BAB II

LANDASAN TEORI

2.1 Perancangan

Definisi perancangan menurut Azhar Susanto dalam bukunya yang berjudul*Sistem Informasi Manajemen Konsep dan Pengembangannya* adalah sebagaiberikut: “perancangan adalah spesifikasi umum dan terinci dari pemecahanmasalah berbasis komputer yang telah dipilih selama tahap analisis”(2004:332).

Definisi menurut John Burch dan Gary Grudnitski yang telah diterjemahkan olehJogiyanto HM dalam bukunya yang berjudul *Analisis dan Desain SistemInformasi* menyebutkan bahwa: ”desain sistem adalah penggambaran,perencanaan dan pembuatan sketsa atau pengaturan dari beberapa elemen yangterpisah dari suatu kesatuan yang utuh dan berfungsi.”(2005:196)

Berdasarkan dua definisi perancangan di atas maka penulis dapatmenyimpulkan bahwa perancangan adalah penggambaran, perencanaan danpembuatan sketsa yang kemudian diterjemahkan ke dalam sebuah konseprancangan sebagai pemecahan masalah berbasis komputer yang telah dipilihselama tahap analisis.

2.2 Berat Badan Ideal

Berat badan ideal sering menjadi dambaan setiap orang. Berat badan ideal adalah di mana berat badan seseorang berada pada posisi sesuai dengan tinggi badannya. Apakah Anda sudah memiliki berat badan yang ideal. Berikut cara menghitung berat badan ideal Anda.

Dari jaman dahulu sampai jaman sekarang rumus yang paling banyak digunakan untuk menghitung berat badan ideal adalah menggunakan rumus broca, seorang yang pada jamannya ahli pada bidang antropometri yaitu ilmu yang berkaitan dengan mengukur manusia, termasuk di dalamnya cara menghitung berat badan ideal.

Adapun cara untuk mengetahui Perhitungan Berat Ideal Konvensional adalah sebagai berikut :

a. Berat Badan Ideal Bayi (Anak 0-12 bulan)

   Rumus = (umur (bln) / 2) + 4

b. Berat Badan Ideal untuk Anak (1-12 tahun)

   Rumus = (umur (thn) x 2) + 8

c. Remaja dan dewasa

   - Berat Ideal untuk Pria Dewasa

     (Tinggi Badan – 100) – (10% (Tinggi Badan – 100)

   - Berat Ideal untuk Wanita Dewasa

     (Tinggi Badan – 100) – (15% (Tinggi Badan – 100)

2.3 Android

Android adalah sebuah sistem operasi perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware* dan aplikasi. Android menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka.

Android merupakan generasi baru *platform mobile* yang memberikan kesempatan kepada pengembang untuk melakukan pengembangan sesuai dengan

yang diharapkan. Sistem operasi yang mendasari Android merupakan lisensi di bawah naungan GNU, General *Public* License Versi 2(GPLv2), yang biasa dikenal dengan istilah *Copyleft*. Istilah *copyleft* ini merupakan lisensi yang setiap perbaikan oleh pihak ketiga harus terus jatuh di bawah *terms*.

Distribusi Android berada di bawah lisensi Apache Software (ASL/Apache2), yang memungkin untuk distribusi kedua atau seterusnya. Pengembang aplikasi Android diperbolehkan untuk mendistribusikan aplikasi mereka di bawah skema lisensi apapun yang mereka inginkan.

Pengembang memiliki beberapa pilihan dalam membuat aplikasi yang berbasis Android. Namun kebanyakan pengembang menggunakan Eclipse sebagai IDE untuk merancang aplikasi mereka. Hal ini diikarenakan Eclipse mendapat dukungan langsung dari Google untuk menjadi IDE pengembangan aplikasi Android.

Aplikasi Android dapat dikembangkan pada berbagai sistem operasi, diantaranya adalah:

a. Windows XP/Vista/7

b. Mac OS X (Mac OS X 10.48 atau yang lebih baru)

c. Linux

2.3.1 Sejarah Android

Pada saat perilisan perdana Android pada tanggal 5 November 2007, Android bersama Open Handset Alliance mendukung pengembangan standar terbuka pada perangkat seluler tersebut. Di sisi lain, Google merilis kode-kode Android di bawah lisensi Apache. Sehingga terdapat dua jenis *distributor* sistem operasi Android yaitu yang mendapat dukungan penuh dari Google dan yang mendapat dukungan penuh dari Open Handset Distribution (OHD).

Telepon selular pertama yang menggunakan sistem operasi Android adalah HTC Dream yang dirilis pada 22 Oktober 2008. Pada 9 Desember 2008, diumumkan anggota baru yang bergabung dalam program kerja Android ARM Holdings, Atheros Communication yang diproduksi oleh Asustek Computer Inc, Garmin Ltd, Softbank, Sony Ericsson, Toshiba Corp dan Vodafone Group Plc.

Hingga saat ini terdapat beberapa versi dari sistem operasi Android, antara lain:

a. Android versi 1.1

Dirilis pada 9 Maret 2009. Android versi ini dilengkapi dengan adanya jam, alarm, *voice search*, pengiriman pesan dengan Gmail dan pemberitahuan *email*.

b. Android versi 1.5 (Cupcake)

Dirilis pada Mei 2009. Terdapat pembaruan dari versi 1.1 diantaranya adalah fitur *upload* video ke Youtube dan gambar ke Picasa langsung dari telepon, dukungan *bluetooth* A2DP, kemampuan terhubung secara otomatis ke headset *bluetooth*, animasi layar, dan *keyboard* pada layar yang dapat disesuaikan dengan sistem.

c. Android versi 1.6 (Donut)

Dirilis pada September 2009. Pembaruan yang terdapat pada versi ini diantaranya adalah proses pencarian yang lebih baik, penggunaan baterai indikator dan kontrol applet VPN. Fitur lainnya adalah memungkinkan pengguna untuk memilih foto yang akan dihapus, kamera, camcorder dan galeri yang diintegrasikan, CDMA/EVDO, 802.1x, VPN, Gestures, *Text-to-speech engine*.

d. Android versi 2.1 (Éclair)

Dirilis pada 3 Desember 2009. Perubahan yang dilakukan adalah pengoptimalan *hardware*, peningkatan Google Maps 3.1.2, perubahan UI dengan *browser* baru dan dukungan HTML5, daftar kontak yang baru, dukungan *flash* untuk kamera 3.2 MP, digital zoom dan *bluetooth* 2.1.

e. Android versi 2.2 (Froyo)

Dirilis pada 20 Mei 2010. Versi Android inilah yang sekarang banyak digunakan sebagai standar sistem operasi mereka. Terdapat perubahan yang cukup signifikan dari versi sebelumnya diantaranya adalah kerangka aplikasi memungkinkan penggunaan dan penghapusan komponen yang tersedia, Dalvik Virtual Machine (DVM) yang dioptimalkan untuk perangkat *mobile*, grafik di 2D dan 3D berdasarkan *libraries* OpenGL, SQLite, mendukung berbagai format audio dan video, GSM, *bluetooth*, EDGE, 3G, Wifi, kamera, *Global Positioning System* (GPS), kompas dan accelerometer.

f. Android versi 2.3 (GingerBread)

Dirilis pada 6 Desember 2010. Beberapa perbaikan fitur dari versi sebelumnya adalah SIP-*based* VoIP, *Near Field Communications* (NFC),

gyroscope dan sensor, *multiple cameras support*, *mixable audio effect* dan *download manager*.

g. Android versi 3.0 (Honeycomb)

Dirilis tahun 2011. Android versi ini dirancang khusus untuk tablet, sehingga terdapat perbedaan dari fitur UI (*User Interface*). Honeycomb sengaja dibuat untuk layar yang lebih besar dan juga dapat mendukung *multiprocessor*.

h. Android versi 4.0 (Ice Cream)

Versi ini masih dalam pengembangan. Dari berbagai informasi menyebutkan bahwa versi Ice Cream merupakan gabungan antara versi Gingerbread dengan Honeycomb. Sehingga bisa digunakan untuk ponsel maupun tablet. Dan kemungkinan dirilis pada quarter ke 4 tahun 2011.

2.3.2 *Dalvik Virtual Machine*

Salah satu elemen penting dari Android adalah *Dalvik Virtual Machine* (DVM). Android berjalan di dalam DVM bukan di *Java Virtual Machine (JVM).* DVM dirancang untuk memastikan bahwa beberapa fitur berjalan lebih efisien pada perangkat *mobile.*

Tidak seperti JVM yang "*stack base*", DVM adalah sebuah "*register-base architecture*". DVM ditulis oleh Dan Bornsten dan beberapa *engineers* dari Google lainnya. Jadi jelas bahwa DVM tidak sama dengan JVM.

DVM menggunakan kernel Linux untuk menangani fungsionalitas tingkat rendah termasuk keamanan, threading dan proses serta manajemen memori. DVM mengeksekusi *executable file*, sebuah format yang dioptimalkan untuk memastikan memori yang digunakan sangat kecil. *Executable file* diciptakan

dengan mengubah kelas bahasa Java dan dikompilasi menggunakan *tools* yang disediakan dalam SDK Android.

2.3.3 Android SDK (*Software Development Kit*)

Android SDK adalah *tools* API (*Application Programming Interface*) yang diperlukan untuk memulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman Java.

Beberapa fitur Android yang paling penting adalah sebagai berikut:

a. *Framework* aplikasi yang mendukung penggantian komponen dan reusable

b. DVM dioptimalkan untuk perangkat *mobile*

c. Integrated *browser* berdasarkan engine *open source* WebKit

d. Grafis yang dioptimalkan dan didukung oleh *libraries* grafis 2D, grafis 3D berdasarkan spesifikasi OpenGL ES 1.0

e. SQLite untuk penyimpanan data

f. Dukungan untuk audio, video dan gambar

g. *Bluetooth*, EDGE, 3G, Wifi

h. Kamera, GPS, kompas dan *accelerometer*

i. Lingkungan development yang lengkap dan kaya termasuk perangkat *emulator*, *tools* untuk *debugging*, profil dan kinerja memori serta plugins untuk IDE Eclipse.

2.3.4 Arsitektur Android

Secara garis besar arsitektur Android dapat dijelaskan dan digambarkan sebagai berikut:

a. *Application* dan *Widget*

Application dan *Widget* ini adalah *layer* dimana kita berhubungan dengan aplikasi saja. Di *layer* terdapat aplikasi inti termasuk klien *email*, program SMS, kalender, peta, *browser*, kontak, dan lain-lain. Semua aplikasi ditulis dengan menggunakan bahasa pemrograman JAVA.

b. *Application Framework*

Application Framework adalah *layer* untuk melakukan pengembangan / pembuatan aplikasi yang akan dijalankan di sistem operasi Android, karena pada *layer* inilah aplikasi dapat dirancang dan dibuat, seperti *content provider* yang berupa SMS dan panggilan telepon.

Komponen-komponen yang termasuk di dalam *Application Framework* adalah sebagai berikut:

a) Views

b) *Content Provider*

c) *Resource Manager*

d) *Notification Manager*

e) *Activity Manager*

c. *Libraries*

Libraries adalah *layer* tempat fitur-fitur Android berada, biasanya para pengembang aplikasi mengakses *libraries* untuk menjalankan aplikasinya.

d. Android *Runtime*

Layer yang membuat aplikasi Android dapat dijalankan di mana dalam prosesnya menggunakan implementasi Linux. Dalvik *Virtual Machine* merupakan mesin yang membentuk dasar kerangka aplikasi Android.

Di dalam Android *Runtime* dibagi menjadi dua bagian yaitu:

1) Core *Libraries*

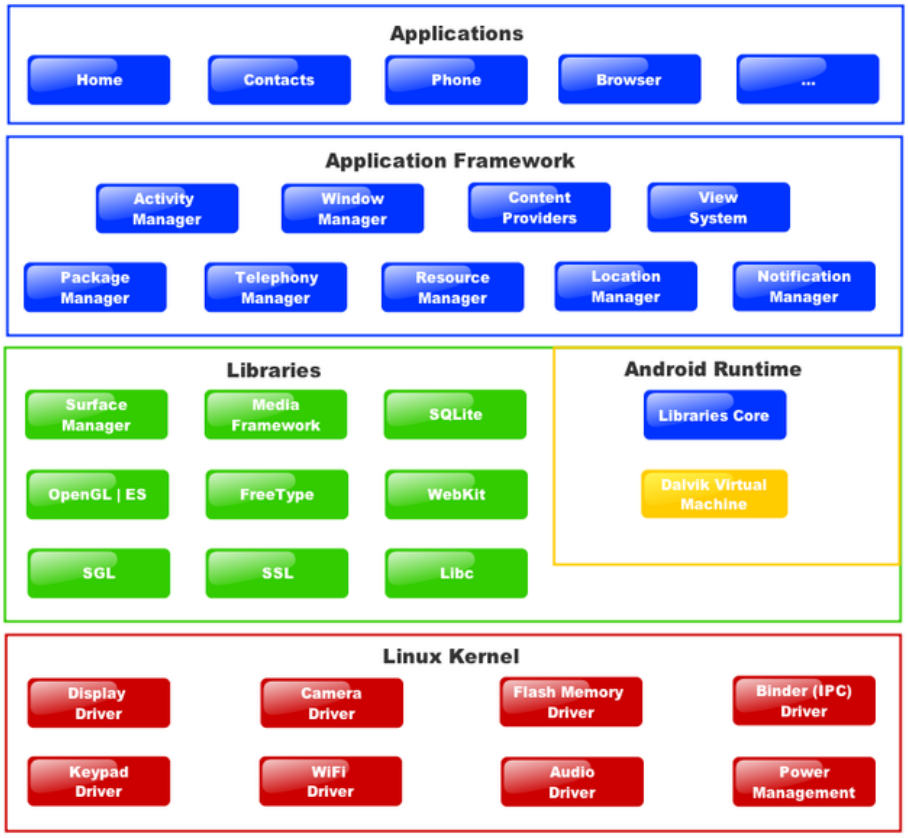
Aplikasi Android dibangun dalam bahasa Java, sementara DVM bukan merupakan *virtual machine* untuk Java. Sehingga diperlukan *libraries* yang berfungsi untuk menterjemahkan bahasa Java/C yang ditangani oleh *Core Libraries*

2) *Dalvik Virtual Machine*

Virtual Machine berbasis *register* yang dioptimalkan untuk menjalankan fungsi-fungsi secara efisien, dimana merupakan pengembangan yang mampu membuat Linux kernel untuk melakukan *threading* dan manajemen tingkat rendah.

e. Linux Kernel

Linux Kernel adalah *layer* dimana inti sistem operasi dari Android itu berada. Berisi file system yang mengatur system processing *memory, resource, drivers*, dan sistem-sistem operasi Android lainnya. Linux Kernel yang digunakan Android adalah Linux Kernel *release* 2.6.



Gambar 2.1 Arsitektur Android

(Sumber : NazruddinSafaat H.2012.*Pemrograman Aplikasi Mobile Smartphone dan Tablet PC berbasis Android*)

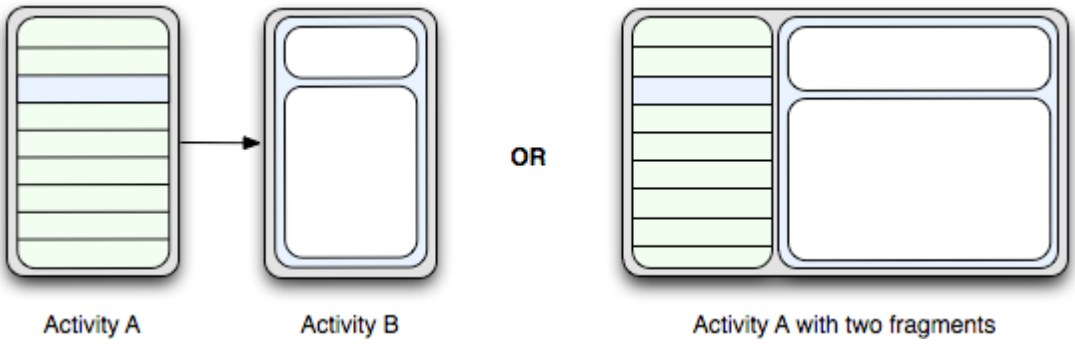
2.3.5 Fundamental Aplikasi

Aplikasi Android ditulis dengan menggunakan bahasa pemrograman Java. Kode Java dikompilasi bersama dengan data file *resource* yang dibutuhkan oleh aplikasi, dimana prosesnya di-*package* oleh *tools* yang dinamakan apt *tools* ke dalam paket Android sehingga menghasilkan *file* dengan ekstensi apk. File apk itu lah yang disebut dengan aplikasi dan siap diinstall di perangkat *mobile*.

Ada empat jenis komponen pada aplikasi Android, yaitu:

a. Activities

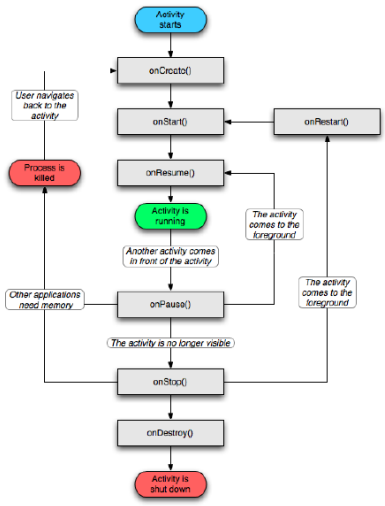
Suatu *activity* akan menyajikan *user interface* (UI) kepada pengguna, sehingga pengguna dapat melakukan interaksi. Sebuah aplikasi Android bisa jadi hanya memiliki satu *activity*, tetapi umumnya aplikasi memiliki banyak *activity* tergantung pada tujuan aplikasi dan desain dari aplikasi tersebut.



Gambar 2.2 Contoh Penggunaan *Activity*

(Sumber : NazruddinSafaat H.2012.*Pemrograman Aplikasi Mobile Smartphone dan Tablet PC berbasis Android***)**

Satu *activity* biasanya akan dipakai untuk menampilkan aplikasi yang bertindak sebagai *user interface*(UI) saat aplikasi diperlihatkan kepada *user*. Untuk pindah dari satu *activity* ke *activity* lain dapat dilakukan dengan satu *even*, misalnya klik tombol, memilih opsi atau menggunakan trigger tertentu. Secara hirarki sebuah window *activity* dinyatakan dengan *method Activity.setContentView*(). *Content View* adalah objek yang berada pada *root* hirarki.



Gambar 2.3 *Activity Lifecycle*

(Sumber : NazruddinSafaat H.2012.*Pemrograman Aplikasi Mobile Smartphone dan Tablet PC berbasis Android***)**

b. Service

Service tidak memiliki *Graphic User interface* (GUI), tetapi service berjalan secara background. Misalnya ketika ponsel sedang memainkan music di media p*layer*, namun user juga sedang mengetik SMS. *Activity* yang bisa terlihat oleh user adalah *activity* SMS, sedangkan *media player* yang sedang memainkan musik tidak terlihat, maka media p*layer* tersebut sedang menjalankan sebuah *service*.

c. Broadcast Receiver

Broadcast Receiver berfungsi menerima dan bereaksi untuk menyampaikan notifikasi. Sebagai contoh ketika baterai dalam keadaan *low* atau misalnya sudah berhasil menerima data hasil *download* atau pun *bluetooth*. *Broadcast Receiver* tidak memiliki *user interface* (UI), tetapi memiliki sebuah

activity untuk meresponn informasi yang mereka terima, atau mungkin menggunakan *Notification Manager* untuk memberitahu kepada pengguna, seperti lampu layar atau vibrating dan lain sebagainya.

d. Content Provider

Content Provider membuat kumpulan aplikasi data secara spesifik sehingga bisa digunakan oleh aplikasi lain. Data disimpan dalam file sistem seperti database SQLite. *Content Provider* menyediakan cara untuk mengakses data yang dibutuhkan oleh suatu *activity*, misalnya ketika menggunakan aplikasi yang membutuhkan peta(*Map*), atau aplikasi yang membutuhkan untuk mengakses data kontak dan navigasi, maka disinilah fungsi *Content Provider*.

2.4 JAVA

Java merupakan bahasa pemrograman berorientasi objek yang bersifat *open source* dan bisa dijalankan di berbagai *platform*. Java mengadopsi sintaks dari bahasa C dan C++. Setiap program yang ditulis dengan bahasa Java awalnya dikompilasi menjadi sebuah kode objek yang disebut *bytecode*. Hasil dari *bytecode* adalah file berekstensi *.*class*, selanjutnya file *.*class* tersebut akan diterjemahkan baris demi baris(*interpreter*).

Dalam terminologi Java terdapat istilah "*write once, run anywhere*". Ini berarti bahwa sekali menulis program Java dan melakukan kompilasi terhadap program Java tersebut, maka *bytecode*nya bisa dijalankan di dalam *platform* manapun selama *platform* tersebut terdapat *Java Virtual Machine (JVM).*

2.4.1 Sejarah Java

Bahasa pemrograman terlahir dari sebuah projek yang diberi nama The Green Project. Proyek tersebut berlangsung selama 18 bulan, mulai dari awal tahun 1991 hingga musim panas 1992. Proyek ini dilakukan oleh Patrick Naughton, Mike Sheridan, James Gosling dan Billy Joy beserta sembilan pemrogram lainnya dari Sun Microsystems.

Pada sekitar bulan Maret 1995, untuk pertama kali kode sumber Java versi 1.0 dibuka. Awalnya produk tersebut dinamakan Oak. Namun ternyata nama tersebut telah terdaftar sebagai merek dagang lain, sehingga mereka merubahnya menjadi Java yang diambil dari kopi murni yang digiling langsung yang juga merupakan kesukaan James Gosling.

Sekitar tahun 1996 mereka mengeluarkan Java versi 1.0 Release. Java versi ini menyertakan banyak paket standar awal yang terus dikembangkan pada versi selanjutnya:

a. Java.lang

b. Java.io

c. Java.util

d. Java.net

e. Java.awt

f. Java.applet

2.4.2 Konsep Pemrograman Berorientasi Objek

Pemrograman berorientasi objek atau lebih lazim dikenal dengan sebutan asing *Object Oriented Programming* (OOP) adalah inti dari pemrograman Java.

Semua program Java adalah objek. Berikut ini adalah ciri dari pemrograman berbasis objek:

2.4.2.1 Abstraksi

Abstraksi adalah pengabstrakan atau penyembunyian kerumitan dari suatu proses. Sebagai contoh sebuah perangkat telepon seluler terdiri dari puluhan atau ribuan komponen kecil seperti transistor, baut, lampu indikator dan lain-lain. Namun pengguna tidak perlu mengetahui bagaiman setiap komponen bisa bekerja sesuai fungsinya. Pengguna hanya cukup mengetahui bagaimana cara mengoperasikan telepon seluler tersebut. Ini artinya si pembuat telepon seluler tersebut telah menyembunyikan kerumitan-kerumitan proses yang terdapat pada telepon selular tersebut.

2.4.2.2 Pengkapsulan/Enkapsulasi

Dalam melakukan pembungkusan kode dan data di dalam Java terdapat tiga tingkat akses, yaitu:

a. Private

Dengan mendeklarasi data dan *method* menggunakan tingkat akses *private*, maka data dan *method* tersebut hanya dapa diakses oleh kelas yang memilikinya saja. Ini artinya data dan *method* tersebut tidak boleh diakses atau digunakan oleh kelas-kelas lain yang terdapat di dalam program. Untuk mendeklarasikan data atau *method* dengan tingkat akse *private* maka harus menggunakan kata kunci *private*.

b. *Protected*

Data atau *method* yang dideklarasikan dengan tingkat akses *protected* dapat diakses oleh kelas yang memilikinya dan juga oleh kelas-kelas yang masih turunan. Untuk mendeklarasikan data atau *method* yang menggunakan tingkat akses *protected* maka harus menggunakan kata kunci *protected*.

c. *Public*

Data atau pun *method* yang dideklarasikan dengan tingkat akses *Public* artinya data tersebut bisa diakses oleh semua bagian di dalam program. Untuk mendeklarasikan data atau *method* yang menggunakan tingkat akses p*ublic* maka harus menggunakan kata kuci *public*.

2.4.2.3 Pewarisan

Pewarisan (*inheritance*) adalah salah satu ciri pemrograman berorientasi objek, yang menyatakan bahwa suatu kelas dapat diturunkan lagi menjadi kelas-kelas baru lainnya sehingga dapat membentuk sebuah hirarki.

Pewarisan yang dimaksud dalam pemrograman berorientasi objek sama seperti pewarisan di dunia nyata. Sebagai contoh, setiap anak pasti mewarisi karakteristik atau beberapa sifat dari orang tuanya. Begitu juga di dalam pemrograman berorientasi objek, setiap kelas dapat diturunkan menjadi kelas-kelas baru dan mewarisi beberapa sifat atau perilakunya.

2.4.2.4 Polimorfisme

Polimorfisme adalah kemampuan suatu objek untuk mengungkapkan banyak hal melalui satu cara yang sama. Polimorfisme merupakan hal paling esensial dalam konsep pemrograman berorientasi objek karena polimorfisme

mengizinkan kelas induk untuk mendefinisikan sebuah *method* genera(bersifat umum) untuk semua kelas turunannya dan selanjutnya kelas-kelas turunan dapat memperbarui implementasi dari *method* tersebut secara lebih spesifik sesuai dengan karakteristiknya masing-masing.

2.4.3 Tipe Data

Di dalam bahasa pemrograman Java terdapat delapan tipe data sederhana, yaitu: byte, short, int, long, char, float, double dan Boolean. Tipe-tipe tersebut dikelompokkan lagi menjadi beberapa bagian, yaitu: tipe integer(bilangan bulat), tipe *floating point*(bilangan riil), karakter dan boolean(logika).

2.4.3.1 Tipe Integer

Tipe data integer terdiri dari empat jenis tipe, yaitu: byte, short, int dan long. Semua tipe data ini bersifat *signed*(bertanda) atau tipe data yang merepresentasikan nilai negative dan positif.

Ukuran rentang dari tipe data integer adalah seperti pada table berikut:

Tabel 2.1 Tipe Data Integer Pada Java

Tipe Data	**Ukuran (dalam bit)**	**Rentang**
byte	8	-128 sampai 127
short	16	-32.768 sampai 32.767
int	32	-2.147.483.648 sampai 2.147.483.647
long	64	-9.223.372.036.854.775.808 sampai 9.223.372.036.854.775.808

Sumber: Budi Raharjo dkk (2009)

2.4.3.2 Tipe *Floating-point*

Tipe *floating point* digunakan untuk merepresentasikan nilai-nilai yang mengandung pecahan atau angka decimal di belakang koma, seperti: 3.25 dan 3.50. Bilangan semacam ini biasanya disebut sebagai bilangan riil. Di dalam

bahasa pemrograman Java, tipe *floating point* dibedakan menjadi dua jenis, yaitu: float dan double.

Ukuran rentang nilai darri tipe *floating point* ditunjukkan pada tabel di bawah ini:

Tabel 2.2 Tipe *Floating*

Tipe Data	**Ukuran (dalam bit)**	**Rentang**
float	32	3.4e-038 sampai 3.4e+038
double	64	1.7e-308 sampai 1.7e+308

Sumber: Budi Raharjo dkk (2009)

2.4.3.3 Tipe Karakter

Pada bahasa pemrograman Java, untuk merepresentasikan data yang berupa karakter digunakan tipe data char. Java menggunakan karakter *Unicode* untuk merepresentasikan semua karakter yang ada. *Unicode* mendefinisikan sekumpulan karakter yang terdapat pada semua bahasa seperti bahasa Latin, Arab, Yunani dan sebagainya. Oleh karena itu, Java membutuhkan unkuran 16-bit untuk tipe karakter dan berada pada rentang 0 sampai 65.536.

2.4.3.4 Tipe Boolean

Tipe Boolean adalah tipe yang digunakan untuk menampung nilai logika, yaitu nilai yang hanya menampung dua buah kemungkinan (benar atau salah).tipe ini ditandai dengan kata kunci boolean. Dalam bahasa Java nilai benar direpresentasikan dengan kata kunci true, sedangkan nilai salah direpresentasikan dengan kata kunci false. Di dalam Java, nilai *true* dan *false* tidak bisa dikonversikan ke tipe numerik.

2.4.4 Variabel

2.4.4.1 Mendeklarasikan Variabel

Berikut ini adalah bentuk umum cara mendeklarasikan variabel pada bahasa pemrograman Java:

```
tipe namaVariabel;
```

Sebagai contoh apabila ingin mendeklarasikan sebuah variabel dengan tipe data int adalah sebagai berikut:

```
int tinggi;
```

Penamaan variabel tidak boleh mengandung spasi atau diawali karakter numerik (angka). Penamaan variabel yang benar adalah sebagai berikut:

```
int tinggiSegitiga;
int tinggi_segitiga;
```

2.4.4.2 Inisialisasi Variabel

Java mengizinkan kepada para pemrogram untuk melakukan inisialisasi terhadap suatu vaiabel yang dideklarasikan. Inisialisasi variabel adalah memberikan nilai awal (*default*) sebelum akhirnya variabel tersebut diisi dengan nilai yang baru. Berikut ini adalah contoh inisialisasi suatu variabel:

```
int index=0;
char ch=’A’;
```

2.4.5 Array

Array adalah sekumpulan variabel bertipe sama yang diacu dengan nama yang sama. Masing-masing elemen array dapat diakses melalui indeksnya. Di dalam bahasa pemrograman Java, indeks array selalu bertipe integer dan dimulai dengan 0.Pada Java, array terdiri dari array satu dimensi dan array multi dimensi. Berikut ini adalah pembahasan dari array satu dimensi dan array multi dimensi.

2.4.5.1 Array Satu Dimensi

Array dideklarasikan dengan tanda [ ] (*bracket*). Dalam Java bentuk umum dari deklarasi array satu dimensi adalah sebagai berikut:

tipe namaArray[];

atau

tipe [] namaArray;

Contoh dari pendeklarasian array satu dimensi adalah sebagai berikut:

int [] jumlahHari;

2.4.5.2 Array Multi Dimensi

Array multi dimensi sebenarnya merupakan array dari array. Artinya, terdapat sebuah array yang setiap elemennya juga bertipe array. Bentuk umum pendeklarasian array multi dimensi pada Java adalah:

tipe namaArray[][];

atau

tipe[][] namaArray;

Berikut ini adalah contoh dari pendeklarasian array multi dimensi:

int duaD[][];

2.4.6 Tipe *String*

Di dalam bahasa pemrograman Java, string bukanlah tipe data sederhana. Pada Java, string didefinisikan sebagai objek. Java telah mendefinisikan objek string untuk merepretasikan tipe string.

Berikut ini adalah pendekarasian dan pemberian nilai pada tipe String:

String nama=”Ismail Adhari”;

Pada contoh di atas, nama adalah objek dari tipe String dan bukan merupakan variabel dari tipe data sederhana. Di sini maksudnya, Java akan membentuk sebuah objek atau instance baru dari kelas String yang berisikan teks Deddy Pamudji. Selanjutnya objek tersebut akan diacu oleh variabel nama, yang sebenarnya merupakan sebuah variabel referensi yang mengacu ke tipe String.

2.4.7 Operator

2.4.7.1 *Operator* Aritmatika

Operator aritmatika adalah *operator-operator* yang digunakan untuk melakukan perhitungan-perhitungan matematis, seperti: penjumlahan, pengurangan, perkalian dan pembagian.

Tabel di bawah ini akan menunjukkan daftar *operator* yang termasuk ke dalam *operator* aritmatika:

Tabel 2.3*Operator* Aritmatika Pada Java

Operator	**Keterangan**
+	Penjumlahan
-	Pengurangan
*	Perkalian
/	Pembagian
%	Modulus(sisa bagi)

++	Increment(menambah nilai dengan 1)
--	Decrement(mengurangi nilai dengan 1)

Sumber: Budi Raharjo dkk (2009)

2.4.7.2 *Operator* Relasional

Operator relasional adalah *operator* yang menyatakan bahwa hubungan antara satu operand dengan operand lainnya; apakah sama dengan, tidak sama dengan, lebih besar, lebih kecil, lebih besar atau sama dengan, lebih kecil atau sama dengan. Hasil yang diberikan dari operasi yang melibatkan *operator* relasional adalah boolean(*true*/*false*).

Tabel di bawah ini akan menunjukkan daftar *operator* yang termasuk ke dalam *operator* relasional:

Tabel 2.4*Operator* Relasional Pada Java

Operator	**Keterangan**
==	Sama dengan
!=	Tidak sama dengan
>	Lebih besar
<	Lebih kecil
>=	Lebih besar atau sama dengan
<=	Lebih kecil atau sama denggan

Sumber: Budi Raharjo dkk (2009)

2.4.7.3 *Operator* Boolean

Operator boolean digunakan untuk melakukan operasi terhadap dua operand yang bertipe boolean. Hasil yang diberikan dari operasi ini juga akan bertipe boolean.

Tabel di bawah ini akan menunjukkan daftar *operator* yang termasuk ke dalam *operator* relasional:

Tabel 2.5*Operator* Boolean Pada Java

Operator	**Keterangan**
&&	Operasi AND
||	Operasi OR
^	Operasi XOR (exclusive OR)
!	Operasi NOT (negasi)

Sumber: Budi Raharjo dkk (2009)

Apabila terdapat dua operand bernilai boolean (missal A dan B) dan keduanya digunakan dalam operasi AND, OR, XOR dan NOT, maka nilai-nilai yang dihasilkan adalah sebagai berikut:

Tabel 2.6 Operasi Boolean

A	**B**	**A&&B**	**A||B**	**A^B**	**!A**
True	*True*	*True*	*True*	*False*	*False*
True	*False*	*False*	*True*	*True*	*False*
False	*True*	*False*	*True*	*True*	*True*
False	*False*	*False*	*False*	*False*	*True*

Sumber: Budi Raharjo dkk (2009)

2.4.7.4 *Operator Bitwise*

Operator bitwise digunakan untuk melakukan operasi boolean terhadap dua buah operand bertipe integer. Operasi ini dilakukan bit demi bit. Dengan demikian, secara *implicit* nilai integer tersebut akan dikonversi terlebih dahulu ke dalam bentuk biner. Dalam operasi ini nilai *true* direpresentasikan dengan 1 dan nilai *false* direpresentasikan dengan nilai 0.

Berikut ini daftar *operator bitwise* yang terdapat di dalam Java:

Tabel 2.7*Operator Bitwise* Pada Java

Operator	**Keterangan**
&	Operasi *bitwise* AND
|	Operasi *bitwise* OR
^	Operasi *bitwise* XOR

~	Operasi NOT (negasi)
>>	Operasi shift right (geser ke kanan sebanyak n bit)
>>>	Operasi shift right zero fill
<<	Operasi shift left (geser ke kiri sebanyak n bit)

Sumber: Budi Raharjo dkk (2009)

2.4.8 *Statement* Kontrol

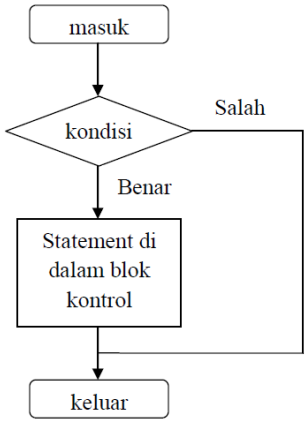
Statement control digunakan untuk mengatur jalannya alur program sesuai dengan yang kita inginkan. *Statement-statement* ini dikategorikan ke dalam tiga jenis, yiatu: pemilihan, pengulangan dan peloncatan. Pemilihan digunakan untuk menentukan *statement* mana yang akan dieksekusi tergantung dari ekspresi atau kondisi yang didefinisikan. Pengulangan digunakan untuk melakukan eksekusi terhadap *statement* secara berulang sesuai dengan kondisi yang ditentukan. *Statement* peloncatan digunakan untuk memindahkan proses eksekusi ke bagian program yang kita inginkan.

2.4.8.1 Pemilihan

Java menyediakan dua buah *statement* untuk proses pemilihan, yaitu: if dan switch. Kedua buah *statement* tersebut digunakan untuk mengontrol eksekusi *statement* tergantung pada kondisi yang ditentukan sebelumnya.

a. *Statement* if

Statement if dapat digunakan untuk menangani percabangan atau pemilihan *statement* yang didasarkan atas: satu, dua atau lebih dari dua kondisi. Berikut ini adalah alur untuk *statement* if dengan satu kondisi:



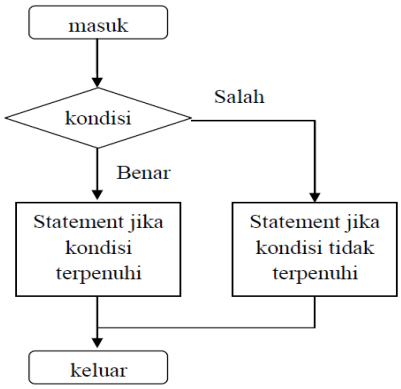
Gambar 2.4 Alur dari *Statement* if Satu Kondisi

Sumber: Budi Raharjo dkk (2009)

Berikut ini adalah bentuk umum dari *statement* if satu kondisi pada bahasa pemrograman Java:

```
If(kondisi){
Statement1;
Statement2;
…..
Statement;
}
```

Berikutnya adalah alur dari *statement* if dua kondisi:



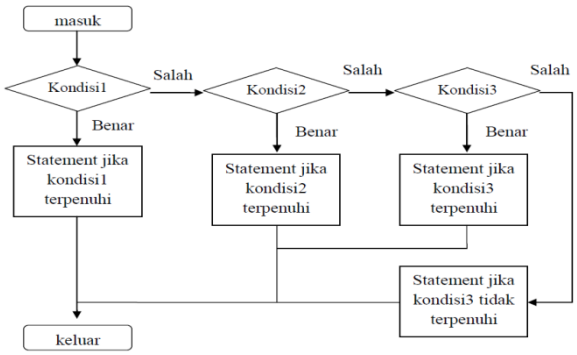
Gambar 2.5 Alur *Statement if* Dua Kondisi

Sumber: Budi Raharjo dkk (2009)

Berikut adalah bentuk umum dari *statement* if dengan dua kondisi:

```
If(kondisi){
StatementBenar;
} else {
StatementSalah;
}
```

Berikutnya adalah alur untuk *statement if* lebih dari dua kondisi, dalam hal ini adalah tiga kondisi:



Gambar 2.6 Alur *Statement if* Lebih dari Dua Kondisi

Sumber: Budi Raharjo dkk (2009)

Selanjutnya adalah bentuk umum dari *statement* if pada bahasa pemrograman Java:

```
If(kondisi1){
  Statement;
} else if(kondisi2){
  Statement;
} else if(kondisi3){
  Statement;
  } else {
  Statement;
  }
```

b. *Statement* switch

Statement switch merupakan alternatif lain untuk melakukan pemilihan *statement*. *Statement* ini biasanya digunakan untuk menyederhanakan *statement* if yang banyak mengandung kondisi. Berikut ini adalah bentuk umum dari *statement* switch:

```
Switch(ekspresi){
case nilai1:
//statement yang dilakukan bila ekspresi sama dengan nilai1
break;
```

```
case nilai2:
//statement yang dilakukan bila ekspresi sama dengan nilai2
break;
case nilai3:
//statement yang dilakukan bila ekspresi sama dengan nilai3
break;
case nilaiN:
//statement yang dilakukan bila ekspresi sama dengan nilaiN
break;
default:
//statement yang dilakukan bila tidak ada nilai yang sama dengan ekspresi
```

2.4.8.2 Pengulangan

Pengulangan adalah suatu proses di dalam program yang dapat mengeksusi satu atau beberapa *statement* yang sama secara berulang sampai ditemukan kondisi untuk berhenti. Di dalam bahasa pemrograman Java terdapat tiga jenis pengulanga, yaitu for, while dan do-while.

a. Struktur for

Struktur for pada umumnya digunakan untuk melakukan pengulangan yang banyaknya dan sudah pasti atau sudah diketahui sebelumnya. Dalam struktur for, harus mendefinisikan terlebih dahulu inisiasi dan kondisi untuk keluar dari pengulangan.

Berikut ini adalah bentuk umum dari penulisan sintaks dari struktur pengulangan for:

```
for(inisialisasi;kondisi;iterasi){
//statement yang akan diulang
}
```

b. Struktur while

Struktur while adalah jenis pengulangan yang mendefinisikan kondisi di awal blok. Ini artinya apabila kondisi tidak terpenuhi (bernilai *false*) maka proses pengulangan tidak akan pernah dilakukan.

Berikut ini adalah bentuk umum penulisan sintaks dari struktur pengulangan while:

```
while(kondisi){
//statement yang akan diulang
iterasi;
}
```

c. Struktur do-while

Struktur do-while sebenarnya hampir sama dengan struktur while. Perbedaannya hanya terletak pada penempatan kondisinya saja. Pada struktur pengulangan while, kondisi ditulis di awal blok program. Sedangkan pada do-while, kondisi ditulis di akhir blok program. Berikut ini adalah bentuk umum penulisan sintaks dari struktur pengulangan do-while:

```
Do{
//statement yang akan diulang
```

Iterasi;

}while(kondisi)

2.4.8.3 Peloncatan

Statement peloncatan digunakan untuk mengontrol jalannya program. Lebih tepatnya, untuk memindahkan eksekusi program ke baris kode yang dikehendaki. Di dalam Java terdapat tiga buah *statement* peloncatan, yaitu: break, continue dan return.

a. break

Di dalam bahasa pemrogram Java, *statement break* memiliki tiga kegunaan yaitu:

- Menghentikan runtunan pemilihan pada struktur *statement* switch
- Menghentikan proses pengulangan
- Keluar dari blok label tertentu

Penggunaan perintah ini adalah dengan menuliskan *statement* break pada blok program.

b. *continue*

Java menyediakan *statement continue* yang berguna untuk memaksa program agar melanjutkan proses pengulangan dengan mengabaikan *statement* atau perintah-perintah selanjutnya. Penggunaannya adalah dengan menuliskan *statement continue* pada blok program.

c. *Return*

Di dalam bahasa pemrograman Java, *statement return* digunakan untuk keluar dari eksekusi kode program yang terdapat dalam sebuah *method*. Jadi untuk penggunaan *statement* tersebut maka harus menulis return pada sebuah *method*.

2.5 *Unified Modeling Language* (UML)

2.5.1. Definisi *Unified Modeling Language* (UML)

Berikut ini definisi *Unified Modeling Language* (UML) menurut para ahli:

1. Menurut (Hend, 2006) "*Unified Modeling Language* (UML)adalah bahasa yang telah menjadi standard untuk visualisasi, menetapkan, membangun dan mendokumentasikan artifak suatu sistem perangkat lunak".
2. Menurut (Adi Nugroho : 2005). "*Unified Modeling Language* (UML) adalah alat bantu analisis serta perancangan perangkat lunak berbasis objek".
3. Menurut(Joomladarihttp://soetrasoft.com:2007). "*Unified Modeling Language* (UML)merupakan standard modeling language yang terdiri dari kumpulan-kumpulan diagram, dikembangkan untuk membantu para pengembang sistem dan *software* agar bisa menyelesaikan tugas-tugas seperti: Spesifikasi, Visualisasi, Desain Arsitektur, Konstruksi, Simulasi dan testing serta Dokumentasi".

Berdasarkan beberapa pendapat yang dikemukakan diatas dapat ditarik kesimpulan bahwa "*Unified Modeling Language* (UML)adalah sebuah bahasa yang berdasarkan grafik atau gambar untuk menvisualisasikan, menspesifikasikan, membangun dan pendokumentasian dari sebuah sistem pengembangan perangkat lunak berbasis OO(*Object Oriented*)".

Unified Modelling Language (UML) adalah sebuah bahasa untuk menentukan, visualisasi, konstruksi, dan mendokumentasikan *artifacts* dari sistem *software*, untuk memodelkan bisnis, dan sistem *nonsoftware* lainnya. *Artifacts* adalah sepotong informasi yang digunakan atau dihasilkan dalam suatu proses rekayasa

software. *Artifacts* dapat berupa model, deskripsi, atau software. Untuk membuat suatu model, UML memiliki diagram grafis yang diberi nama berdasarkan sudut pandang yang berbeda-beda terhadap sistem dalam proses analisa atau rekayasa. Diagram grafis tersebut antara lain

1. Use Case diagram

Use Case diagram adalah jenis diagram perilaku yang didefinisikan dan diciptakan dari analisis kasus. Tujuannya adalah untuk menyajikan gambaran grafis dari fungsionalitas yang disediakan oleh sistem dalam hal pelaku, tujuan mereka (direpresentasikan sebagai kasus penggunaan), dan setiap ketergantungan antara kasus-kasus digunakan. Dan tujuan utama dari sebuah diagram *Use Case* adalah untuk menunjukkan apa yang dilakukan fungsi sistem yang aktor. Peran aktor dalam sistem dapat digambarkan. *Use Case* diagram secara formal dimasukkan dalam dua bahasa pemodelan yang didefinisikan oleh OMG: Unified Modeling Language (UML) dan Sistem Modeling Language(SysML).

Sebuah *Use Case* menggambarkan suatu urutan tindakan yang memberikan sesuatu yang bernilai terukur untuk aktor. Dalam *Use Case* diagram ada empat jenis hubungan yang saling terkait satu sama dan sering digunakan dalam praktek. Empat hubungan ini adalah

a. Include

Relasi ini menyatakan bahwa satu *Use Case* selalu menggunakan fungsionalitas yang disediakan oleh *Use Case* lainnya

b. Extend

Relasi ini menyatakan bahwa suatu *Use Case* tidak selalu menggunakan fungsionalitas yang disediakan oleh *Use Case* lainnya.

c. Generalization

Generalization ada 2 jenis yaitu : generalisasi antar aktor dan generalisasi antar *Use Case*. Generalisasi antar actor adalah relasi antara satu actor dengan actor yang lain. Dimana salah satu actor berperan sebagai Actor yang diwariskan sifatnya (Base Actor) dan Actor yang lain sebagai Actor yang mewarisi sifatnya (Child Actor). Sedangkan Generalisasi antar *Use Case*adalah relasi antara satu *Use Case* dengan *Use Case* yang lain. Dimana salah satu *Use Case* berperan sebagai *Use Case* yang diwariskan kerjanya (Base *Use Case*) dan *Use Case* yang lain mewarisi kerja (Child *Use Case*).

d. Associations

Adalah komunikasi antara aktor dengan use yang berada dalam sebuah sistem. Asosiasi ini menggambarkan apa yang actor bisa lakukan dengan sistem. Dimana setiap use menggambarkan satu buah perilaku yang bisa actor lakukan.

2. Class diagram

Class diagram adalah jenis diagram struktur statis yang menggambarkan struktur sistem dengan menunjukkan kelas sistem, atribut mereka, operasi (atau) metode dan hubungan antara kelas-kelas. Class diagram digunakan untuk pemodelan konseptual sistematiska aplikasi, dan untuk pemodelan rinci yang menerjemahkan midel kedalan kode program. Kelas dalam diagram kelas mewakili kedua objek utama dan atau interaksi dalam aplikasi dan objek yang

akan diprogram. Dalam class diagram kelas – kelas ini diwakili dengan kotak – kotak yang berisi tiga bagian. Sebuah kelas dengan tiga bagian :

- Bagian atas memegang nama kelas
- Bagian tengah berisi attribut kelas
- Bagian bawah memberikan metode atau operasi kelas dapat mengambil atau melakukan

3. Activity diagram

Activity diagram adalah representasi grafis dari alur kerja tahapan aktifitas. Diagram ini mendukung pilihan tindakan, iterasi dan *concurrency*. *Activity diagram* memodelkan alur kerja sebuah proses bisnis dan urutan aktivitas dalam suatu proses. Diagram ini sangat mirip dengan *flowchart* karena dapat memodelkan sebuah alur kerja satu aktivitas ke aktivitas lainnya atau dari satu aktivitas ke dalam keadaan sesaat (*state*). Seringkali bermanfaat bila membuat sebuah proses *activity diagram* terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan.

4. Sequence diagram

Sequence diagram adalah suatu diagram yang menggambarkan interaksi antar obyek dan mengindikasikan komunikasi diantara obyek-obyek tersebut. Diagram ini juga menunjukkan serangkaian pesan yang dipertukarkan oleh obyek-obyek yang melakukan suatu tugas atau aksi tertentu. Obyek-obyek tersebut kemudian diurutkan dari kiri ke kanan, aktor yang menginisiasi interaksi biasanya ditaruh di paling kiri dari diagram.

2.5.2. Langkah-langkah Penggunaan *Unified Modeling Language* (UML)

Menurut (Afif Amrullah:2002). "Langkah-langkah penggunaan *Unified Modeling Language* (UML) sebagai berikut:

1. Buatlah daftar *business process* dari *level* tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.
2. Petakan *Use Case* untuk setiap *business process* untuk mendefinisikan dengan tepat fungsional yang harus disediakan oleh sistem, kemudian perhalus *Use Casediagram* dan lengkapi dengan *requirement, constraints* dan catatan-catatan lain.
3. Buatlah*deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem.
4. Definisikan *requirement* lain *non fungsional, security*dansebagainya yang jugaharusdisediakanolehsistem.
5. Berdasarkan*Use Case diagram,*mulailahmembuat*activity diagram.*
6. Definisikan obyek-obyek level atas *package* atau *domain* dan buatlah *sequence* dan/atau *collaboration* utuk tiap alir pekerjaan, jika sebuah *Use Case* memiliki kemungkinan alir normal dan *error*, buat lagi satu diagram untuk masing-masing alir.
7. Buatlah rancangan *user interface model* yang menyediakan antamuka bagi pengguna untuk menjalankan *skenario Use Case.*
8. Berdasarkan model-model yang sudah ada, buatlah *class diagram*. Setiap *package* atau *domian* dipecah menjadi *hirarki class* lengkap dengan

*atribut*dan metodenya. Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji *fungsionalitas class* dan interaksi dengan *class* lain.

9. Setelah *class diagram* dibuat, kita dapat melihat kemungkinan pengelompokkan *class* menjadi komponen-komponen karena itu buatlah *component diagram* pada tahap ini. Juga, definisikan *test integrasi* untuk setiap komponen meyakinkan ia bereaksi dengan baik.
10. Perhalus*deployment diagram* yang sudahdibuat. Detilkan kemampuan dan *requirement* piranti lunak, sistem operasi, jaringan dan sebagainya. Petakankomponen ke dalam node.
11. Mulailah membangun sistem. Ada dua pendekatan yang tepat digunakan:

a. Pendekatan *Use Case* dengan meng*assign* setiap *Use Case* kepada tim pengembang tertentu untuk mengembangkan unit kode yang lengkap dengan *test.*

b. Pendekatan*komponen* yaitu *mengassign* setiap komponen kepada tim pengembang tertentu.

12. Lakukan uji modul dan uji integrasi serta perbaiki model beserta *code*nya. Model harus selalu sesuai dengan *code* yang aktual.
13. Perangkat lunak siap dirilis".

2.6 Eclipse

Eclipse adalah sebuah IDE (*Integrated Development Environment*) untuk mengembangkan perangkat lunak dan dijalankan di semua *platform*. Eclipse sendiri juga merupakan sebuah komunitas *open source*, yang memiliki proyek

yang berfokus pada membangun sebuah *platform* pengembangan terbuka dari extensible *Framework, tools* dan runtime untuk membangun, menyebarkan dan mengelola perangkat lunak di seluruh siklus hidup perangkat lunak tersebut.

Umumnya Eclipse digunakan untuk membuat sebuah program yang menggunakan bahasa pemrograman Java. Namun, Eclipse juga bisa digunakan untuk penggunaan bahasa pemrograman lainnya seperti C, C++, COBOL, Perl, PHP, Python, dan sebagainya.

2.6.1 Versi Eclipse

Berikut ini adalah versi Eclipse yang telah dirilis:

Tabel 2.8 Versi Eclipse

Nama	**Tanggal rilis**	**Versi**
Eclipse 3.0	21 Juni 2004	3.0
Eclipse 3.1	28 Juni 2008	3.1
Callisto	30 Juni 2006	3.2
Europa	29 Juni 2007	3.3
Ganymade	25 Juni 2008	3.4
Galileo	24 Juni 2009	3.5
Helios	23 Juni 2010	3.6
Indigo	22 Juni 2011	3.7

Sumber: Eclipse *Foundation* (2011)