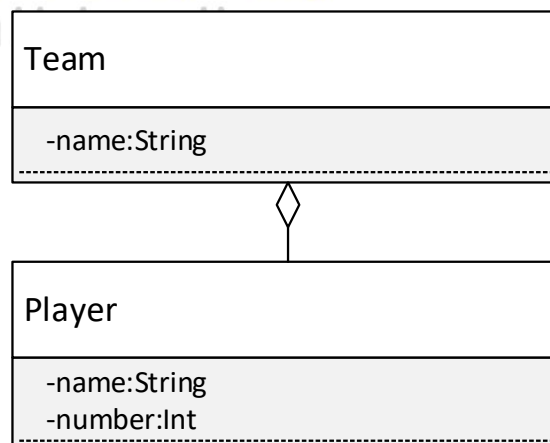
```

Gambar 7.5 Implementasi kelas penghubung Department-Employee

7.2.2 Aggregation

Aggregation relationship adalah sebuah hubungan yang direpresentasikan dengan frase “*Whole-part*”. Bagian *class* yang lebih besar disebut dengan “*Whole*” yang mengandung bagian *class* yang lebih kecil yakni “*part*”. Secara umum, implementasi dari aggregation tidak berbeda jauh dengan association terutama pada bahasa pemrograman Java, C#, dan sejenisnya. Perbedaan utama aggregation ada pada kepastian bahwa objek dari class “*Whole*” menyimpan pointer untuk objek-objek dari class “*part*”.

Contoh penerapan aggregation untuk sebuah class “*team*” yang memiliki beberapa “*player*” adalah sebagai berikut:



Gambar 7.6 Contoh class diagram dengan aggregation

```

import java.util.LinkedList;

public class Team {

    String name;
    LinkedList<Player> players;

    public Team(String name) {
        this.name = name;
        players = new LinkedList<Player>();
    }

    public void AddPlayer(Player newPlayer) {
        players.add(newPlayer);
    }

}

```

Gambar 7.7 Implementasi kelas Team

```

public class Player {

    String name;
    int number;

    public Player(String playerName, int playerNumber) {
        name = playerName;
        number = playerNumber;
    }

}

```

Gambar 7.8 Implementasi kelas Player

```

public class MainClass {

    public static void main(String[] args) {

        Team team = new Team("FIF FC");
        team.AddPlayer(new Player("Sultan", 10));
        team.AddPlayer(new Player("Mamang", 8));

    }

}

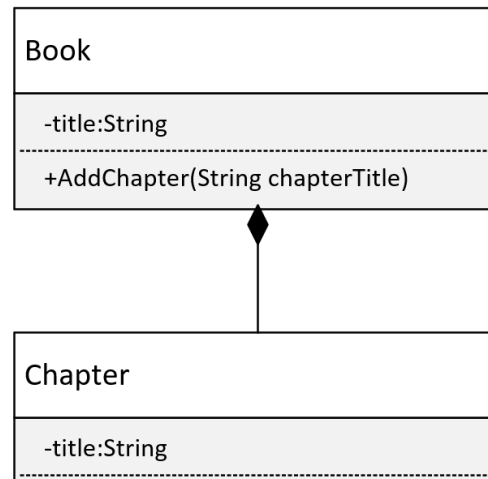
```

Gambar 7.9 Implementasi kelas penghubung Team-Player

7.2.3 Composition

Composition relationship adalah sebuah hubungan agregasi dimana “*Whole*” mengendalikan *creation* dan *destruction* dari “*part*”. Singkatnya “*part*” tidak akan ada tanpa “*Whole*”, dan jika “*Whole*” dihapus maka “*part*” akan ikut hilang juga. Pada implementasinya, objek dari class “*Whole*” mengendalikan sepenuhnya pembentukan dan penghapusan objek-objek yang menjadi partnya.

Contoh penerapan composition sebagai berikut:



Gambar 7.10 Contoh class diagram dengan composition

```
import java.util.LinkedList;

public class Book {

    String title;
    private LinkedList<Chapter> chapters;

    public Book(String title) {
        this.title = title;
        chapters = new LinkedList<Chapter>();
    }

    public void AddNewChapter(String chapterTitle){
        Chapter newChapter = new Chapter(chapterTitle);
        chapters.add(newChapter);
    }

}
```

Gambar 7.11 Implementasi kelas Book

```

public class Chapter {

    String title;

    public Chapter(String title) {
        this.title = title;
    }

}

```

Gambar 7.12 Implementasi kelas Chapter

TUGAS PENDAHULUAN

(dikumpulkan sebelum masuk Lab)

Banyak aplikasi web saat ini yang memanfaatkan API dari web lain. Misalnya:

- ketika mengisi form registrasi, memanfaatkan API dari Google untuk mengisi form otomatis dengan data-data pengguna yang ada di Google
- mengisi kelengkapan data alamat pengguna, dengan data IP Address (atau lokasi GPS android)
 - a. Cari info bagaimana mengakses API Google tersebut
 - b. Bagaimana memanfaatkan data IP Address (atau GPS Android) untuk melengkapi data alamat

KEGIATAN PRAKTIKUM

1. a. Pilih bahasa pemrograman OO untuk implementasi program TUBES. Jelaskan alasan pemilihan Anda.
b. Download compiler bahasa pemrograman tsb, dan install.
2. Implementasikan setidaknya tiga class dari Class Diagram yang ada di DPPL Anda dengan bahasa pemrograman yang telah Anda install :
 - a. Class paling sederhana (paling sedikit atribut & methodnya)
 - b. Class yang sedang (jumlah atribut & methodnya "rata-rata")
 - c. Class yang paling kompleks (paling banyak atribut & methodnya, atau mengandung kode untuk pengksesan API soal Tugas Pendahuluan, atau yang mengandung aspek "kecerdasan buatan" (jika ada))

#Selamat bekerja

Modul 8 IMPLEMENTASI SEQUENCE DIAGRAM

TUJUAN PRAKTIKUM

- | |
|--|
| 1. Mampu mengimplementasikan sequence diagram. |
|--|

8.1 Sequence Diagram

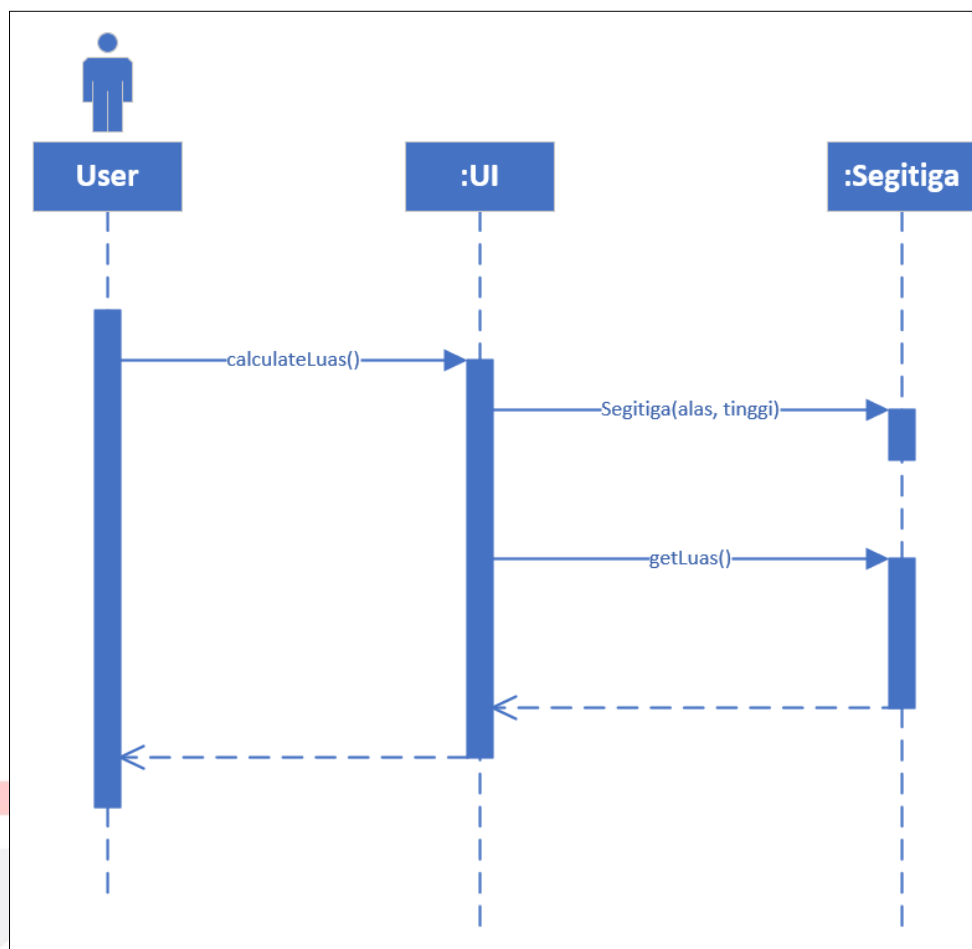
Sequence diagram menggambarkan interaksi antara aktor dan sistem dalam suatu skenario *use case*, menggambarkan interaksi seluruh objek yang terkait dengan skenario. *Sequence diagram* memodelkan *logic* dari *use case*, menunjukkan sejumlah *object* dan pesan-pesan yang dilewatkan antara *object-object* ini dalam skenario tersebut dalam sebuah urutan waktu. *Sequence diagram* digunakan sebagai alat komunikasi dengan *programmer* untuk menjelaskan runtutan *method (behavior)* yang digunakan dalam mengimplementasikan *use case*.

8.2 Implementasi Sequence Diagram

Untuk menerapkan sebuah sequence diagram yang sudah dibuat dapat mengikuti langkah-langkah sebagai berikut:

1. Implementasikan class diagram yang sudah dibangun beserta dengan objeknya.
2. Mulai dari objek yang berinteraksi dengan actor, buatlah operasi yang dapat dipanggil oleh actor sesuai dengan message yang tercantum pada sequence diagram.
3. Dalam operasi yang dibuat, panggil operasi dari objek lain sesuai dengan message atau operasi lain dari objek itu sendiri jika ada self-call.
4. Ulangi nomor 3 hingga seluruh operasi yang tertulis pada message dan self-call di sequence diagram terpanggil.
5. Apabila operasi yang dipanggil pada nomor 3 dan 4 memiliki *return message*, tampung return message tersebut untuk diolah sesuai dengan algoritma yang ada pada operasi tersebut.

8.3 Contoh Implementasi Sequence Diagram



Gambar 8.1 Contoh Sequence Diagram.

Sequence diagram di atas menggambarkan use case sederhana untuk menghitung luas segitiga. Terdapat actor user yang akan berinteraksi dengan objek dari class UI dan class Segitiga yang bertanggung jawab untuk melakukan perhitungan.

```

import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JTextField;
import javax.swing.JLabel;
import java.awt.Font;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class UI {

    private JFrame frmKalkulatorSegitiga;
    private JTextField fieldAlas;
    private JTextField fieldTinggi;
    private JTextField fieldHasil;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {}

    /**
     * Create the application.
     */
    public UI() {}

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {}

    private void calculateLuas() {
        float alas = Float.parseFloat(fieldAlas.getText());
        float tinggi = Float.parseFloat(fieldTinggi.getText());
        Segitiga segitiga = new Segitiga(alas, tinggi);
        float luas = segitiga.getLuas();
        fieldHasil.setText(String.valueOf(luas));
    }
}

```

Gambar 8.2 Class UI.

Class pertama yang diimplementasi adalah kelas UI yang bertindak sebagai interface antara actor user dan sistem dibelakangnya. Di kelas UI tersebut kita mengimplementasikan operasi calculateLuas() yang akan dipanggil ketika user ingin melakukan perhitungan. Operasi calculateLuas() akan membuat sebuah objek segitiga yang akan digunakan untuk menampung informasi luas dan tinggi untuk menghasilkan perhitungan. Operasi calculateLuas() akan memanggil dua operasi dari objek Segitiga, yang pertama adalah operasi Segitiga(alas,tinggi) untuk membuat objek dan menginisiasi nilai alas dan tinggi dan yang kedua getLuas() untuk mendapatkan nilai luas segitiga.

```

public class Segitiga {

    private float alas;
    private float tinggi;

    public Segitiga(float alas, float tinggi) {
        this.alas = alas;
        this.tinggi = tinggi;
    }

    public float getLuas() {
        return alas * tinggi / 2;
    }

}

```

Gambar 8.3 Class Segitiga.

Class berikutnya yang diimplementasi adalah class Segitiga yang akan menerima nilai alas dan tinggi dan melakukan perhitungan luas. Alas dan tinggi sesuai dengan sequence diagram dikirimkan melalui operasi constructor dari class Segitiga. Kemudian operasi getLuas() juga diterapkan yang mengembalikan nilai float sesuai dengan yang sudah didesain.

TUGAS PENDAHULUAN

Membuat SEQUENCE DIAGRAM

1. Periksa kembali apakah di DPPL Kelompok Anda, terdapat Sequence Diagram untuk setiap USE CASE yang ada di USE CASE DIAGRAM.
(Misal ada 5 use case, maka perlu dibuat 5 Sequence Diagram)
2. Jika belum ada buatlah SEQUENCE DIAGRAM di DPPL Anda
3. Jika sudah ada, periksa kembali kelengkapan dan keakuratannya.
Laporkan dokumen DPPL yang sudah lengkap Sequence Diagramnya sebelum masuk Lab.

KEGIATAN PRAKTIKUM

Sequence Diagram pada umumnya menggambarkan interaksi Actor dengan sistem dalam satu rangkaian masukan dan keluaran.

Masukan dari Actor via IMK, akan direspon oleh aplikasi melalui pemanggilan method-method dari class-class yang saling berinteraksi, yang jika diperlukan sampai perlu mengakses tabel basis data yang relevan dengan SQL yang di-"embed" di bahasa pemrograman, lalu memberikan hasil pengaksesannya ke Actor via IMK lagi.

Implementasikan setidaknya satu Sequence Diagram yang dapat bekerja sesuai deskripsi di atas.

#Selamat bekerja

#Laporkan hasilnya di akhir Praktikum

Modul 9 IMPLEMENTASI ROBUSTNESS ANALYSIS

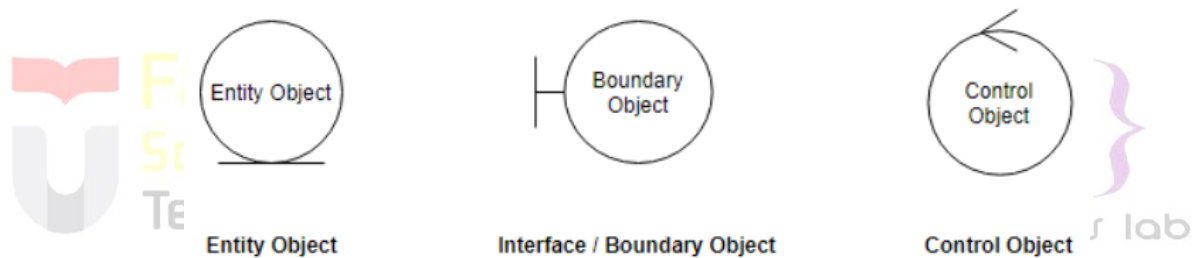
TUJUAN PRAKTIKUM

1. Mampu mengimplementasikan robustness analysis.

9.1 Robustness analysis

Robustness analysis adalah suatu pendekatan untuk memastikan bahwa desain dari use case diagram, class diagram, dan sequence diagram sudah berkaitan satu sama lain. Robustness analysis memastikan bahwa “what” yang tergambar pada analisis dan “how” yang tergambar pada desain terkait satu sama lain. Pada robustness analysis model terdapat tiga kategori objek:

1. Entity object: Objek yang muncul dari model domain, biasanya objek yang terlihat dengan jelas dan nyata ketika melakukan analisis. Umumnya digambarkan dengan kata benda.
2. Boundary object: Objek yang bertugas untuk menjembatani komunikasi antar actor dan sistem.
3. Control object: Objek yang bertugas menjembatani boundary dengan entity, umumnya berisi fungsi-fungsi yang akan dieksekusi. Umumnya digambarkan dengan kata kerja.



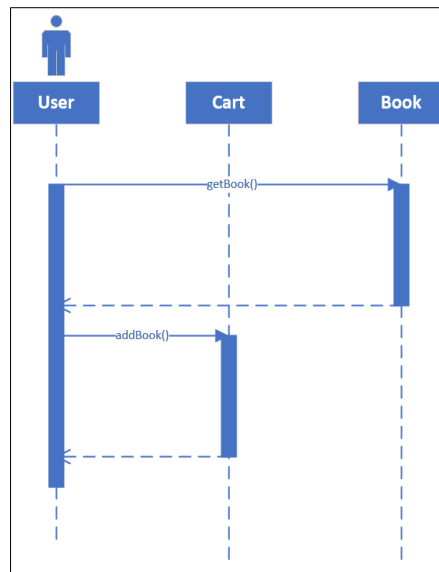
Gambar 9.1 Kategori objek pada robustness analysis.

Dalam implementasinya relasi antar ketiga kategori objek itu dibatasi dengan aturan-aturan sebagai berikut:

1. Sebuah objek entity hanya boleh berhubungan dengan objek control.
2. Sebuah objek boundary hanya boleh berhubungan dengan objek control.
3. Sebuah objek control dapat berhubungan dengan objek entity dan boundary maupun dengan objek control lainnya.
4. Sebuah actor hanya boleh berhubungan dengan objek boundary.

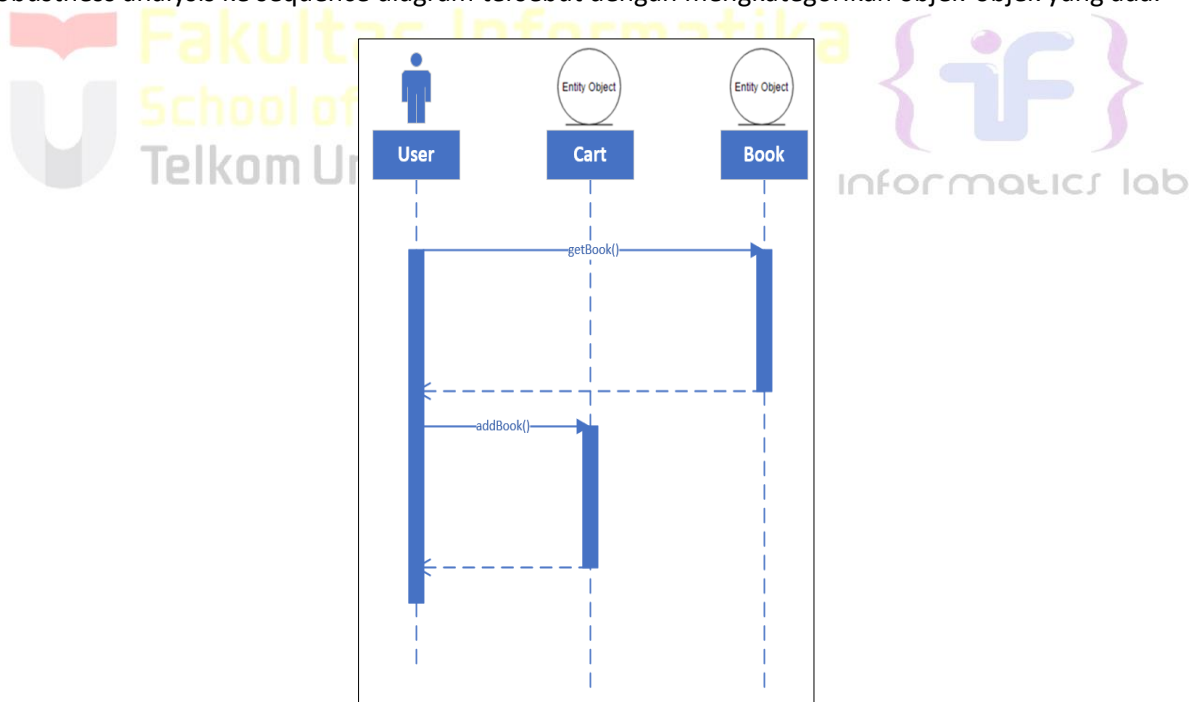
9.2 Implementasi Robustness analysis

Implementasi robustness analysis dapat dilakukan dengan melakukan pengembangan dari sequence diagram yang sudah dibuat. Berikut tahapan-tahapan dari penerapan robustness analysis dan contohnya:



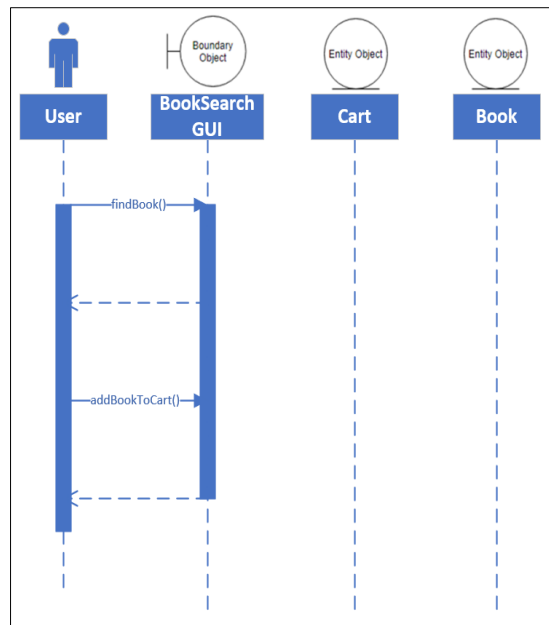
Gambar 9.2 Sequence diagram add book to cart.

Sequence diagram di atas berusaha untuk menggambarkan interaksi untuk menyelesaikan use case “add book to cart”. Tetapi pada sequence tersebut masih terdapat masalah, diantaranya adalah tidak adanya informasi mengenai objek yang menjadi antarmuka bagi user dan bagaimana objek dari Cart dan Book saling berinteraksi. Agar masalah tersebut dapat terlihat, kita dapat mulai menerapkan robustness analysis ke sequence diagram tersebut dengan mengkategorikan objek-objek yang ada.



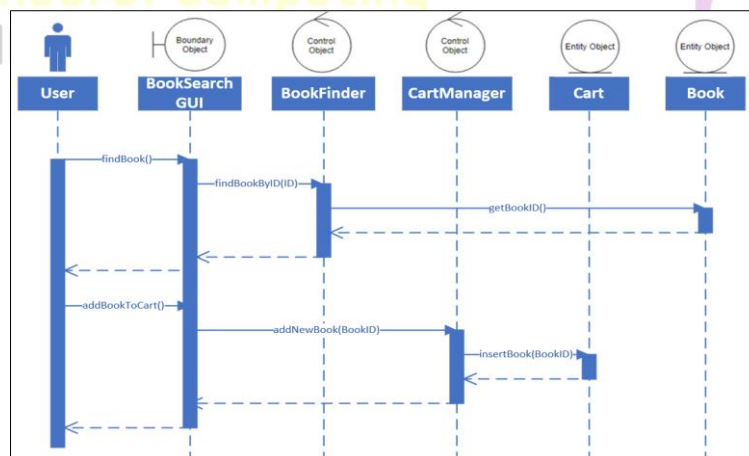
Gambar 9.3 Sequence diagram (Entity Object).

Setelah dikategorikan, dapat dilihat bahwa baik objek dari kelas cart maupun objek dari kelas book adalah entity object. Sesuai dengan aturan relasi antar kategori, user seharusnya hanya dapat mengakses boundary object. Oleh karena itu, kita dapat memulai dengan menambahkan boundary object yang dapat diakses oleh user.



Gambar 9.4 Menambahkan GUI ke Sequence Diagram.

Disini kita menambahkan sebuah GUI yang kita beri nama BookSearchGUI untuk menjadi boundary object di sequence diagram ini. Kita juga menerapkan dua buah operasi pada objek tersebut yaitu findbook() untuk mencari buku dan addBookToCart() untuk menambahkan buku kedalam cart sesuai dengan bentuk awal dari sequence diagram ini. Setelah kita memiliki boundary object, perlu diingat bahwa boundary tidak boleh berkomunikasi dengan entity, tetapi harus melalui sebuah control object. Oleh karena itu, kita harus menambahkan control object di sequence diagram tersebut.



Gambar 9.5 Menambahkan 2 fungsi pada Sequence Diagram.

Pada kasus ini terdapat dua buah fungsi yang harus difasilitasi oleh control object yaitu mencari buku dan menambahkan buku yang ditemukan kedalam cart. Karena dua hal tersebut memiliki concern yang berbeda, kita menambahkan dua buah control object: BookFinder untuk menjalankan fungsi-fungsi pencarian buku dan CartManager untuk menjalankan fungsi manajemen cart. Kita juga menambahkan operasi yang dibutuhkan untuk menyelesaikan use case yang terkait. Setelah robustness diagram selesai diterapkan pada sebuah sequence diagram, langkah berikutnya adalah memperbaiki class diagram sesuai dengan objek dan operasi baru yang dibuat. Penerapan kode dari sequence diagram dan class diagram yang sudah diperbaiki sama dengan yang sudah dijelaskan pada dua modul sebelumnya.

TUGAS PENDAHULUAN

Membuat ROBUSTNESS ANALYSIS

Dari dokumen DPPL anda, pilihlah use case paling kompleks di Use Case Diagram.

Lihat kembali Use Case Scenarionya.

Lihat kembali Sequence Diagramnya.

Lihat kembali button-buton yang ada di *user interface* untuk use case tersebut.

Lakukan Analisis Robustness (Menyempurnakan Sequence Diagram) terhadap use case tersebut.

Dan laporkan hasilnya dalam dokumen DPPL, sebelum masuk Lab.

KEGIATAN PRAKTIKUM

Implementasikan hasil Analisis Robustness Anda dalam Pemrograman Berorientasi Objek di Lab

#Selamat bekerja

#Laporkan hasilnya ke Asisten



Modul 10 DOKUMENTASI PERANCANGAN PERANGKAT LUNAK, SOURCE CODE DAN USER MANUAL

TUJUAN PRAKTIKUM
1. mampu mendokumentasikan perancangan perangkat lunak
2. mampu mendokumentasikan <i>source code</i>
3. mampu mendokumentasikan <i>user manual</i>

10.1 Dokumentasi Perangkat Lunak

Dokumentasi adalah komponen pembentuk perangkat lunak selain kode program dan struktur data. Dokumentasi yang baik dapat membantu tim pengembang dalam mengkomunikasikan informasi-informasi yang ada mengenai perangkat lunak yang dibangun. Penggunaan pendekatan dokumentasi yang baik dapat menjadi pembeda antara perangkat lunak yang dapat dipelihara untuk waktu yang panjang dan tidak.

Dokumentasi untuk desain perangkat lunak disebut Dokumen Perancangan Perangkat Lunak (DPPL). Di dalamnya terdapat bagian-bagian yang menjelaskan hasil desain dari perangkat lunak yang akan maupun sudah dibangun, rationale terhadap keputusan desain yang dilakukan, serta keterkaitan antara hasil desain dengan kebutuhan yang dipenuhi.

10.2 Template Dokumen Perancangan Perangkat Lunak

Dalam membangun sebuah dokumen, template dapat menjadi panduan untuk membangun dokumen yang komplet. Pada praktikum mata kuliah ini, template yang digunakan adalah template DPPL yang merupakan bagian dari satuan template dokumen tugas akhir capstone. Isi dari template tersebut antara lain:

1. Pendahuluan
 - 1.1. Tujuan Penulisan Dokumen
 - 1.2. Lingkup Masalah
 - 1.3. Definisi dan istilah
 - 1.4. Referensi
 - 1.5. Deskripsi Umum Dokumen
2. Deskripsi Perancangan
 - 2.1. Rancangan Perangkat Keras
 - 2.1.1. Rancangan Arsitektur Perangkat Keras
 - 2.1.2. Dasar teori
 - 2.2. Rancangan Perangkat Lunak
 - 2.2.1. Rancangan Arsitektur Perangkat Lunak
 - 2.2.2. Rancangan Detil Perangkat Lunak

2.2.3. Rancangan Data

3. Perancangan Antarmuka Manusia
4. Referensi

Perlu diingat bahwa penggunaan template dapat membatasi informasi yang dapat disampaikan dalam sebuah dokumen. Agar template dapat digunakan dengan bijak perlu dipahami bahwa bagian-bagian template dapat ditambahkan dan dikurangi sesuai dengan kebutuhan informasi dari pembaca dokumen tersebut.

10.3 Dokumen *Source Code*

Source code yang baik adalah source code yang tidak hanya programmer pembuatnya yang bisa membaca.

Semua programmer lain bisa membacanya.

Untuk itu setiap source code perlu diberi catatan/komentar (*):

- Pendahuluan : tujuan code, deskripsi singkat, rujukan dokumen DPPL
- Penjelasan data yang diolah
- Penjelasan setiap method
- Penjelasan line code yang berisi rumus/inti pemrosesan

10.4 Dokumen *User Manual*

Dokumen User Manual dibuat untuk memandu pengguna menjalankan dan memanfaatkan aplikasi.

Template user Manual kurang lebih sebagai berikut:

1. PENDAHULUAN
 - 1.1. Tujuan Pembuatan Dokumen
 - 1.2. Deskripsi Umum System
 - 1.2.1. Deskripsi Umum Aplikasi
 - 1.2.2. Deskripsi Umum Kebutuhan Aplikasi
 - 1.3. Deskripsi Dokument (Ikhtisar)
2. PERANGKAT YANG DIBUTUHKAN
 - 2.1. Perangkat Lunak
 - 2.2. Perangkat Keras
 - 2.3. Pengguna Aplikasi
 - 2.4. Pengenalan Dan Pelatihan
3. MENU DAN CARA PENGGUNAAN
 - 3.1. Struktur Menu
 - 3.2. Pengguna

- 3.2.1. Cara Membuka Situs
 - 3.2.2. Halaman Admin
 - 3.2.3. Menu A
 - 3.2.4. Menu B
 - 3.2.5. Menu
- dst

KEGIATAN PRAKTIKUM

Manfaatkan :

- slide kuliah topik terkait
- hasil browsing internet terkait pembuatan dokumentasi code Java 'Javadoc'
- hasil browsing internet terkait 'Natural Docs' di web <http://www.naturaldocs.org>
- hasil browsing internet terkait dokumentasi bahasa pemrograman yang dipakai setiap kelompok

Buatlah : **Dokumentasi Code** dari code yang sudah berhasil dibuat selama ini di Lab, dimana *setiap anggota kelompok membuat satu dokumen Source Code.*

#Selamat bekerja

#Laporkan hasilnya ke Asisten di akhir kegiatan Lab

Modul 11 DESIGN PATTERN

TUJUAN PRAKTIKUM

1. Mampu mengimplementasikan design pattern singleton.
2. Mampu mengimplementasikan design pattern adapter.
3. Mampu mengimplementasikan design pattern command.

Design patterns adalah sebuah solusi untuk sebuah problem yang sering terjadi pada software design. Hal tersebut adalah sebuah blueprints yang sudah terbuat sehingga bisa di sesuaikan untuk memecahkan sebuah masalah design yang terus terjadi pada kodingan kalian.

Kalian tidak bisa membuat solusi hanya dengan menemukan sebuah pattern dan copy pattern tersebut kedalam program kalian, tetapi dengan menggunakan perkumpulan library atau function. Pattern tersebut bukanlah sebuah program, tetapi sebuah konsep umum untuk memecahkan sebuah masalah yang spesifik. Kalian bisa mengikuti detail pattern dan mengimplementasi sebuah solusi yang cocok pada program yang kalian buat

Patterns sering di samakan dengan sebuah algoritma, karena dua konsep tersebut menggambarkan sebuah solusi untuk sebuah masalah. Walaupun sebuah algoritma selalu didefinisikan sebagai perkumpulan Tindakan yang dicapai untuk memenuhi tujuan, sedangkan sebuah pattern adalah sebuah deskripsi level yang tinggi pada sebuah solusi . sebuah code dari pattern yang sama diterapkan pada dua algoritma bisa saja berbeda.

Sebuah analogi dari algoritma adalah resep: memiliki sebuah step yang jelas untuk mencapai sebuah tujuan. Di samping itu, sebuah pattern adalah sebuah blueprint: kalian bisa melihat hasil yang seperti apa dan fitur yang seperti apa, tetapi urutan untuk mengimplementasi sesuai dengan kemauan kalian.

Pattern terdiri dari apa saja?

Banyak pattern yang digambarkan dengan formal sehingga orang dapat memproduksi pattern dengan bermacam macam contex. Berikut adalah bagian yang biasa dipakai dalam definisi pattern:

- **Inten** dalam sebuah pattern dengan singkat menjelaskan masalah dan juga solusi.
- **Motivasi** lebih dalam menjelaskan masalah dan solusi yang mungkin pada pattern.
- **Struktur** perkumpulan class menunjukan setiap bagian dari pattern dan bagaimana mereka terkait.
- **Contoh code** pada salah satu Bahasa pemrograman yang populer mempermudah untuk memahami ide pada pattern.

Beberapa katalog pattern memiliki hal penting yang lain, seperti penerapan pada sebuah pattern, Langkah implementasi dan relasi pada pattern lain nya.

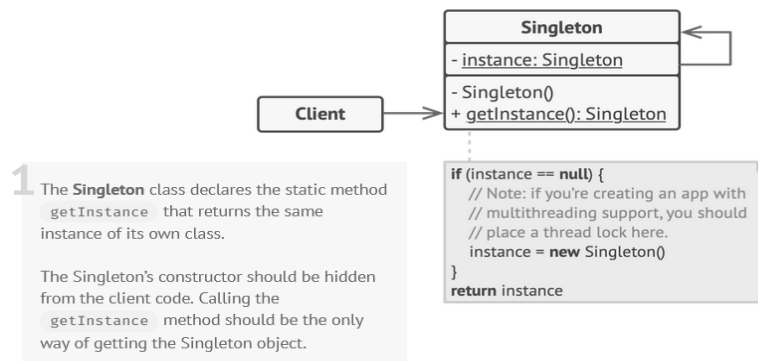
Design patterns awalnya dikategorikan menjadi 3 sub-klasifikasi berdasarkan jenis masalah yang di pecahkan.

1. [Creational patterns](#) menyediakan kemampuan untuk membuat object sesuai dengan kriteria yang di butuhkan.
2. [Structural patterns](#) adalah tentang mengatur class dan object yang berbeda untuk membuat struktur yang besar dan menyediakan fungsionalitas yang baru

3. [Behavioral patterns](#) adalah tentang mengidentifikasi komunikasi umum pada pattern antara object dan menyadari bahwa hal tersebut adalah pattern

11.1 Singleton

Singleton adalah sebuah Creational pattern yang dapat kamu pastikan pada sebuah class yang hanya memiliki satu instance, Ketika memberikan akses point global untuk instance tersebut. berikut adalah contoh struktur singleton:



Gambar 11.1 Struktur Singleton.

berikut adalah contoh Pseduocode, dimana class connection berperan sebagai singleton. Class tersebut tidak memiliki public constructor, karena itu salah satu cara untuk memanggil object tersebut adalah dengan memanggil method `getInstance`. Method tersebut menyimpan object yang pertama dibuat dan mengembalikan panggilan selanjutnya. Berikut ini adalah contoh code singleton:

```
using System;  
  
namespace Singleton  
{  
    // The Singleton class defines the `GetInstance` method that  
    serves as an  
    // alternative to constructor and lets clients access the same  
    instance of  
    // this class over and over.  
    class Singleton  
    {  
        // The Singleton's constructor should always be private to  
        prevent  
        // direct construction calls with the `new` operator.  
        private Singleton() { }  
  
        // The Singleton's instance is stored in a static field. There  
        there are  
        // multiple ways to initialize this field, all of them have  
        various pros  
        // and cons. In this example we'll show the simplest of these  
        ways,  
        // which, however, doesn't work really well in multithreaded  
        program.  
        private static Singleton _instance;  
    }  
}
```

```

        // This is the static method that controls the access to the
        singleton
        // instance. On the first run, it creates a singleton object
        and places
        // it into the static field. On subsequent runs, it returns
        the client
        // existing object stored in the static field.
        public static Singleton GetInstance()
        {
            if (_instance == null)
            {
                _instance = new Singleton();
            }
            return _instance;
        }

        // Finally, any singleton should define some business logic,
        which can
        // be executed on its instance.
        public static void someBusinessLogic()
        {
            // ...
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // The client code.
            Singleton s1 = Singleton.GetInstance();
            Singleton s2 = Singleton.GetInstance();

            if (s1 == s2)
            {
                Console.WriteLine("Singleton works, both variables
contain the same instance.");
            }
            else
            {
                Console.WriteLine("Singleton failed, variables contain
different instances.");
            }
        }
    }
}

```

11.2 Adapter

Adapter adalah sebuah Structural design pattern, dimana memberikan object yang tidak cocok untuk di kolaborasikan. Adapter berperan sebagai pembungkus antara dua object. Mereka menyimpan panggilan untuk satu object dan merubahkan mereka kedalam bentuk format dan interface yang dapat di ketahui dengan object yang kedua.

Contoh yang berada di bawah ini mengilustrasikan struktur pola design Adapter. Ini berfokus pada menjawab pertanyaan – pertanyaan dibawah:

- Terdiri dari class apa saja?
- Peran apakah yang ada tiap class?
- Dengan cara apa elemen pattern tersebut terkait?

```
using System;

namespace DesignPatterns.Adapter.Conceptual
{
    // The Target defines the domain-specific interface used by the
    client code.
    public interface ITarget
    {
        string GetRequest();
    }

    // The Adaptee contains some useful behavior, but its interface is
    // incompatible with the existing client code. The Adaptee needs
    some
    // adaptation before the client code can use it.
    class Adaptee
    {
        public string GetSpecificRequest()
        {
            return "Specific request.";
        }
    }

    // The Adapter makes the Adaptee's interface compatible with the
    Target's
    // interface.
    class Adapter : ITarget
    {
        private readonly Adaptee _adaptee;

        public Adapter(Adaptee adaptee)
        {
            this._adaptee = adaptee;
        }

        public string GetRequest()
        {
            return $"This is '{this._adaptee.GetSpecificRequest()}'";
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Adaptee adaptee = new Adaptee();
            ITarget target = new Adapter(adaptee);
        }
    }
}
```

```

        Console.WriteLine("Adaptee interface is incompatible with
the client.");
        Console.WriteLine("But with adapter client can call it's
method.");

        Console.WriteLine(target.GetRequest());
    }
}
}

```

Berikut adalah hasil output program:

```

Adaptee interface is incompatible with the client.
But with adapter client can call it's method.
This is 'Specific request.'

```

Gambar 11.2 Hasil code.

11.3 Command

Command adalah sebuah Behavioral design pattern yang mengubah request atau operasi sederhana kedalam object. Contoh di bawah ini menggambarkan design pattern command. Ini berfokus pada menjawab pertanyaan – pertanyaan di bawah:

- Terdiri dari class apa saja?
- Peran apakah yang ada tiap class?
- Dengan cara apa elemen pattern tersebut terkait?

```

using System;

namespace DesignPatterns.Command.Conceptual
{
    // The Command interface declares a method for executing a
    command.
    public interface ICommand
    {
        void Execute();
    }

    // Some commands can implement simple operations on their own.
    class SimpleCommand : ICommand
    {
        private string _payload = string.Empty;

        public SimpleCommand(string payload)
        {
            this._payload = payload;
        }

        public void Execute()
        {
            Console.WriteLine($"SimpleCommand: See, I can do simple
things like printing ({this._payload})");
        }
    }
}

```

```

// However, some commands can delegate more complex operations to
other
// objects, called "receivers."
class ComplexCommand : ICommand
{
    private Receiver _receiver;

    // Context data, required for launching the receiver's
methods.
    private string _a;

    private string _b;

    // Complex commands can accept one or several receiver objects
along
    // with any context data via the constructor.
    public ComplexCommand(Receiver receiver, string a, string b)
    {
        this._receiver = receiver;
        this._a = a;
        this._b = b;
    }

    // Commands can delegate to any methods of a receiver.
    public void Execute()
    {
        Console.WriteLine("ComplexCommand: Complex stuff should be
done by a receiver object.");
        this._receiver.DoSomething(this._a);
        this._receiver.DoSomethingElse(this._b);
    }
}

// The Receiver classes contain some important business logic.
They know how
// to perform all kinds of operations, associated with carrying
out a
// request. In fact, any class may serve as a Receiver.
class Receiver
{
    public void DoSomething(string a)
    {
        Console.WriteLine($"Receiver: Working on ({a}.)");
    }

    public void DoSomethingElse(string b)
    {
        Console.WriteLine($"Receiver: Also working on ({b}.)");
    }
}

// The Invoker is associated with one or several commands. It
sends a
// request to the command.
class Invoker
{
    private ICommand _onStart;

```

```

private ICommand _onFinish;

// Initialize commands.
public void SetOnStart(ICommand command)
{
    this._onStart = command;
}

public void SetOnFinish(ICommand command)
{
    this._onFinish = command;
}

// The Invoker does not depend on concrete command or receiver
classes.
// The Invoker passes a request to a receiver indirectly, by
executing a
// command.
public void DoSomethingImportant()
{
    Console.WriteLine("Invoker: Does anybody want something
done before I begin?");
    if (this._onStart is ICommand)
    {
        this._onStart.Execute();
    }
    Console.WriteLine("Invoker: ...doing something really
important...");
    Console.WriteLine("Invoker: Does anybody want something
done after I finish?");
    if (this._onFinish is ICommand)
    {
        this._onFinish.Execute();
    }
}

class Program
{
    static void Main(string[] args)
    {
        // The client code can parameterize an invoker with any
commands.
        Invoker invoker = new Invoker();
        invoker.SetOnStart(new SimpleCommand("Say Hi!"));
        Receiver receiver = new Receiver();
        invoker.SetOnFinish(new ComplexCommand(receiver, "Send
email", "Save report"));

        invoker.DoSomethingImportant();
    }
}

```

KEGIATAN PRAKTIKUM

DESIGN PATTERN

Dari 23 design pattern yang ditemukan *Gang of Four*, *Singleton* adalah yang paling menarik diperhatikan.

Dengan pattern singleton, dijamin hanya satu objek yang dicreate untuk mengakses resource bersama (misalnya basis data).

Coba lakukan modifikasi code pengaksesan basis data Tubes Anda, hingga nampak implementasi dari design pattern Singleton.

Jika dalam aplikasi TUBES Anda tidak memungkinkan diimplementasikan Singleton, silahkan cari design pattern yang lain untuk diimplementasikan.

Sembari mengerjakan hal tersebut, karena saat ini adalah minggu ke-11, tinggal dua minggu ke depan waktu untuk implementasi (code) dari desain aplikasi di DPPL, sempurnakan code TUBES Anda hingga mencapai 85%.

#Selamat bekerja

#Laporkan hasil kegiatan Lab ke Asisten



Modul 12 REFACTORING

TUJUAN PRAKTIKUM

- | |
|---|
| 1. Mampu melakukan refactoring sederhana. |
|---|

12.1 Refactoring

Refactoring adalah sebuah proses untuk memperbaiki sebuah kode program tanpa mengubah fungsionalitas dari kode program tersebut untuk menghasilkan kode program yang lebih baik. Sebuah kode program yang baik adalah kode program yang dapat dibaca, dipahami, dan dipelihara dengan mudah. Perangkat lunak yang memiliki kode program yang baik akan mempermudah proses pengembangan dan pemeliharaan pasca produksi. Kode program yang memiliki kualitas-kualitas tersebut disebut juga dengan *clean code*.

Kebalikan dari *clean code* adalah *dirty code* yang dapat muncul karena beberapa hal. Sebuah kode mungkin ditulis secara terburu-buru untuk mengejar *deadline* dari proyek pengembangan. Selain itu ketidakpahaman atau ketidakpedulian pengembang mengenai pentingnya kualitas dari kode program yang ditulis juga dapat menghasilkan kode program yang buruk. Sebuah *dirty code* ditulis dengan buruk sehingga menyebabkan kode program tidak dapat dipahami lagi setelah beberapa waktu.

12.2 Code smells

Agar dapat melakukan refactoring seorang programmer perlu mendeteksi apakah sebuah kode program yang termasuk ke dalam *dirty code*. Untuk melakukan pendeteksian tersebut kita dapat menggunakan apa yang disebut dengan *code smells*. *Code smells* adalah karakteristik-karakteristik yang ada pada sebuah kode program yang dapat menunjukkan adanya kemungkinan terdapat masalah pada kode program tersebut. Tim pengembang kemudian dapat mengevaluasi lebih lanjut apakah kode program tersebut bermasalah atau tidak.

Beberapa beberapa *code smell* dan penjelasannya:

1. Duplicated code
Kode program yang identik atau serupa muncul di dua tempat yang berbeda atau lebih. Dapat menyulitkan dalam melakukan perubahan dan perbaikan terhadap fungsi dari kode tersebut karena dapat menyebabkan perbaikan-perbaikan yang tidak menyeluruh.
2. Long method
Method yang terlalu besar dan menyelesaikan lebih dari tugas sehingga sulit untuk memahami tugas dari method tersebut.
3. Large class dan Lazy class
Large class adalah sebuah class yang memiliki terlalu banyak attribute dan method, sebaliknya lazy class adalah sebuah class yang hanya memiliki sedikit attribute dan method.
4. Long parameter list
Sebuah method yang memiliki parameter yang banyak sehingga menyulitkan pemanggilan.

12.3 Refactoring Strategy

Dalam upaya memperbaiki *dirty code* yang sudah ditemukan, beberapa pola strategi *refactoring* sudah didefinisikan. Setiap strategi *refactoring* memiliki tujuan yang berbeda, sesuai dengan masalah yang ditemukan pada *dirty code*. Berikut beberapa jenis refactoring strategy yang umum dipakai beserta contohnya:

1. Extract method

Extract method bertujuan untuk memisahkan sebuah blok program yang memiliki fungsi tersendiri dari sebuah long method. Berikut contoh dari penerapan extract method:

```
import java.util.LinkedList;

public class Contoh {

    LinkedList<Student> students;

    public boolean AddNewStudent(Student newStudent) {

        if (newStudent.age < 17) {
            return false;
        }

        if (newStudent.id.length() < 10) {
            return false;
        }

        if (newStudent.name.length() > 15) {
            return false;
        }

        students.add(newStudent);
        return true;
    }

}
```

Gambar 12.1 Contoh *dirty code*.

Dapat dilihat bahwa kode diatas memiliki method yang cukup panjang, dimana ada 4 fungsi yang dilakukan: memvalidasi umur, memvalidasi panjang dari ID, memvalidasi panjang dari nama, dan menambahkan data ke dalam list. Barisan kode yang memiliki fungsi yang berbeda dapat dipisahkan menjadi method tersendiri seperti di bawah ini.

```
import java.util.LinkedList;

public class Contoh {

    LinkedList<Student> students;

    public boolean AddNewStudent(Student newStudent) {

        if (IsAgeValid(newStudent) && IsIdValid(newStudent) && IsNameValid(newStudent)) {
            students.add(newStudent);
            return true;
        }

        return false;
    }

    public boolean IsAgeValid(Student studentData) {
        return studentData.age > 17;
    }

    public boolean IsIdValid(Student studentData) {
        return studentData.id.length() > 10;
    }

    public boolean IsNameValid(Student studentData) {
        return studentData.id.length() < 15;
    }

}
```

Gambar 12.2 Contoh hasil extract method.

Pada refactoring tersebut dilakukan ekstraksi baris kode yang memiliki fungsi masing-masing menjadi sebuah method yang terpisah. Method yang dihasilkan dari ekstraksi kemudian dipanggil pada method asal agar fungsi dari perangkat lunak yang dibangun tidak berubah.

2. Extract class

Extract class bertujuan untuk memastikan bahwa sebuah class hanya memiliki satu tanggung jawab sehingga tidak menjadi *large class*. Hal ini dilakukan dengan memindahkan operasi dan atribut yang memiliki tanggung jawab yang berbeda ke class masing-masing. Berikut contoh dari penerapan extract class.

```
public class Student {  
    private String name;  
    private int age;  
    private String id;  
    private String address;  
    private String phoneNumber;  
    private String email;  
  
    public Student(String name, String id, int age, String address, String phoneNumber, String email) {  
        this.name = name;  
        this.id = id;  
        this.age = age;  
        this.address = address;  
        this.phoneNumber = phoneNumber;  
        this.email = email;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public String getId() {  
        return id;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public String getAddress() {  
        return address;  
    }  
  
    public String getPhoneNumber() {  
        return phoneNumber;  
    }  
}
```

Gambar 12.3 Contoh *dirty code*.

Pada *dirty code* di atas dapat dilihat bahwa class student memiliki tanggung jawab untuk menyimpan data mahasiswa. Namun dapat dilihat juga bahwa sebenarnya tanggung jawab kelas tersebut dapat dipecah menjadi bagian yang lebih kecil, yaitu menyimpan data umum mahasiswa (nama, umur, id) dan data kontak mahasiswa (alamat, no telp., dan email). Oleh karena itu, *extract class* dapat diterapkan untuk membuat kode di atas menjadi lebih baik. Hasil penerapan *extract class* sebagai berikut:

```

public class AddressInformation {

    private String address;
    private String phoneNumber;
    private String email;

    public AddressInformation(String address, String phoneNumber, String email) {
        this.address = address;
        this.phoneNumber = phoneNumber;
        this.email = email;
    }

    public String getAddress() {
        return address;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public String getEmail() {
        return email;
    }

}

```

Gambar 12.4 Class AddressInformation.

Hal yang pertama dilakukan adalah memindahkan atribut dan operasi yang berhubungan dengan alamat ke dalam class baru yang dalam contoh ini dinamakan AddressInformation.

```

public class Student {

    private String name;
    private int age;
    private String id;
    private AddressInformation addressInformation;

    public Student(String name, String id, int age, AddressInformation addressInformation) {
        this.name = name;
        this.id = id;
        this.age = age;
        this.addressInformation = addressInformation;
    }

    public String getName() {
        return name;
    }

    public String getId() {
        return id;
    }

    public int getAge() {
        return age;
    }

    public AddressInformation getAddressInformation() {
        return addressInformation;
    }

}

```

Gambar 12.5 Class Student.

Pointer obyek dari class tersebut kemudian akan disimpan dalam class yang lama (student) dan disediakan tempat mengaksesnya agar fungsi keseluruhan perangkat lunak tetap sama.

TUGAS PENDAHULUAN

Kunjungi www.refactoring.com dan pelajari slide kuliah terkait.
Cari setidaknya 10 macam refactoring dari bermacam kategori.
Laporkan sebelum masuk Lab.

KEGIATAN PRAKTIKUM

Dari coding yang sudah Anda lakukan di aplikasi Tugas Besar Anda,
cari barangkali ada yang kurang efisien,
buat lebih efisien dengan cara-cara yang dijelaskan di konsep refactoring.

Sembari mengerjakan hal tersebut, karena saat ini adalah minggu ke-12, tinggal satu minggu ke depan waktu untuk implementasi (code) dari desain aplikasi di DPPL, maka sempurnakan code TUBES Anda hingga mencapai 92%.

#Selamat bekerja

#Laporkan hasil pekerjaan Kelompok Anda ke Asisten



Modul 13 UNIT TESTING

TUJUAN PRAKTIKUM

1. Mampu membuat unit testing sederhana.

Unit tests biasanya tes otomatis yang ditulis dan dijalankan oleh pengembang perangkat lunak untuk memastikan bahwa bagian dari aplikasi (dikenal sebagai "unit") memenuhi desainnya dan berperilaku seperti yang diinginkan. Dalam pemrograman prosedural, sebuah unit bisa menjadi seluruh modul, tetapi lebih umum merupakan fungsi atau procedure individual. Dalam pemrograman berorientasi objek, unit sering kali merupakan seluruh antarmuka, seperti kelas, tetapi bisa menjadi metode individual. Dengan menulis tes terlebih dahulu untuk unit terkecil yang dapat diuji, kemudian perilaku gabungan di antara mereka, seseorang dapat membangun tes komprehensif untuk aplikasi yang kompleks.

Untuk mengisolasi masalah yang mungkin timbul, setiap kasus uji harus diuji secara independen. Pengganti seperti stub metode, objek tiruan, palsu, dan test harness dapat digunakan untuk membantu pengujian modul secara terpisah.

Selama pengembangan, pengembang perangkat lunak dapat menuliskan kriteria, atau hasil yang diketahui baik, ke dalam pengujian untuk memverifikasi kebenaran unit. Selama eksekusi kasus uji, framework mencatat pengujian yang gagal dalam kriteria apa pun dan melaporkannya dalam ringkasan. Untuk ini, pendekatan yang paling umum digunakan adalah uji - fungsi - nilai yang diharapkan.

13.1 Membuat Unit Test

Berikut ini langkah langkah untuk melakukan unit test:

Buka the project yang akan dilakukan unit test di eclipse. Untuk Kemudahan contoh buatlah sebuah project baru bernama **"CalculatorUnitTest"** dan buat class **"Calculator"** sebagai berikut:

```
public class Calculator {  
  
    public int add(int a, int b) {  
        return a+b;  
    }  
  
    public int substract(int a, int b) {  
        return a-b;  
    }  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
  
    public int divide(int a, int b) {  
        return a / b;  
    }  
  
}
```

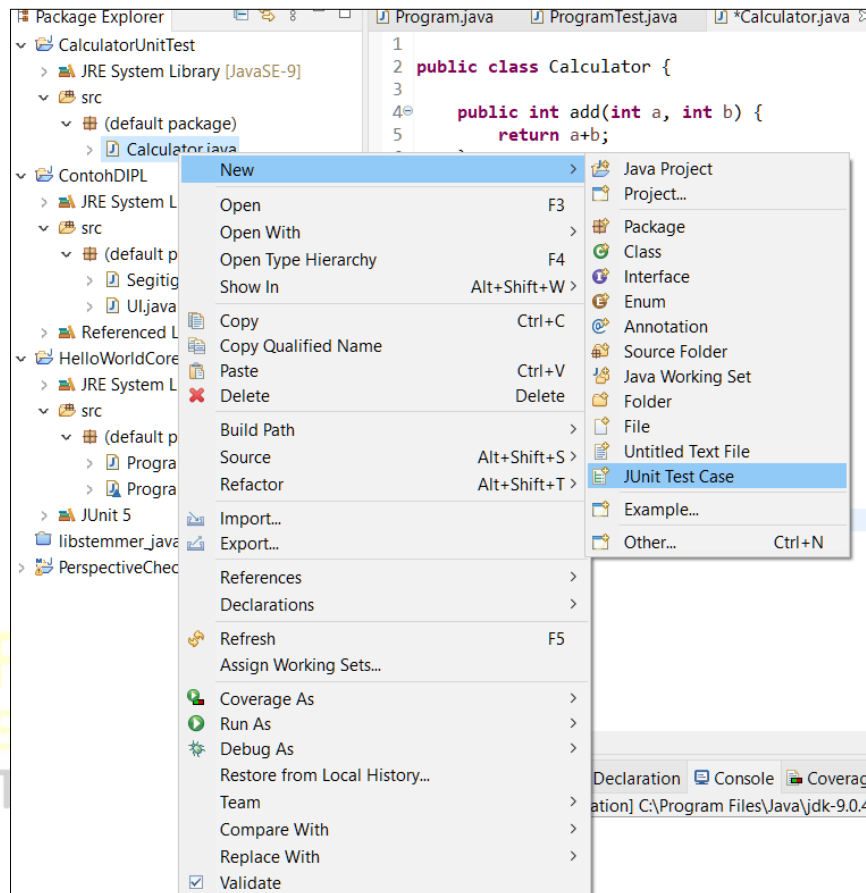
Gambar 13.1 Class Calculator.

Lalu kita buat class unit testnya dengan melakukan hal-hal berikut:

1. Klik kanan pada file “Calculator.java”

2. Pilih new

3. Pilih JUnit TestCase



Gambar 13.2 JUnit Test Case.

4. Eclipse akan memberi nama JUnit Test Case yang dibuat menjadi “CalculatorTest” secara otomatis.

5. Pilih “Finish”, klik ok jika ada prompt untuk menambahkan JUnit ke build path.

6. Pada “CalculatorTest.java” tuliskan test sebagai berikut:

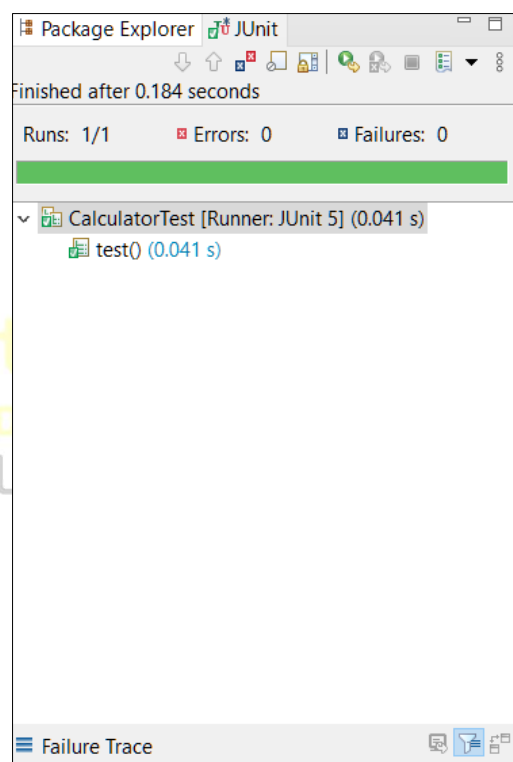
```
import static org.junit.jupiter.api.Assertions.*;

class CalculatorTest {

    @Test
    void test() {
        Calculator calculator = new Calculator();
        int a = 10;
        int b = 3;
        assertEquals(a+b, calculator.add(a, b)); //menguji apakah nilai yang dimasukan sama
    }
}
```

Gambar 13.3 Class CalculatorTest.

7. Jalankan unit test dengan menekan fungsi run. Pada sebelah kiri akan muncul windows Junit.



Gambar 13.4 Package Explorer window.

8. Cobalah mengubah code unit test yang sudah dibuat! Gunakan bentuk assertion yang lain, uji juga metode lain yang ada pada "Calculator.Java"!

TUGAS PENDAHULUAN

Pelajari dengan teliti konsep *Cyclomatic Complexity (CC)*.

Cari method suatu kelas di aplikasi Tugas Besar Anda yang paling banyak pemeriksaan kondisionalnya (perintah "if" dan atau perintah "while" nya).

Kemudian buat "Test Case"/kasus pengujian sebanyak angka CC yang ditemukan.

Laporkan hasilnya.

KEGIATAN PRAKTIKUM

Dari Tugas Pendahuluan,
buat code di JUnit (atau semisalnya) untuk menerapkan semua "test case" yang ditemukan tersebut.

Sembari mengerjakan hal tersebut, karena saat ini adalah minggu ke-13,
maka sempurnakan code TUBES Anda hingga mencapai $\geq 95\%$.

#Selamat Bekerja

#Laporkan hasilnya ke Asisten



Modul 14 PRESENTASI

TUJUAN PRAKTIKUM

- | |
|--|
| 1. Mampu menjelaskan rationale perancangan dan implementasi. |
|--|

14.1 Pengantar

Salah satu aspek utama dalam rekayasa perangkat lunak adalah kemampuan dari tim pengembang untuk mempertanggungjawabkan apa yang dihasilkan, termasuk desain dan implementasi kode. Pada modul terakhir ini anda dan kelompok anda diharuskan untuk menunjukkan kemampuan dalam mempertanggungjawabkan apa yang sudah anda hasilkan dalam tugas besar yang sudah anda kerjakan pada mata kuliah Desain dan Implementasi Perangkat Lunak.

Pada presentasi anda akan diminta untuk menunjukkan hal-hal sebagai berikut:

1. Kesesuaian desain dengan requirement.
2. Kesesuaian implementasi dengan desain.
3. Kesuksesan hasil implementasi untuk dijalankan.
4. Pemahaman terhadap kode program yang sudah diselesaikan.

14.2 Persiapan Presentasi

Sebelum melakukan presentasi, siapkan hal-hal berikut ini agar anda dapat melaksanakan presentasi dengan baik:

1. Dokumen DPPL yang sudah anda bangun bersama kelompok.
2. File executable dari kode yang sudah anda implementasi.
3. Simpanlah source code yang sudah anda implementasi pada github.
4. Pahami kode yang anda buat.
5. Pahami kode yang teman anda buat.

DAFTAR PUSTAKA

- [1] Tim Dosen MK SISTER. 2020. *Slide Perkuliahan Sister Paralel dan Terdistribusi*. Fakultas Informatika. Telkom University, Bandung.
- [2] "What's a design pattern?", Refactoring.guru, 2021. [Online]. Available: <https://refactoring.guru/design-patterns/what-is-pattern>. [Accessed: 12- Feb- 2021]
- [3] k. ganfield, "Writing JUnit Tests in NetBeans IDE", Netbeans.org, 2021. [Online]. Available: https://netbeans.org/kb/docs/java/junit-intro.html#Exercise_20. [Accessed: 12- Feb- 2021]
- [4] "A Practical Tutorial on Robustness Analysis", Visual-paradigm.com, 2021. [Online]. Available: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/robustness-analysis-tutorial/>. [Accessed: 12- Feb- 2021]
- [5] Software Engineering Body of Knowledge Guide V3.0.
- [6] Sommerville, Software Engineering Ninth Edition, Addison-Wesley, 2011.
- [7] Jeffry L. Wihitten, Lonnie D. Bentley, System Analysis and Design Methods 7ed, Mc-Graw Hill, 2007.
- [8] Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language, Addison Wesley, 1999.
- [9] E. Freeman, E. Robson and B. Bates, Head First Design Pattern, California: O'Reilly Media, Inc., 2004.