BAB III
ANALISIS DAN PERANCANGAN SISTEM

Analisis dan perancangan sistem adalah tahapan sistematis untuk mendapatkan aplikasi yang baik dan bersesuaian dengan kegunaan dan tujuannya. Tahap awal dari analisis adalah menganalisis kebutuhan-kebutuhan sistem mulai dari kebutuhan pengguna, kebutuhan non-fungsional dan kebutuhan fungsional. Sedangkan untuk tahapan perancangan aplikasi yaitu perancangan *storyboard* pembelajaran, perancangan sistem dan perancangan antarmuka.

3.1. Gambaran Sistem

Aplikasi *e-learning* untuk pemrograman berorientasi objek ini adalah sebuah aplikasi berbasis *website* dimana mahasiswa teknik informatika dapat mempelajari konsep pemrograman berorientasi objek menggunakan bahasa pemrograman Java.

3.2. Analisis Sistem

Analisis sistem dapat didefinisikan sebagai penguraian dari suatu sistem informasi yang utuh kedalam bagian-bagian komponennya dengan maksud untuk mengidentifikasi dan mengevaluasi permasalahan, kesempatan serta hambatan yang terjadi dan kebutuhan-kebutuhan yang diharapkan sehingga dapat sesuai dengan kebutuhan. Dalam membangun perangkat lunak ini dilakukan beberapa analisis yaitu:

1. Analisis Masalah
2. Analisis Kebutuhan Non-Fungsional
3. Analisis Kebutuhan Fungsional

3.2.1. Analisis Masalah

Menurut hasil pengamatan terhadap beberapa buku referensi pemrograman berorientasi objek dan hasil kuisioner dari beberapa mahasiswa teknik informatika Universitas Komputer Indonesia yang telah mengambil mata kuliah pemrograman dapat disimpulkan bahwa mahasiswa mengalami kesulitan

dalam memahami konsep pemrograman berorientasi objek dikarenakan perbedaan cara penulisan *source code* dalam buku referensi dengan cara penulisan yang digunakan dalam *Integrated Development Environtment* (IDE) yang ada seperti *Netbeans dan Eclipse.* Antar muka IDE yang terlalu rumit juga menyulitkan mahasiswa dalam proses belajar dikarenakan mahasiswa harus mempelajari terlebih dulu antar muka yang komplek sebelum mempelajari konsep-konsep pemrograman berorientasi objek.

3.2.2. Analisis Kebutuhan Non-Fungsional

Analisis kebutuhan non-fungsional adalah sebuah langkah dimana seorang pembangun perangkat lunak (*software develover)* menganalisis sumber daya manusia yang akan menggunakan perangkat lunak yang dibangun, perangkat keras dan perangkat dimiliki sesuai dengan kebutuhan, sehingga dapat ditentukan kompabilitas aplikasi yang dibangun terhadap sumber daya yang ada.

Setelah melakukan analisis non-fungsional, maka dilanjutkan ke langkah berikutnya yaitu menentukan kebutuhan non-fungsional sistem yang akan dibangun untuk disesuaikan dengan fakta yang ada. Apabila kebutuhan tidak terpenuhi maka sistem yang dibangun tidak akan berjalan dengan baik sesuai yang diharapkan.

Analisis kebutuhan non-fungsional yang dilakukan dibagi dalam tiga tahap, yaitu :

1. Analisis perangkat keras (*hardware*).
2. Analisis perangkat lunak (*software*).
3. Analisis pengguna (*user)*.

3.2.2.1. Analisis Kebutuhan Perangkat Keras

Pembangunan aplikasi *e-learning* untuk pemrograman berorientasi objek ini menggunakan satu unit komputer dengan spesifikasi antara lain:

1. *Prosesor* : Intel® Core™ i3 350M/370M 2.26 GHz
2. *Memory* : 3GB DDR3
3. *Hardisk* : 250GB
4. VGA : Integrated Intel® GMA HD 256MB shared VGA

5. *Monitor* : 14.0" LED
6. *Modem* internet
7. *Mouse* dan *Keyboard*

Kebutuhan perangkat keras (*hardware*) yang diperlukan untuk mengimplementasikan aplikasi *e-learning* untuk pemrograman berorientasi objek ini adalah satu buah *web server online* dengan spesifikasi minimum sebagai berikut:

1. *Prosesor* : Intel Pentium 3/ AMD
2. *Memori* : 2GB
3. *Hardisk* : 80GB

Perangkat keras yang yang dibutuhkan sudah 90% memenuhi standar kebutuhan minimum spesifikasi perangkat keras yang dibutuhkan untuk membangun aplikasi *e-learning* untuk pemrograman berorientasi objek. Dibutuhkan penambahan *web server* online agar aplikasi *e-learning* untuk pemrograman berorientasi objek ini dapat diakses secara online.

3.2.2.2. Analisis Kebutuhan Perangkat Lunak

Perangkat lunak (*software*) yang digunakan dalam pembangunan aplikasi *e-learning* untuk bahasa pemrograman berorientasi objek saat ini adalah sebagai berikut:

1. Sistem Operasi : Microsoft Windows 7
2. Software Lainnya : *Netbeans, Adobe Photoshop* CS6

Spesifikasi perangkat lunak yang dibutuhkan untuk mendukung aplikasi yang akan dibangun adalah sebagai berikut:

1. Sistem Operasi : *Linux/Microsoft Windows Server*
2. *Netbeans, Adobe Photoshop*
3. Php5
4. MySql 4
5. Mozilla Firefox, Internet Explorer dan Opera sebagai *web browser*

Perangkat lunak yang ada saat ini belum memadai untuk mengimplementasikan aplikasi yang akan dibangun. Untuk itu diperlukan penambahan aplikasi yang diperlukan.

3.2.2.3. Analisis Pengguna

Spesifikasi pengguna adalah kriteria yang harus dimiliki oleh pengguna agar tidak kesulitan dalam menjalankan aplikasi *e-learning* untuk bahasa pemrograman berorientasi objek, spesifikasi penggunanya adalah sebagai berikut :

1. Mahasiswa adalah mahasiswa jurusan teknik informatika yang telah belajar pemrograman sebelumnya.
2. Mampu mengoperasikan *web browser*
3. Mempunyai akses internet

Sedangkan karakteristik user menurut analisis adalah sebagai berikut seperti terlihat pada tabel 3.1 berikut:

Tabel 3.1. Karakteristik Pengguna

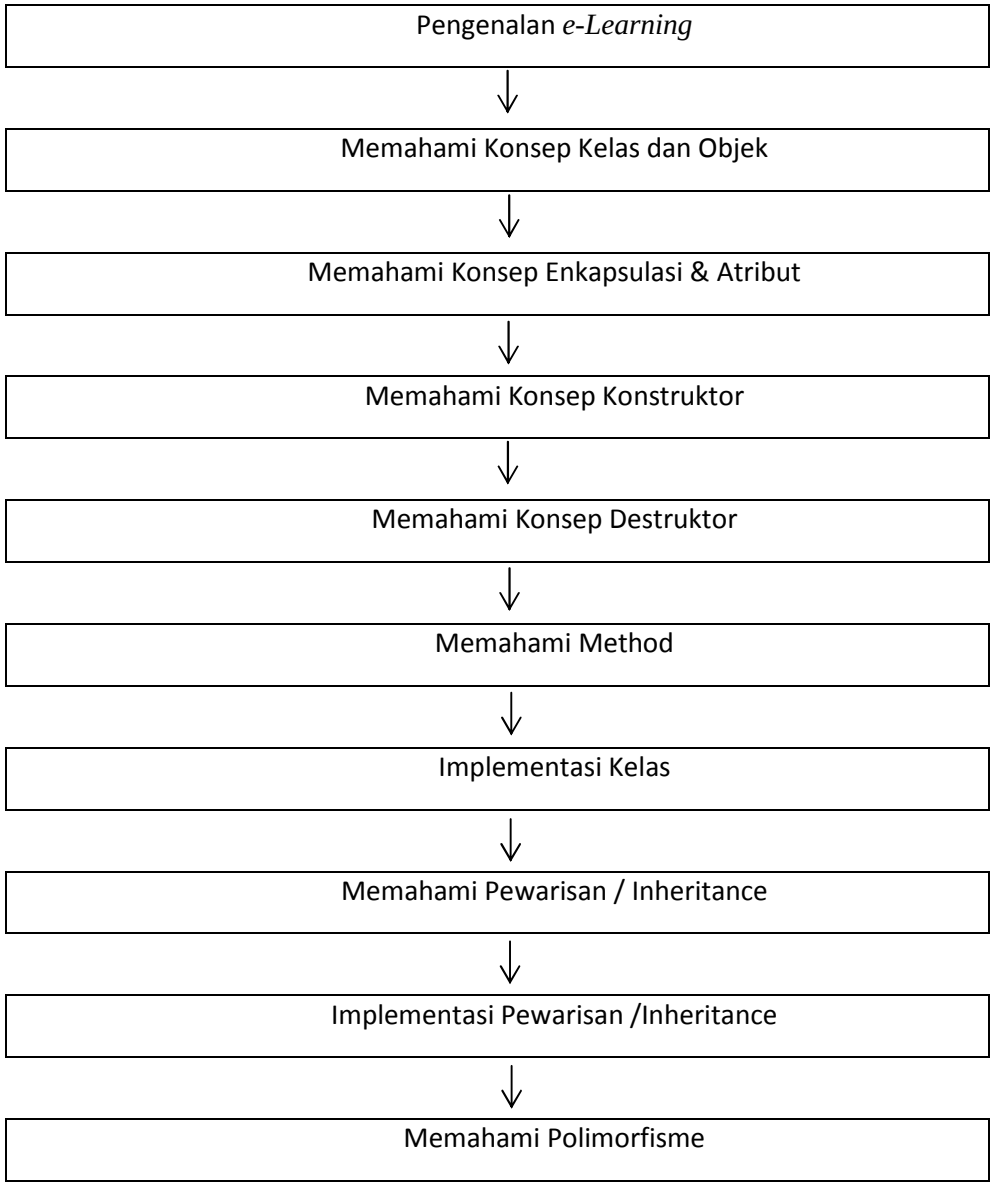
User	Hak Akses	Tingkat Pendidikan	Tingkat Keterampilan	Pengalaman
Mahasiswa	Melihat materi pembelajaran	Mahasiswa S1 Teknik Informatika/setara	Bisa mengikuti peunjuk yang ada pada sistem	Mampu mengoperasikan komputer dan bisa menggunakan aplikasi *web browser*

3.2.3. Analisis Kebutuhan Fungsional

Analisis kebutuhan fungsional menggambarkan proses kegiatan yang akan diterapkan dalam sebuah sistem dan menjelaskan kebutuhan yang diperlukan sistem agar sistem dapat berjalan dengan baik sesuai kebutuhan. Analisis kebutuhan fungsional dimodelkan dengan menggunakan *Unified Modeling Language* (UML).

3.2.3.1. Hirarki Materi Pembelajaran

Berikut ini merupakan hirarki materi pembelajaran pemrograman berorientasi objek, hirarki disusun mulai dari pengenalan media pembelajaran yakni e-learning, materi pembelajaran hingga evaluasi dapat dilihat pada gambar 3.1 berikut:



Gambar 3.1. Hirarki Materi Pembelajaran

3.2.3.2. *Storyboard* Pembelajaran

Storyboard merupakan visualisasi ide dari aplikasi yang akan dibangun, sehingga dapat memberikan gambaran dari aplikasi yang akan dihasilkan. *Storyboard* dapat dikatakan juga *visual script* yang akan dijadikan *outline* dari sebuah proyek yang ditampilkan *shot-by-shoot* yang bisa disebut dengan istilah *scene*.

Dalam pembangunan *e-learning* untuk pemrograman berorientasi objek ini digunakan pendekatan *storyboarding* untuk mempermudah dalam proses analisis kebutuhan antar muka yang diperlukan dalam penyampaian materi pembelajaran pemrograman berorientasi objek. Berdasarkan hirarki materi pembelajaran maka penyusunan *storyboard* pembelajaran yang digunakan adalah sebagai berikut:

Tahap pertama dalam pembelajaran ilah tahap pengenalan tentang media dan sistem pembelajaran yang digunakan untuk itu dibuatlah satu *scene* tentang Pengenalan applikasi *e-learning* untuk pemrograman berorientasi objek ini, didalamnya terdapat penjabaran tentang bagaimana pembelajaran diberikan dan media yang dipergunakan. Untuk lebih jelasnya bisa dilihat pada tabel 3.2 sebaga berikut:

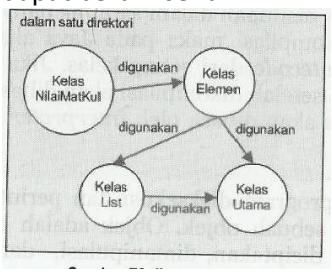
Tabel 3.2. *Storyboard* Untuk Pengenalan *E-Learning*

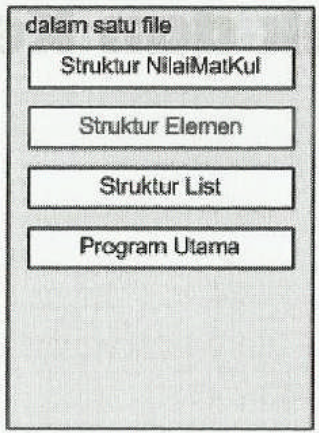
Content Screen			
Judul Scene	Pengenalan *e-learning* OOP	scene #	
Tujuan	Mengenalkan *e-learning* OOP kepada mahasiswa sebelum memulai materi agar mahasiswa lebih mengerti cara pengunaan website ini		
On Screen Teks	Selamat datang di web E-learning, dalam web ini kamu akan belajar: -Cara mengerti pembelajaran OOP dengan Mudah -Cara mengaplikasikan konsep OOP dalam bahasa Java *e-learning* ini disajikan dalam bentuk learner guide, jadi penyampaian materi lebih menekankan pada materi yang mengajak mahasiswa untuk aktif bereksperimen dengan materi yang akan disajikan Strategi Pembelajaran yang digunakan antara lain dengan menggunakan grafik atau ilustrasi sebagai penekanan terhadap materi yang diajarkan. Adapula quiz yang bertujuan untuk mengukur tingkat pemahaman		

	pribadi		
Informasi Ekstra			
Spesifikasi Grafis	Gambar selamat datang, screenshoot tentang salah satu materi yang disajikan, screenshoot quiz, screenshoot materi beserta ilustrasi yang digunakan		
Catatan			
Kembali ke scene#	-	Selanjutnya scene#	2

Tahap selanjutnya adalah pengenalan pemrogrman berorientasi objek (PBO), pada tahap ini dibuat scene untuk menjelaskan tentang konsep *class* dan *object* serta penerapannya dalam bahasa pemrograman Java. Untuk lebih jelasnya dapat dilihat pada tabel 3.3 sebagai berikut:

Tabel 3.3. *Storyboard* Untuk Pengenalan PBO

Content Screen			
Judul Scene	Pengenalan PBO	scene #	2
Tujuan Pembelajaran	Mengenalkan konsep konsep pembelajaran pemrograman berorientasi objek pada mahasiswa		
On Screen Teks	-Pengertian Objek Dan Class Sebuah class menyerupai sebuah struktur yang telah dibahas sebagai tipe data tersendiri, misalkan tipe data titik yang terdiri dari koordinat x dan y. Pada penjelasan mengenai tipe sendiri, bahasa pemrograman Java telah menggunakan sebuah class untuk menyatakan tipe data titik karena bahasa Java merupakan bahasa pemrograman berorientasi objek murni sehingga tidak mengenal struktur tapi mengenal apa yang disebut class Perbedaan sebuah class dengan sebuah struktur adalah sebuah class dapat berdiri sendiri dan dapat dipergunakan untuk berbagai keperluan kelas-kelas yang lain, sedangkan sebuah struktur tidak dapat berdiri sendiri  [ilustrasi kelas] Adapun ilustrasi sebuah struktur dapat dilihat pada gambar dibawah		

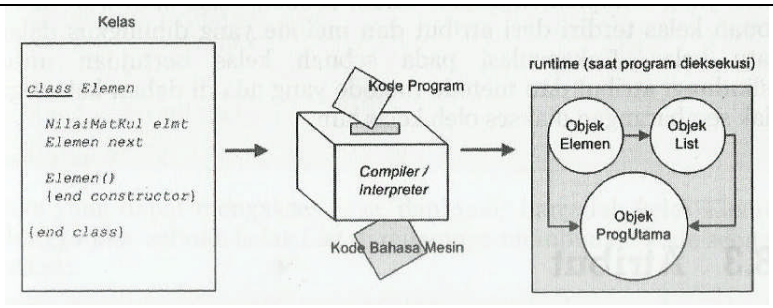


Sebuah kelas lebih fleksibel untuk digunakan oleh kelas lain tanpa harus membongkar kode program utama, sedangkan jika digunakan struktur maka kode program harus dibongkar untuk disalin bagian strukturnya ke kode program utama yang lain.
Sebuah file dapat terdiri dari beberapa kelas, namun biasanya pada bahasa pemrograman Java sebuah file hanya terdiri dari satu kelas yang disimpan dengan nama kelas, misalfiie List.java berisi kelas List, namun jika kelas yang dibuat misalnya (dalam bahasa pemrograman Java):

```
public class nama _kelas{
//……..
}
```

maka kelas itu harus disimpan dalam satu file hanya untuk satu kelas. Setelah dilakukan kompilasi maka pada Java akan ada sebuah file .class yang berisi bytecode dari setiap kelas. Jika sebuah file terdiri dari dua kelas maka setelah dikompilasi akan dihasilkan dua buah file .class yang nantinya akan dibaca oleh interpreter Java saat program dieksekusi.

Sebuah kelas saat program dieksekusi dan perintah new dijalankan maka akan dibuat sebuah objek. Objek adalah elemen pada saat runtime yang akan diciptakan, dimanipulasi, dan dihancurkan saat eksekusi sehingga sebuah objek hanya ada saat sebuah program dieksekusi, jika masih dalam bentuk kode, disebut sebagai kelas jadi pada saat runtime (saat sebuah program dieksekusi), yang kita punya adalah objek, di dalarn teks program yang kita lihat hanyalah kelas. Ilustrasi kelas dan objek dapat dilihat pada gambar berikut

<table>
<tr><td></td><td colspan="4">

[ilustrasi kelas dan objek]

Sebuah kelas diizinkan memiliki sebuah kelas internal di dalamnya yang biasa disebut kelas di dalam kelas (nested class), misalnya sebagai berikut:

class Elemen
NilaiMatKul elmt Elemen next
Element()
{end constructor}
Class NilaiMatKul
string nim
string nama
string nilai
nilaiMatKul()
{end NilaiMatkul}
{end class}</td></tr>
<tr><td>Informasi Ekstra</td><td colspan="4"></td></tr>
<tr><td>Spesifikasi Grafis</td><td colspan="4"></td></tr>
<tr><td>Catatan</td><td colspan="4"></td></tr>
<tr><td>Kembali ke scene#</td><td>1</td><td>Selanjutnya scene#</td><td>3</td></tr>
<tr><td colspan="5"></td></tr>
</table>

Tahap selanjutnya adalah pengenalan konsep Enkapsulasi dan Attribut dalam suatu objek dan penerapannya. Untuk lebih jelasnya dapat dilihat pada tabel 3.4 sebagai berikut:

Tabel 3.4. *Storyboard* Untuk Materi Enkapsulasi Dan Atribut

Content Screen			
Judul Scene	Enkapsulasi Dan Atribut	scene #	3
Tujuan Pembelajaran	Mengenalkan konsep konsep Enkapsulasi dan Atribut		
On Screen Teks	-Enkapsulasi Enkapsulasi dap at dianggap sebagai sebuah bungkusan. Enkapsulasi inilah yang diimplementasikan dalam sebuah kelas bahwa di dalani sebuah kelas terdiri dari atribut dan metode yang dibungkus suatu kelas. Enkapsulasi pada sebuah kelas bertujuan untuk melindungi atribut clan metode-metode yang ada di dalam kelas agar tidak sembarangan diakses oleh kelas lain -Atribut Attribut dari sebuah kelas adalah variabel global yang dimiliki oleh sebuah kelas, misalkan pada kelas sebagai berikut: Class Elemen NilaiMatKul elmt Element next Element() {end constructor} {end class} [listing code] maka elmt dan next adalah atribut dari kelas Elemen. Atribut pada sebuah kelas memiliki izin akses jika kelas digunakan oleh kelas lain, izin akses itu seperti private, public, dan protected yang aki' dijelaskan pada subbab-subbab selanjutnya. -Attribut private Izin akses private pada sebuah atribut biasanya digunakan oleh sebuah kelas untuk melindungi atribut-atributnya agar tidak dapal diakses oleh kelas lain. Sebuah atribut yang dinyatakan sebagai private hanya dapat diakses secara langsung oleh kelas yang membungkusnya sedangkan kelas lainnya tidak dapat mengakses attribut ini secara langsung, misalkan kelas sebagai berikut: class Eiemen private NilaiMatKul elmt private Elemen next Elemen()		

	{end constructor} (end class} [listing code] maka yang dapat mengakses elmt dan next hanyalah kelas Elemen sehingga jika sebuah kelas List di dalamnya mempunyai kode sebagai berikut: Elemen e <- new Elemen() [listing code] maka pengaksesan *e . next* tidak diizinkan pada kelas List. Agar isi dari sebuah atribut private dapat diakses oleh kelas lain dapat dibuat sebuah metode yang mengembalikan nilai atribut itu, misalnya sebagai berikut: public getNext()->Elemen -> next {end getNext} [listing code] sehingga kelas lain akan mengakses atribut next pada kelas Elemen dengan kode Elemen n <- e.getNext . -Attribut Public Izin akses public sebuah atribut biasanya digunakan oleh sebuah kelas, jika sebuah atribut diperbolehkan diakses secara langsung oleh kelas lain. Sebuah atribut yang dinyatakan sebagai public dapat diakses secara langsung oleh kelas lain di luar kelas yang membungkusnya, misalkan pada kelas Elemen sebagai berikut: Class Elemen Public NilaiMatKul elmt Element next Elemen() {end class} [listing code] maka atribut elmt dan next dapat diakses secara langsung oleh kehim lain, misalkan dengan kode: Elemen e <- new Elemen() e.next <- NULL [listing code] Jika sebuah atribut tidak ditulis izin aksesnya misalkan hanya Elemen next, maka interpreter Java akan menganggap atribut flex' mempunyai izin akses public.

	-Atribut protected Izin akses protected sebuah atribut biasanya digunakan oleh sebuah kelas, jika sebuah atribut diperbolehkan diakses secara langsung oleh kelas lain yang merupakan kelas turunannya (inheritance). Sebuah atribut yang dinyatakan sebagai protected tidak dapat diakses secant langsung oleh kelas lain diluar kelas yang membungkusnya, kecuali kelas yang mengaksesnya adalah kelas turunan dari kelas yang membungkusnya, misalkan pada kelas Elemen sebagai berikut: protected NilaiMatKul elmt protected Elemen next Elemen() end constructor} {end class} [listing code] maka atribut elmen dan next dapat diakses secara langsung oleh kelas lain yang merupakan turunan kelas Elemen. Akses protected dimaksudkan untuk melindungi atribut agar tidak diakses secara langsung oleh sembarang kelas lain, namun diizinkan diakses secara langsung oleh kelas trunannya.
Informasi Ekstra	
Spesifikasi Grafis	
Catatan	

Kembali ke scene#	2	Selanjutnya scene#	4

Tahap selanjutnya adalah pengenalan konsep konstruktor dan implementasisnya dalam bahasa pemrograman Java. Untuk lebih jelasnya dapat dilihat pada tabel 3.5 sebagai berikut:

Tabel 3.5. *Storyboard* Untuk Materi Konstruktor

Content Screen			
Judul Scene	Konstruktor	scene #	4
Tujuan Pembelajaran	Mengenalkan konsep konsep konstruktor pada pemrograman berorientasi objek		
On Screen Teks	-Konstruktor Sebuah kelas harus memiliki sebuah metode yang disebut sebagai konstruktor. Nama sebuah konstruktor hams sama dengan nama dari sebuah kelas, misalkan kelas Elemen maka konstruktornya adalah ElemenO. Sebuah konstruktor juga dapat menerima sebuah masukan seperti halnya prosedur pada pemrograman prosedural. Fungsi dari sebuah konstruktor adalah:		

	• mengalokasikan sebuah objek saat program dieksekusi • memberikan nilai awal sebagai inisialisasi dari semua atribut yang perlu diinisialisasi. • mengerjakan proses-proses yang diperlukan saat sebuah objek dibuat namun pada kenyataannya sebuah konstruktor dapat tidak berisi apa-apa, hal ini jika memang tidak diperlukan adanya inisialisasi atau proses yang dikerjakan ketika sebuah objek dibuat. Konstruktor harus bersifat public karena sebuah konstruktor akan diakses oleh kelas lain untuk membuat objek suatu kelas. Sebuah kelas dapat memiliki konstruktor lebih dari satu. Pada saat eksekusi program, compiler atau interpreter akan mencari konstruktor mana yang sesuai dengan konstruktor yang dipanggil, hal ini disebut sebagai overloading, misalkan kelas sebagai berikut: class Tititk private integer x private integer y Titik() x <- 0 y <- 0 {end Titik} Titik(integer xp, integer yp) x<- xp y <- YP {end Titik} [listing code] maka ketika objek Titik dibuat dengan kode berikut: Titik t <- new Titik() [listing code] konstruktor yang akan dipanggil adalah Titik() , namun jika objek Titik dibuat dengan kode berikut: Titik t <- new Titik(5, 7) [listing code] maka konstruktor yang dipanggil adalah Titik (input : integer xp, integer yp)
Informasi Ekstra	

Spesifikasi Grafis			
Catatan			
Kembali ke scene#	3	Selanjutnya scene#	5

Tahap selanjutnya adalah pengenalan konsep destruktor dan implementasisnya dalam bahasa pemrograman Java. Untuk lebih jelasnya dapat dilihat pada tabel 3.6 sebagai berikut:

Tabel 3.6. *Storyboard* Untuk Materi Destruktor

Content Screen			
Judul Scene	Destruktor	scene #	5
Tujuan Pembelajaran	Mengenalkan konsep konsep destruktor dalam bahasa pemrograman berorientasi objek		
On Screen Teks	-Destruktor Destruktor adalah metode yang dipanggil secara otomatis ketika objek dihancurkan. Sebuah destruktor tidak harus ada pada kode program sebuah kelas. Penamaan sebuah destruktor sama dengan konstruktor hanya saja pada destruktor diberi tanda ~ pada bagian depan penamaan metode, misalkan sebagai berikut: class Titik private integer x private integer y Titik() x <- 0 y <- 0 {end Titik} ~Titik() {end –Titik} {end class} [listing code] Metode Titik () adalah konstruktor dan metode Titik O adalah destruktor. Sebuah destruktor biasanya hanya terdapat satu metode pada sebuah kelas clan tidak memiliki parameter masukan. Destruktor biasanya berisi proses-proses finalisasi dari suatu kelas, misalnya menghapus alokasi memori yang telah dipakai oleh objek. Pada bahasa pemrograman C++ destruktor dipanggil secara otomatis saat sebuah objek dihancurkan, tapi pada bahasa pemrograman Java terdapat metode finalize() yang dapat difungsikan seperti destruktor namun metode ini tidak harus diimplementasikan pada sebuah kelas pada bahasa pemrograman Java. Pada bahasa pemrograman Java objek dibebaskan secara otomatis oleh bagian dari runtime Java yang disebut dengan		

	garbage collector yang berjalan secara background yaitu Java akan melakukan pengecekan secara bertahap dan menghapus objek-objek yang sudah tidak diacu lagi.		
Informasi Ekstra			
Spesifikasi Grafis			
Catatan			
Kembali ke scene#	4	Selanjutnya scene#	6

Tahap selanjutnya adalah pengenalan konsep method dan implementasisnya dalam bahasa pemrograman Java. Untuk lebih jelasnya dapat dilihat pada tabel 3.7 sebagai berikut:

Tabel 3.7. *Storyboard* Untuk Materi Method

Content Screen			
Judul Scene	Method	scene #	6
Tujuan Pembelajaran	Mengenalkan konsep konsep method pada bahasa pemrograman berorientasi objek		
On Screen Teks	-Metode (Method) Metode atau method pada sebuah kelas hampir sama dengan fungsi atau prosedur pada pemrograman prosedural. Pada sebuah metode di dalam sebuah kelas juga memiliki izin akses seperti halnya atribut pada kelas, izin akses itu antara lain private, public, dan protected yang memiliki arti sama pada izin akses atribut yang telah dibahas sebelumnya. Sebuah metode yang mengembalikan sebuah nilai ditulis seperti halnya sebuah fungsi misalnya sebagai berikut: public getNext() -> Elemen -> next {end getNext} dan metode yang tidak mengembalikan sebuah nilai ditulis seperti halnya sebuah prosedur misalnya sebagai berikut: public setNext(Elemen e) next <- e {end setNext} Sebuah kelas boleh memiliki lebih dari satu metode dengan nama yang sama (oveloading) asalkan memiliki parameter masukan yang berbeda sehingga compiler atau interpreter dapat mengenali metode mana yang dipanggil. Di dalam sebuah kelas, terdapat juga yang disebut sebagai metoill. atau atribut statis yang memiliki kata kunci static. Maksud dari statis		

	di sini adalah metode yang dapat diakses secara berbagi (share) dengan semua objek lain tanpa hams membuat objek yang memilikl metode statis tadi (tanpa proses new), tapi sebuah metode statik mempunyai keterbatasan yaitu hanya dapat mengakses atribut atau metode lain di dalam kelas yang membungkusnya yang juga bersifat statis. Metode statis biasanya diimplementasikan untuk metode main.		
Informasi Ekstra			
Spesifikasi Grafis			
Catatan			
Kembali ke scene#	5	Selanjutnya scene#	7

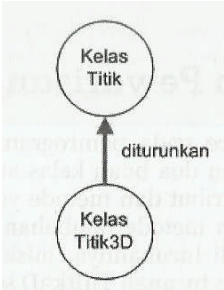
Tahap selanjutnya adalah pengenalan konsep pewarisan dalam bahasa pemrograman Java. Untuk lebih jelasnya dapat dilihat pada tabel 9.8 sebagai berikut:

Tabel 3.8. *Storyboard* Untuk Materi Pewarisan/Inheritance

Content Screen			
Judul Scene	Pengenalan PBO	scene #	8
Tujuan Pembelajaran	Mengenalkan konsep konsep inheritance pada bahasa pemrograman berorientasi objek		
On Screen Content	-Pewarisan Pewarisan atau inheritance pada pemrograman berorientasi objek merupakan suatu hubungan dua buah kelas atau lebih dalam hal ini ada kelas yang memiliki atribut dan metode yang sama dengan kelas iainnya beserta atribut dan metode tambahan yang merupakan sifat khusus kelas yang menjadi turunannya, misalkan ada sebuah kelas Titik yang mempunyai kelas turunan Tititk3D sebagai berikut: Class Titik private integer x private integer y titik() x<-0 y<-0 {end Titik} public getX()->integer ->x Public getY()->integer ->y {end class} class Titik3D : Titik		

private integer z
Titik3D()
Z <- 0
{end Titik3D}
public getZ()->intr z
{end getZ)
{end class)

Secara ilustrasi keterkaitan kelas Titik dan Titik3D dapat dilihat pada gambar dibawah



Keterkaitan antara kelas Titik dan Titik3D adalah kelas Titik3D merupakan kelas turunan dari kelas Titik, dalam hal ini kelas Titik disebut dengan kelas dasar atau super class atau base class sedangkan kelas Tititk3D disebut sebagai kelas turunan atau derived class atau subclass.
Pada contoh di atas, ketika kelas Titik3D dibuat objeknya maka objek tersebut dapat menggunkan metode yang ada pada kelas Titik walau berikut:
pada kode programnnya metode itu tidak dituliskan, misalkan sebagai

```
Titik3D p      <-    new Titik3D()
integer x      <-    p.getX()
integer y      <-    p.getY()
integer z      <-    p.getZ()
```

Keuntungan dari pewarisan adalah tidak perlu mengutak atik kode kelas yang membutuhkan tambahan atribut atau metode saja karena tinggal membuat kelas turunannya tanpa harus mengubah kode kelas dasarnya.

Kelas dasar akan mewariskan semua atribut dan metodenya kecuali kosntruktor dan destruktor yang memiliki izin akses public dan protected ke kelas turunannya dengan izin akses yang sama dengan pada kelas dasar.

Ketika sebuah kelas turunan dibuat objeknya saat eksekusi, maka secara implisit konstruktor kelas dasar dipanggil terlebih dahulu baru kemudian konstruktor kelas turunan dijalankan. Begitu juga saat objek dimusnahkan maka secara destruktror kelas turunan

	akan dijalankan baru kemudian destruktor kelas dasar dijalankan. Kelas turunan juga dapat mengimplementasikan metode yang ada pada kelas dasar, maka pada saat eksekusi, yang dipanggil adalah metode yang ada pada kelas turunan, misalnya sebagai berikut: class Titik3D : Titik private integer z Titik3D () z <- 0 {end Titik3D} public getX()->integer -> 0 {end getX} public getZ() -› integer z {end getZ} {end class} maka saat metode getX() dijalankan dengan kode: Titik3D p <- new Titik3D() integer x <- n.getX() variabel x akan berisi nilai o karena yang dijalankan adalah metode getX() yang ada pada kelas Titik3D.
Informasi Ekstra	
Spesifikasi Grafis	
Catatan	

Kembali ke scene#	7	Selanjutnya scene#	9

Tahap selanjutnya adalah implementasi dari konsep pewarisan dalam bahasa pemrograman Java. Untuk lebih jelasnya dapat dilihat pada tabel 3.9 sebagai berikut:

Tabel 3.9. *Storyboard* Untuk Materi Implementasi Pewarisan

Content Screen			
Judul Scene	Implementasi pewarisan	scene #	9
Tujuan Pembelajaran	Mengenalkan konsep konsep inheritance pada bahasa pemrograman berorientasi objek		
On Screen Content	-Berikut ini dalah contoh implementasi pewarisan pada bahasa pemrigraman java class Titik{ /*kelas yang digunakan untuk mengimpelementasikan sebuah tipe titik*/		

```
  private int x; /*koordinat x*/
  private int y; /*koordinat y*/


  Titik(){
    /*konstruktor*/
    x = 0;
    y = 0;
  }

  Titik(int xp, int yp){
    /*konstruktor*/
    x = xp;
    y = yp;
  }

  public void setX(int xp){
    /*mengeset nilai koordinat x*/
    x = xp;
  }

  public int getX(){
  /*mengembalikan nilai koordinat x*/
    return x;
  }

  public void setY(int yp){
    /*mengeset nilai koordinat y*/
    y = yp;
  }

  public int getY(){
  /*mengembalikan nilai koordinat y*/
    return y;
  }

  public void finalize(){
    /*destruktor*/

  }

}

class Titik3D extends Titik{
```

```
  /*kelas turunan kelas Titik*/

  private int z; /*koordinat z*/

  Titik3D(){
   /*konstruktor*/
   z = 0;
  }

  Titik3D(int xp, int yp, int zp){

   /*konstruktor*/
   setX(xp);
   setY(yp);
   z = zp;
  }

  public void setZ(int zp){
   /*mengeset nilai koordinat z*/
   z = zp;
  }

  public int getZ(){
  /*mengembalikan nilai koordinat z*/
   return z;
  }

}

class CobaTitik3D{
/*metode main untuk mengetes kelas Titik dan kelas Titik3D*/

  public static void main(String[] args) {
   Titik3D t = new Titik3D(0, 0, 7);

   t.setX(28);
   t.setY(1);

   System.out.println("t : nilai X : " + t.getX());
   System.out.println("t : nilai Y : " + t.getY());
   System.out.println("t : nilai Z : " + t.getZ());

  }

}
```

Setelah di runing maka hasil dari program adalah:

	T: nilai X : 28 T: nilai Y : 1 T: nilai Z : 7		
Informasi Ekstra			
Spesifikasi Grafis			
Catatan			
Kembali ke scene#	8	Selanjutnya scene#	10

Tahap selanjutnya adalah pengenalan konsep polimorfisme dan implementasisnya dalam bahasa pemrograman Java. Untuk lebih jelasnya dapat dilihat pada tabel 3.10 sebagai berikut:

Tabel 3.10. *Storyboard* Untuk Materi Polimorfisme

Content Screen			
Judul Scene	Polimorfisme	scene #	10
Tujuan Pembelajaran	Mengenalkan konsep konsep inheritance pada bahasa pemrograman berorientasi objek		
On Screen Content	-Pengertian Polimorfisme Polimorfisme berarti banyak bentuk. Pada pemrograman berorientasi objek, polimorfisme merupakan konsep yang menyatakan sesuatu yang sama dapat memiliki berbagai bentuk dan perilaku yang berbeda. Overloading merupakan bentuk polimorfisme yaitu beberapa metode dapat memiliki nama yang sama dengan isi dan parameter yang berbeda. Bentuk polimorfisme yang lain adalah dengan menggunakan metode virtual yakni sebuah kelas dapat memiliki sebuah metode virtual sebagai antarmuka dari metode yang diimplementasikan kelas turunannya. Misalkan sebagai berikut: class Tititk private integer x private integer y Titik() x <- 3 y <- 3 {end Titik} virtual public integer getX() -> x {end getX} virtual public integer getY() -> y {end getY} {end class}		

```
class Titik3D : Titik
private integer z
Titik3D()
z <-
{end Titik3D}
public getX() ->integer
->0
{end getX} -> integer
public getZ()-> integer
-> z
{end getZ}
{end class}
{algoritma utama}
Titik pl, p0;
'p0 <- new Titik()
pi <- PO
integer x <- pt.getX()
Titik3D p1;
pl <- new Titik3D()
Pi <- pl
integer xl <- pi.getX()
{end algoritma utama}
```

dari kode program utama di atas variabel x akan berisi nilai 3 dan variabel xi akan berisi nilai o. Sebuah kelas yang berisi minimal satu metode virtual disebut sebagai kelas abstrak.

-Implementasi polimorfisme

```
class Titik{

/*kelas yang digunakan untuk mengimpelementasikan sebuah tipe titik*/


  private int x; /*koordinat x*/
  private int y; /*koordinat y*/


  Titik(){
    /*konstruktor*/
    x = 0;
    y = 0;
  }

  Titik(int xp, int yp){
```

```
  /*konstruktor*/
  x = xp;
  y = yp;
 }

 public void setX(int xp){
  /*mengeset nilai koordinat x*/
  x = xp;
 }

 public int getX(){
 /*mengembalikan nilai koordinat x*/
  return x;
 }

 public void setY(int yp){
  /*mengeset nilai koordinat y*/
  y = yp;
 }

 public int getY(){
 /*mengembalikan nilai koordinat y*/
  return y;
 }

 public void printTitik(){
  /*menampilkan nilai koordinat titik*/
  System.out.println("nilai X : " + getX());
  System.out.println("nilai Y : " + getY());
 }

 public void finalize(){
  /*destruktor*/

 }

}

class Titik3D extends Titik{

 /*kelas turunan kelas Titik*/

 private int z; /*koordinat z*/

 Titik3D(){
  /*konstruktor*/
  z = 0;
```

```
  }

  Titik3D(int xp, int yp, int zp){

   /*konstruktor*/
   setX(xp);
   setY(yp);
   z = zp;
  }

  public void setZ(int zp){
   /*mengeset nilai koordinat z*/
   z = zp;
  }

  public int getZ(){
  /*mengembalikan nilai koordinat z*/
   return z;
  }

  public final void printTitik(){
   /*menampilkan nilai koordinat titik*/
   System.out.println("nilai X : " + getX());
   System.out.println("nilai Y : " + getY());
   System.out.println("nilai Z : " + getZ());
  }

}

class CobaTitik3DP{
/*metode main untuk mengetes kelas Titik dan kelas Titik3D*/

  public static void main(String[] args) {
   Titik3D t = new Titik3D(0, 0, 7);

   t.setX(28);
   t.setY(1);

   t.printTitik();

   Titik t1;
   System.out.println("====================");
   t1 = t;

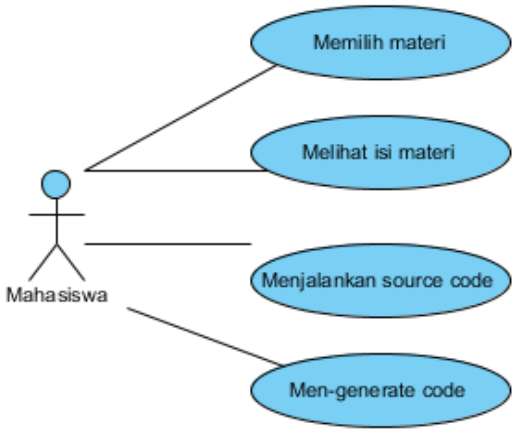
   t1.printTitik();

  }
```

	}		
Informasi Ekstra			
Spesifikasi Grafis			
Catatan			
Kembali ke scene#	9	Selanjutnya scene#	

3.2.3.3. *Use Case Diagram*

Langkah selanjutnya yang dilakukan dalam analisis kebutuhan adalah kebutuhan fungsional untuk pembangunan *e-learning* untuk pemrograman berorientasi objek, analisis kebutuhan fungsional untuk sistem yang akan dibangun ini digambarkan seperti dalam diagram *Use Case* seperti gambar 3.2 dibawah ini:



Gambar 3.2. *Use Case Diagram e-Learning*

Gambar 3.1 menjelaskan bahwa mahasiswa sebagai pengguna aplikasi berinteraksi dengan sistem melalui wen browser.

3.2.3.4. Identifikasi Aktor

Identifikasi aktor adalah identifikasi tentang aktor yang berperan dalam menjalankan sistem, aktor yang berperan dapat dilihat pada Tabel 3.11berikut ini:

Tabel 3.11. Identifikasi Aktor

No.	Aktor	Deskripsi
A01	Mahasiswa	Merupakan aktor yang berperan dalam menggunakan aplikasi *e-learning* untuk pemrograman berorientasi objek ini, mahasiswa berperan memilih materi, melihat isi materi, menjalankan source code, men-*generate code*.

3.2.3.5. Identifikasi *Use Case*

Berikut ini dijelaskan mengenai identifikasi *Use Case* yang yang digunakan dalam aplikasi *e-learning* untuk pemrograman berorientasi objek seperti pada tabel 3.12 berikut ini:

Tabel 3.12. Identifikasi *Use Case*

No	Nama use case	Deskripsi
UC01	Memilih materi	*Use case* ini memproses informasi tentang daftar materi yang ada didalam sistem berupa menu atau sekumpulan link yang bisa dipilih oleh mahasiswa.
UC03	Melihat isi materi	*Use case* ini memproses informasi tentang isi materi yang dipilih mahasiswa.
UC03	Menjalankan *source code*	*Use case* ini memproses contoh *source code* yang ada di dalam materi agar dapat di olah dan di jalankan untuk melihat hasil dari eksekusi *source code.*
UC04	Men-*generate* kelas	*Use case* ini memproses *input* dari mahasiswa berupa nama kelas, atribut dan metode dan menampilkan informasi kelas yang kerangka sebuah kelas berdasarkan input dari mahasiswa tersebut

3.2.3.6. Skenario *Use Case*

Berikut ini adalah skenario *use case* dari masing-masing *use case* yang telah dianalisis.

Hasil analisis skenario *use case* memilih materi dapat dilihat pada tabel 3.13 dibawah ini:

Tabel 3.13. Skenario *Use Case* Memilih Materi

Identifikasi	
Nama *Use Case*	Memilih Materi
Aktor	Mahasiswa
Tujuan	Mahasiswa mendapatkan informasi berupa daftar materi yang ada didalam sistem berupa menu atau sekumpulan *link* yang bisa dipilih oleh mahasiswa
Deskripsi	*Use case* ini memproses informasi tentang daftar materi yang ada didalam sistem berupa menu atau sekumpulan link yang bisa dipilih oleh mahasiswa.
Skenario Utama (UC01-A01)	
Kondisi awal	Tampilan awal setelah program dijalankan
Aksi Aktor	**Reaksi Sistem**
1. Aktor membuka aplikasi	2. Sistem menampilkan daftar materi yang ada didalam sistem berupa menu atau sekumpulan link yang bisa dipilih oleh mahasiswa.
Skenario Alternatif	
Aksi Aktor	**Reaksi Sistem**
Kondisi akhir	Sistem menampilkan daftar materi yang ada didalam sistem berupa menu atau sekumpulan link yang bisa dipilih oleh mahasiswa.

Hasil analisis skenario *use case* melihat isi materi dapat dilihat pada tabel 3.14 dibawah ini:

Tabel 3.14. Skenario *Use Case* Melihat Isi Materi

Identifikasi	
Nama *Use Case*	Melihat Isi Materi
Aktor	Mahasiswa
Tujuan	Mahasiswa mendapatkan informasi berupa tentang isi materi yang dipilih mahasiswa.
Deskripsi	*Use case* ini memproses informasi tentang isi materi yang

	dipilih mahasiswa.
Skenario Utama (UC02-A01)	
Kondisi awal	Tampilan awal setelah program dijalankan
Aksi Aktor	**Reaksi Sistem**
1. Aktor memilih salah satu materi yang terdapat dalam menu materi	2. Sistem menampilkan isi dari dari materi yang telah dipilih oleh mahasiswa.
Skenario Alternatif	
Aksi Aktor	**Reaksi Sistem**
Kondisi akhir	Sistem menampilkan informasi berupa isi dari dari materi yang telah dipilih oleh mahasiswa.

Hasil analisis skenario *use case* menjalankan *source code* dapat dilihat pada tabel 3.15 dibawah ini:

Tabel 3.15. Skenario *Use Case* Menjalankan *Source Code*

Identifikasi	
Nama *Use Case*	Menjalankan *Source Code*
Aktor	Mahasiswa
Tujuan	Mahasiswa memproses contoh *source code* yang ada di dalam materi agar dapat di olah dan di jalankan untuk melihat hasil dari eksekusi *source code*
Deskripsi	*Use case* ini memproses contoh *source code* yang ada di dalam materi agar dapat di olah dan di jalankan untuk melihat hasil dari eksekusi *source code.*
Skenario Utama (UC03-A01)	
Kondisi awal	Tampilan awal setelah program dijalankan
Aksi Aktor	**Reaksi Sistem**
1. Aktor mengklik tombol “Coba Source Code” yang ada di bawah contoh *source code*	2. Sistem mengarahkan mahasiswa menuju website rextexter.com yang berisi contoh *source code* yang terkait dengan materi
Skenario Alternatif	
Aksi Aktor	**Reaksi Sistem**
Kondisi akhir	Sistem mengarahkan mahasiswa menuju website rextexter.com yang berisi contoh *source code* yang terkait dengan materi

Hasil analisis skenario *use case* men-*generate code* dapat dilihat pada tabel 3.16 dibawah ini:

Tabel 3.16. Skenario *Use Case* Men-*Generate Code*

Identifikasi	
Nama *Use Case*	Men-*generate* kelas
Aktor	Mahasiswa
Tujuan	Mahasiswa memproses *input* dari mahasiswa berupa nama kelas, atribut dan metode dan menampilkan informasi kelas yang kerangka sebuah *class* berdasarkan input dari mahasiswa tersebut
Deskripsi	*Use case* ini memproses *input* dari mahasiswa berupa nama kelas, atribut dan metode dan menampilkan informasi kelas yang kerangka sebuah kelas berdasarkan input dari mahasiswa tersebut

Skenario Utama (UC04-A01)	
Kondisi awal	Tampilan awal setelah program dijalankan

Aksi Aktor	**Reaksi Sistem**
	1. Sistem menampilkan form kelas
2. Aktor mengisi form kelas	
3. Aktor mengklik tombol “Buat Kelas”	4. Sistem menampilkan kerangka kelas berdasarkan isi form yang telah diisi oleh aktor.

Skenario Alternatif	
Aksi Aktor	**Reaksi Sistem**

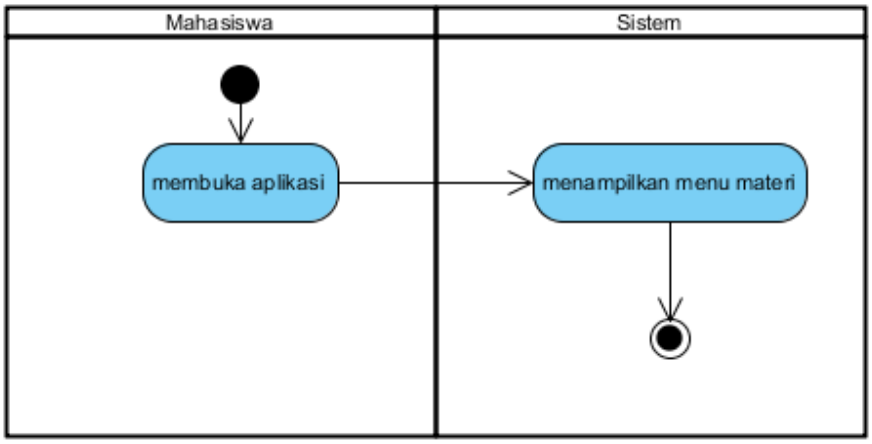
Kondisi akhir	Sistem menampilkan kerangka kelas berdasarkan isi form yang telah diisi oleh aktor.

3.2.3.7. *Activity Diagram*

Activity diagram adalah teknik untuk mendeskripsikan logika prosedural dan alur kerja program. *Activity diagram* mempunyai peran seperti halnya *flowchart* akan tetapi perbedaannya dengan *flowchart* adalah *activity diagram* bisa mendukung perilaku pararel sedangkan *flowchart* tidak bisa.

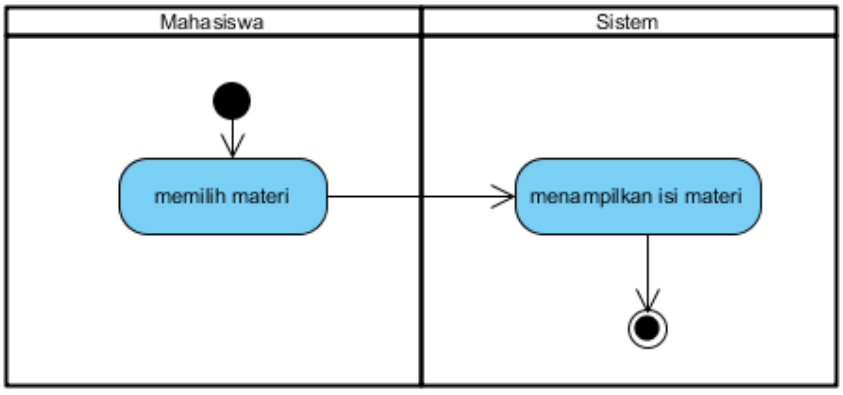
Activity diagram yang digunakan dalam aplikasi *e-learning* ini dapat dilihat pada gambar-gambar dibawah ini:

Activity diagram memilih materi dapat dilihat pada gambar 3.3 dibawah ini:



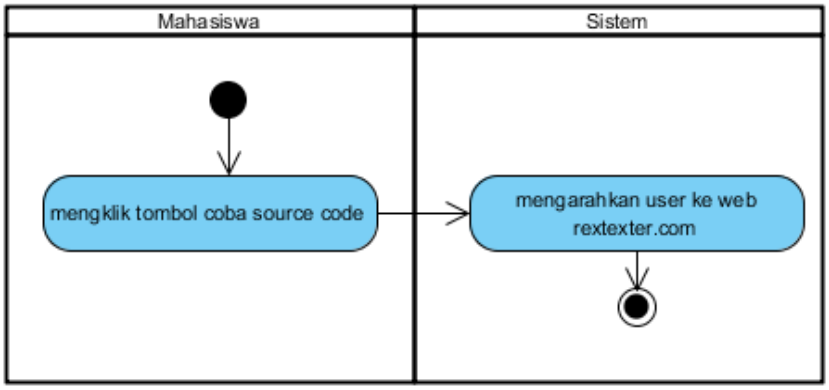
Gambar 3.3. *Activity Diagram* Memilih Materi

Activity diagram menampilkan isi materi dapat dilihat pada gambar 3.4 dibawah ini:



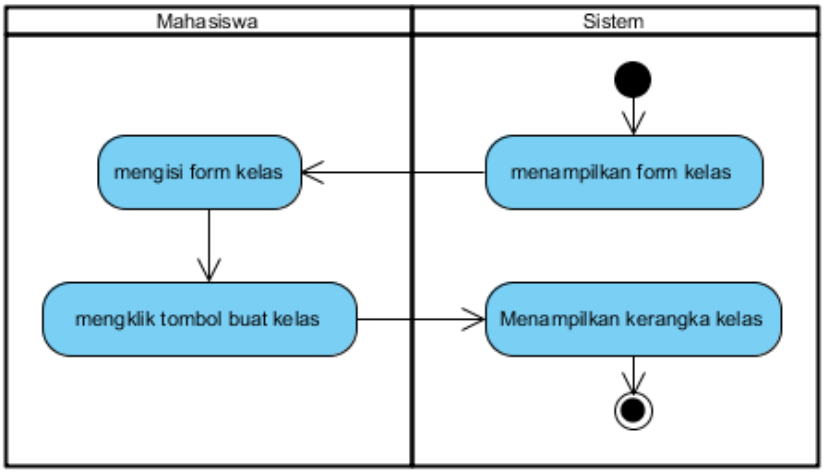
Gambar 3.4. *Activity Diagram* Menampilkan Isi Materi

Activity diagram menjalankan *source code* dapat dilihat pada gambar 3.5 dibawah ini:



Gambar 3.5. *Activity Diagram* Menjalankan *Source Code*

Activity diagram men-*generate* kelas dapat dilihat pada gambar 3.6 dibawah ini:

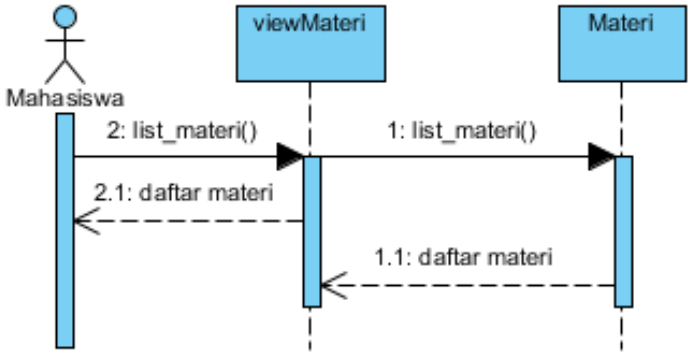


Gambar 3.6. *Activity Diagram* Men-*generate* Kelas

3.2.3.8. *Sequence Diagram*

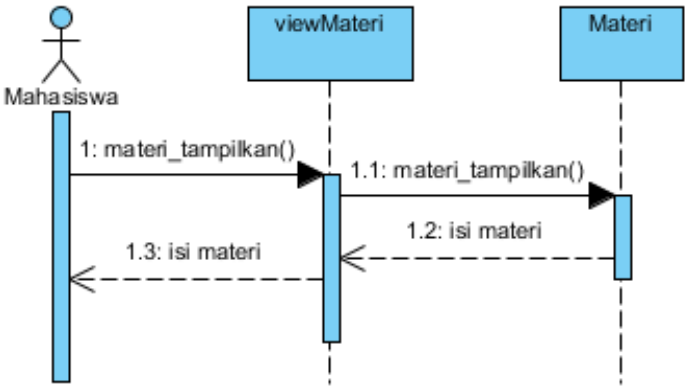
Diagram *sequence* atau *sequence* diagram merupakan gambaran kelakuan objek pada use case dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek

Sequence diagram pilih materi dapat dilihat pada gambar 3.7 dibawah ini:



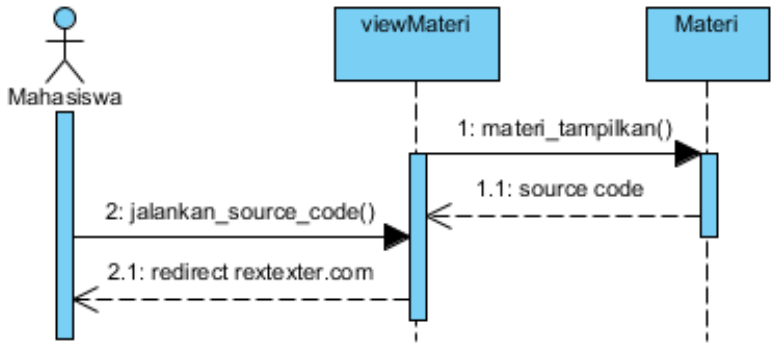
Gambar 3.7. *Sequence Diagram* Pilih Materi

Sequence diagram lihat isi materi dapat dilihat pada gambar 3.8 dibawah ini:



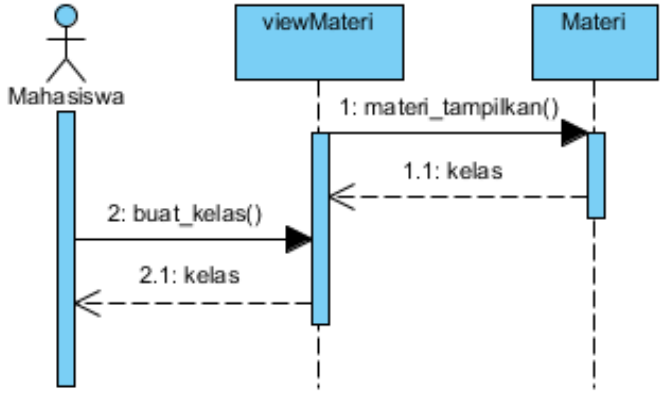
Gambar 3.8. *Sequence Diagram* Lihat Isi Materi

Sequence diagram menjalankan *source code* dapat dilihat pada gambar 3.9 dibawah ini:



Gambar 3.9. *Sequence Diagram* Menjalankan *Source Code*

Sequence diagram men-*generate* kelas dapat dilihat pada gambar 3.10 dibawah ini:

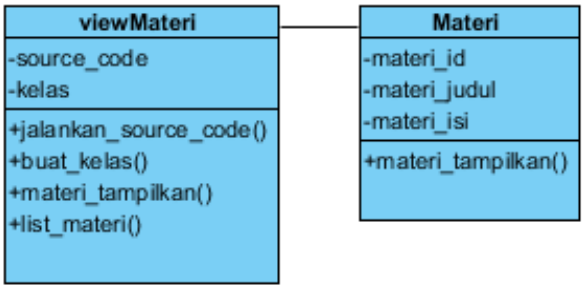


Gambar 3.10. *Sequence Diagram* Men-*generate* Kelas

3.2.3.9. *Class Diagram*

Class diagram adalah spesifikasi objek, yang memiliki atribut/properti dan layanan / fungsional (metode/fungsi).

Class diagram yang digunakan dalam aplikasi e-learning dapat dilihat pada gambar 3.11 dibawah ini



Gambar 3.11. *Class Diagram*

3.3. Perancangan Sistem

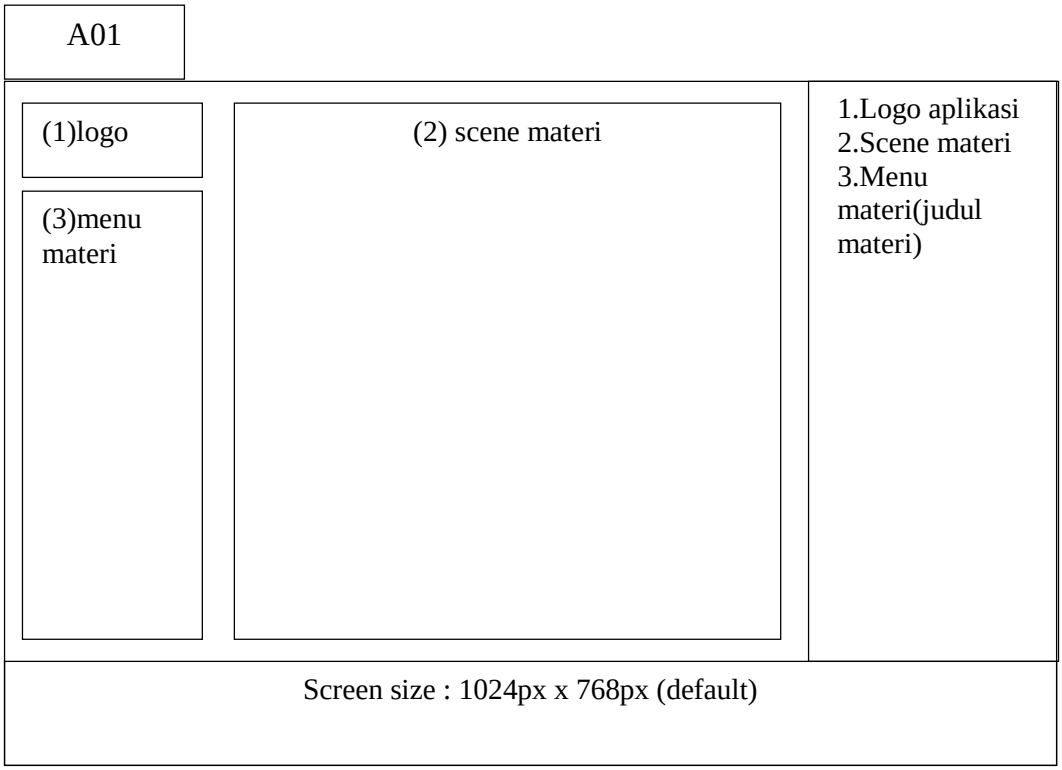
Perancangan merupakan penggambaran, perencanaan dan pembuatan sketsa atau pengaturan dari beberapa elemen yang terpisah ke dalam suatu kesatuan yang utuh. Tahapan ini meliputi konfigurasi perangkat lunak dan perangkat keras dari suatu sistem. Tahap perancangan desain bertujuan untuk mencari bentuk yang optimal dari aplikasi yang akan dibangun dengan pertimbangan faktor-faktor permasalahan dan kebutuhan yang ada pada sistem seperti yang telah ditetapkan dengan cara mengkombinasikan penggunaan teknologi perangkat keras dan perangkat lunak yang tepat sehingga diperoleh yang optimal dan mudah diimplementasikan. Adapun perancangan sistem dari aplikasi *e-learning* untuk bahasa pemrograman berorientasi objek ini adalah sebagai berikut:

3.3.1. Perancangan Antar Muka Perangkat Lunak

Perancangan antarmuka merupakan sebuah penggambaran, perencanaan dan pembuatan sketsa atau pengaturan dari beberapa elemen yang terpisah ke dalam satu kesatuan yang utuh dan berfungsi.

3.3.1.1. Perancangan Antar Muka Materi

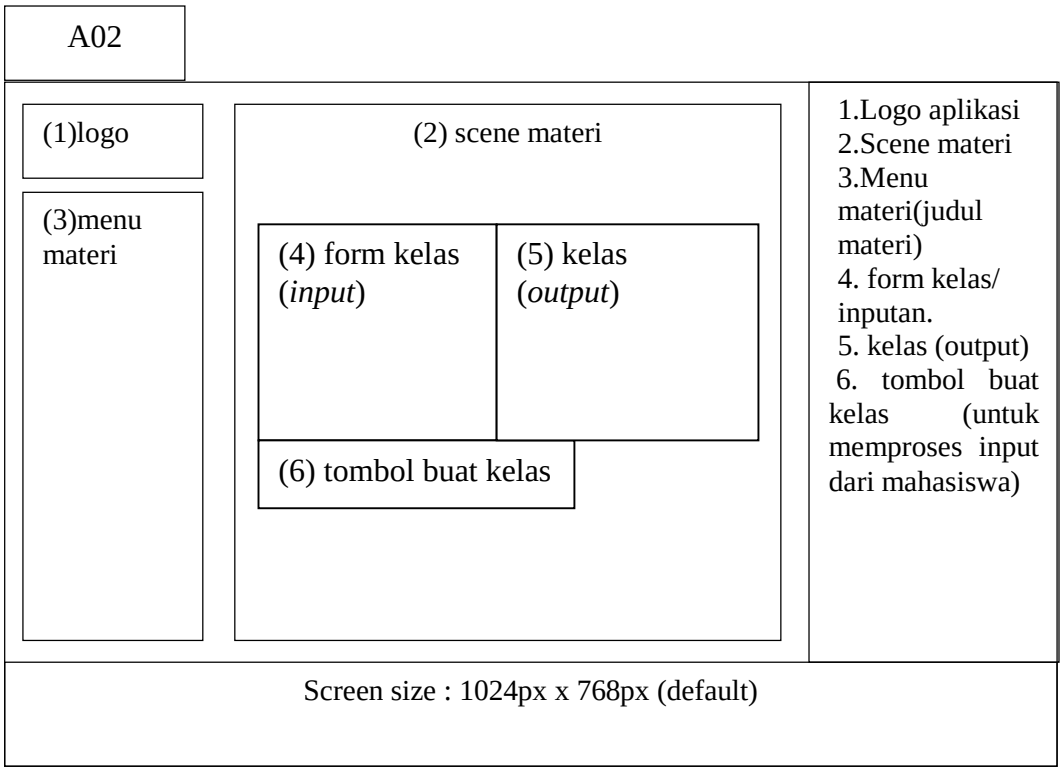
Perancangan antar muka yang akan dibuat untuk manampilkan materi pembelajaran pemrograman berorientasi objek dapat dilihat pada gambar 3.12 dibawah ini:



Gambar 3.12. Perancangan Antar Muka Materi

3.3.1.2. Perancangan Antar Muka *Source Code*

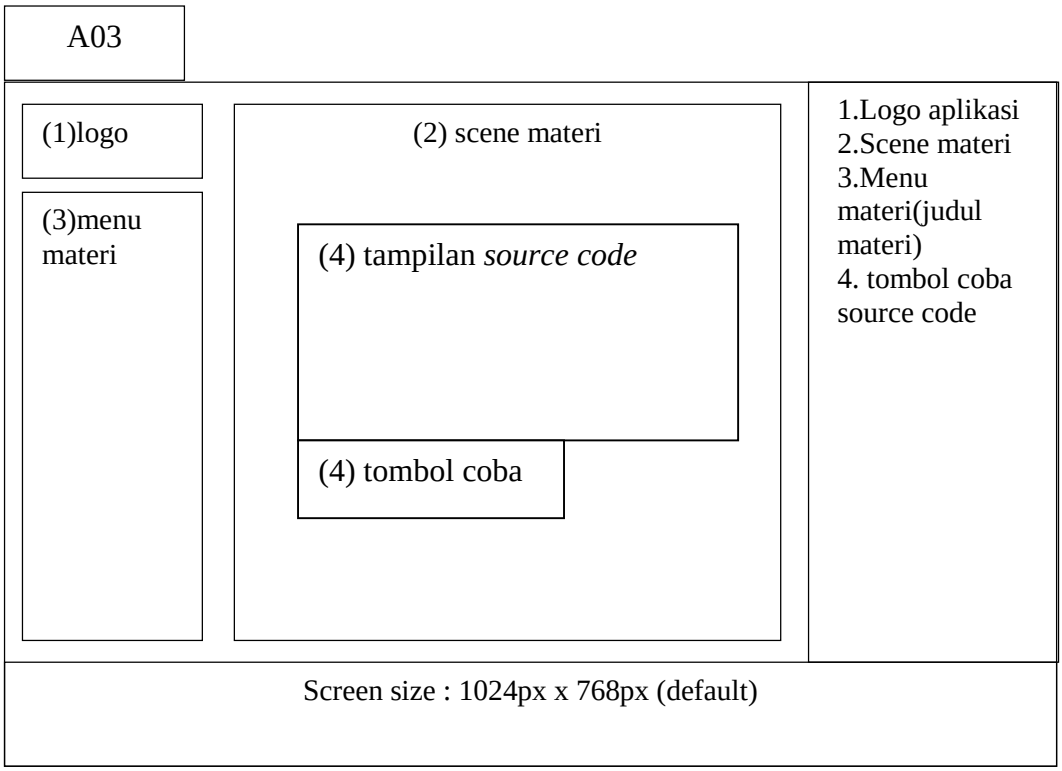
Perancangan antar muka yang akan dibuat untuk manampilkan *source code* materi pembelajaran pemrograman berorientasi objek dapat dilihat pada gambar 3.13 dibawah ini:



Gambar 3.13. Perancangan Antar Muka *Source Code*

3.3.1.3. Perancangan Antar Muka Men-g*enerate* Source Code

Perancangan antar muka yang akan dibuat untuk manampilkan *source code* materi pembelajaran pemrograman berorientasi objek dapat dilihat pada gambar 3.14 dibawah ini:



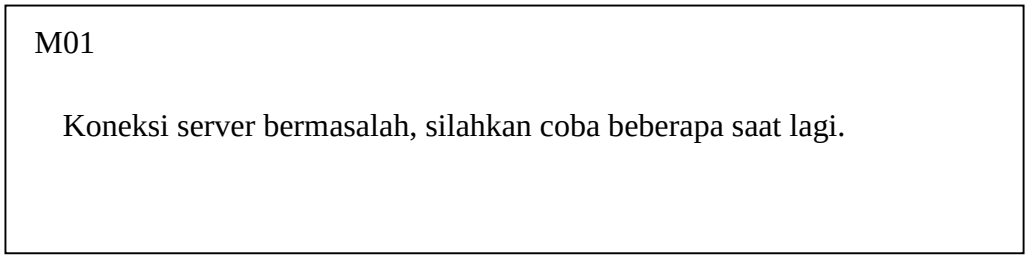
Gambar 3.14. Perancangan Antar Muka Men-*generate* Kelas

3.3.1.4. Perancangan Antar Muka Pesan

Pada subbab ini akan dibahas perancangan pesan yang akan muncul dalam aplikasi yang akan dibangun ini:

M01: Pesan koneksi server gagal

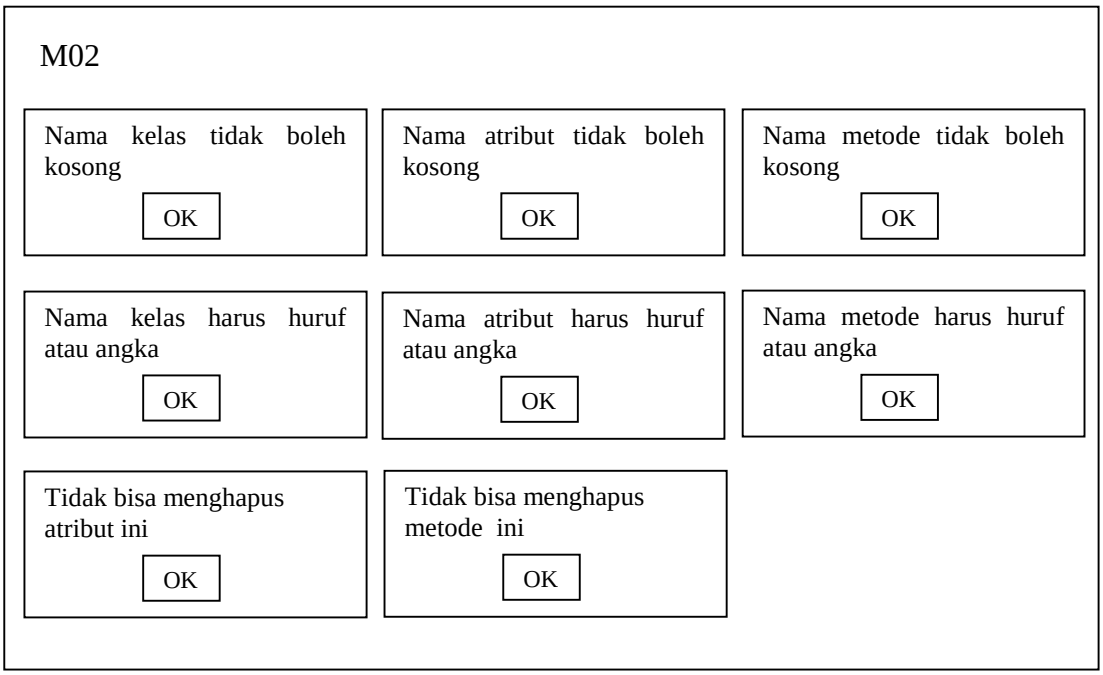
Form ini menampilkan pesan ketika aplikasi gagal melakukan koneksi ke server ideone.com sebagai penyedia *Application Programing Interface* (API).



Gambar 3.15. Perancangan Pesan Koneksi Server Gagal

M02: Pesan validasi *code generator*

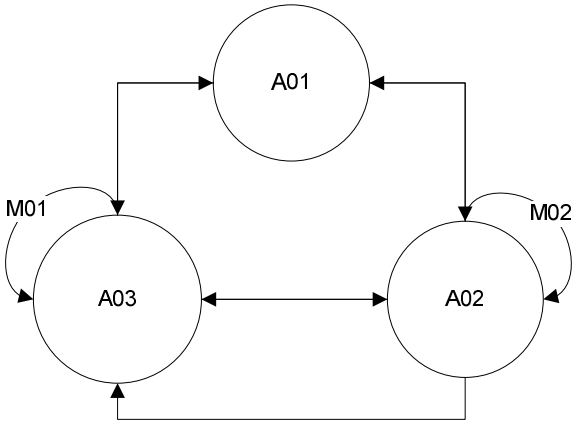
Form ini menampilkan pesan ketika nama kelas, nama atribut dan nama metode dikosongkan, pesan juga ditampilkan ketika field-field tersebut diatas diisi dengan karakter khusus seperti: <>@!#$%^&*()_+[]{}?:;|'\"\\,./~`-=



Gambar 3.16. Perancangan Pesan Validasi *Code Generator*

3.3.2. Jaringan Semantik

Jaringan Semantik merupakan gambaran pengetahuan grafis yang menunjukkan hubungan antar objek (lingkaran) dan garis yang menggambarkan formasi antar objek tersebut. Adapun jaringan semantik untuk aplikasi ini dapat dilihat pada Gambar 3.15 dibawah ini:



Gambar 3.17. Jaringan Semantik