BAB 2

TINJAUAN PUSTAKA

2.1. Closed Domain Question Answering System

Closed Domain Question Answering System merupakan sistem pemrosesan teks yang digunakan untuk menjawab pertanyaan dengan format bahasa sehari-hari yang diberikan oleh pengguna dari sistem. *Question Answering System* kemudian mengekstraksi jawaban dari koleksi dokumen yang terbatas[4].

Bila dilihat dari jenis pertanyaan yang diberikan dan jawaban yang dikembalikan, QAS dapat dibagi menjadi lima jenis [5], yaitu:

1. *Factoid*, pertanyaan dengan jawaban berupa sebuah fakta singkat yang biasanya terdiri dari beberapa kata saja.

   Contoh: pertanyaan : "siapakah presiden pertama Republik Indonesia?"

   Jawaban: "Soekarno"

2. *Non factoid*, pertanyaan dengan jawaban fakta yang cukup panjang yang biasanya terdiri dari satu klausa atau lebih.

3. *Definition*, pertanyaan yang menanyakan definisi dari sesuatu.

   Contoh: pertanyaan : "apakah ular itu?"

   Jawaban : "ular adalah reptil yang tak berkaki dan bertubuh panjang"

4. *How*, pertanyaan yang menanyakan cara.

   Contoh: pertanyaan : "bagaimana cara menggunakan sedotan?"

Jawaban : “cara menggunakan sedotan adalah dengan memasukan sedotan ke air, kemudian hisap dari ujung lubangnya”

5. *Why*, pertanyaan mengenai penyebab.

Contoh: pertanyaan : “mengapa indonesia miskin?”

Jawaban : “banyak pejabat yang koruptor”

6. *Hypothetical*, pertanyaan hipotesis yaitu penjelasan yang belum bisa dibuktikan.

Contoh: pertanyaan: “bila seluruh hutan habis menjadi pemukiman manusia, apa yang akan terjadi?”

Jawaban: “aneka ragam flora dan fauna yang berhabitat di hutan tersebut akan punah”

7. *List*, pertanyaan dengan jawaban berupa daftar atau beberapa jawaban lebih dari satu.

Contoh: pertanyaan : “apa saja tim yang memenangkan liga champion 3 tahun terakhir?”

Jawaban : “Barcelona, inter milan, Bayern Muenchen”

8. *Yes / No*, pertanyaan yang jawabannya terdiri dari dua jawaban “ya” atau “tidak”.

Contoh: pertanyaan : “apakah Soeharto adalah presiden ke 2 Republik Indonesia?”

Jawaban : “ya”

9. *Opinion*, pertanyaan yang mengharapkan jawaban berupa pendapat.

Contoh: pertanyaan : “apakah mengerjakan skripsi itu susah?”

Jawaban : “menurut beberapa orang, mengerjakan skripsi itu sangat susah karena banyak yang tidak lulus karena belum sanggup mengerjakannya”

2.2. Skema Question Answering

Pada umumnya sistem QA terdiri atas tiga komponen utama[6], yaitu *question analyzer, passage retriever,* dan *answer finder*.

1. *Question analyzer*

   Komponen ini berfungsi untuk menentukan beberapa informasi yang dapat diperoleh dari kalimat pertanyaan masukan, antara lain: tipe jawaban yang diharapkan / EAT (*Expected answer type*) dan kata-kata kunci. Kata-kata kunci yang dihasilkan oleh komponen ini selanjutnya akan digunakan oleh komponen *passage retriever* untuk mengambil dokumen-dokumen yang diperkirakan mengandung jawaban dari pertanyaan.

2. *Passage retriever*

   Komponen ini berfungsi untuk menemukan / mengambil dokumen-dokumen (atau bagian dari dokumen) yang kemungkinan mengandung jawaban dari pertanyaan yang diberikan berdasarkan kata kunci yang dihasilkan oleh komponen *question analyzer*. Komponen ini menggunakan metode *information retrieval* (IR) untuk mengambil dokumen-dokumen tersebut.

3. *Answer finder*

   Komponen ini berfungsi untuk menentukan kandidat-kandidat jawaban yang diperoleh dari dokumen-dokumen yang telah ditemukan sebelumnya oleh komponen *passage retriever*. Biasanya jawaban ditentukan dengan mencocokkan kandidat jawaban dengan EAT yang diperoleh sebelumnya oleh komponen *question analyzer*.

2.3. Pendekatan Dan Metode Dalam *Question Answering*

Pendekatan dan metode yang dibahas dalam subbab ini difokuskan pada pendekatan yang digunakan untuk QA yang dapat menangani pertanyaan terstruktur. Pembahasan mengenai pendekatan dan metode dalam QA ini akan dibagi menjadi tiga bagian sesuai dengan komponen yang ada pada sistem QA yaitu *question analyzer, passage retriever*, dan *answer finder*.

2.3.1 Komponen *Question Analyzer*

Komponen ini berfungsi untuk mengklasifikasikan jenis pertanyaan dan menentukan informasi-informasi yang dibutuhkan dari pertanyaan yang dimasukkan tersebut. Informasi tersebut misalnya *Expected Answer Type* (EAT), kata-kata kunci, fokus/topik pertanyaan dan informasi lain yang dapat digunakan untuk membantu proses di komponen selanjutnya.

Beberapa metode yang dapat digunakan untuk komponen question analyzer antara lain:

1. *Question pattern matching* [7] [8] [9],
2. *Machine learning* [10],
3. *Parsing* [11],
4. Analisis semantik [12] [13].

1. Question pattern matching

Pada metode *question pattern matching*, setiap kategori pertanyaan sudah mempunyai pola tersendiri sehingga pertanyaan yang masuk hanya perlu dicocokkan dengan pola yang sudah ada tersebut. Pola yang didefinisikan sendiri ini tentu sangat bergantung pada bahasa yang digunakan oleh sistem QA. Contoh penggunaan metode ini dapat dilihat pada tabel 2.1.

Tabel 2.1 contoh pattern untuk pertanyaan *factoid* dan *non-factoid* [9]

Question type	**Question pattern**
Non-factoid	
Definition	[what]+[is]+ <term>
Reason	[why]+<term>
Relationship	[what]+[is]+[relationship]+<entity-1>+[and]+<entity-2>
Biography	[who]+[is]+<person>
Event	[what]+[is/are]+<event>
Method	[how]+[to]+[use+do]+<term>
Factoid	
Person	[who]+[is]+<point-to-person>
Location	[where]+[is]+<point-to-location>
Date	[when]+<point-to-time>;[what]+[time]
Organization	[what/which]+<organization>
quantity	[how]+[many+much]+<point-to-quantity>

2. Machine Learning

Pada metode *machine learning* (pembelajaran mesin), digunakan algoritma tertentu agar sistem dapat mempelajari jenis-jenis pertanyaan yang ada dan menentukan EAT dari pertanyaan tersebut. Penggunaan metode pembelajaran mesin ini memerlukan data pembelajaran (*training data*) agar dapat mengklasifikasikan pertanyaan dengan akurat. Beberapa atribut yang ditentukan

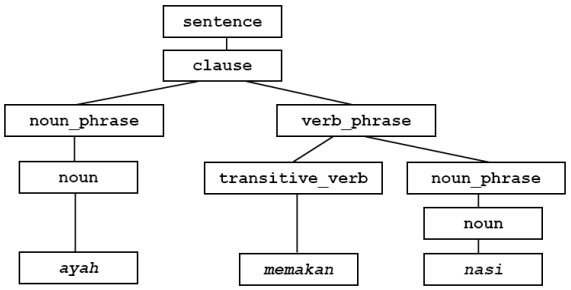
antara lain kata tanya, preposisi, kandidat EAT, *WordNet distance*, dan *bi-gram frequency*. Contoh penggunaan dan penerapan dapat dilihat pada tabel 2.2.

Tabel 2.2 *Rule* untuk menentukan kandidat EAT [10]

Kata tanya	**preposisi**	**frase**	**Kandidat EAT**
Kapan			Date
Berapa			Date, quantity, name
Siapa			Person, organization, location, name
Mana	Di, ke, dari	PN	Location, organization
Apa	Not blank	NP	Location, organization, name, date
Apa/mana	blank	NP	Organization, location, name, person

3. Parsing

Pada metode *parsing*, dilakukan proses pada kalimat pertanyaan yang menjadi masukan. Dari kalimat ini kemudian dihasilkan *parse tree* yang menunjukan struktur kalimat ini dan posisi dari masing-masing kata (penyusun kalimat) tersebut. Kemudian *parse tree* yang dihasilkan akan dicocokkan dengan struktur kalimat yang ada pada koleksi dokumen. Berikut contoh *parse tree* yang dihasilkan dari sebuah kalimat pada gambar 2.1.



Gambar 2.1 Contoh *parse tree* dari sebuah kalimat [13]

4. Analisis Semantik

Pada metode analisis semantik, dihasilkan representasi semantik dari kalimat yang diberikan. Komponen yang khusus menangani pembentukan representasi semantik dari kalimat tersebut biasanya adalah *semantic analyzer*. Komponen ini menghasilkan representasi semantik dari *parse tree* yang dihasilkan sebelumnya. Gambar 2.2 adalah contoh representasi semantik dari kalimat pernyataan "presiden mengumumkan susunan kabinet baru kepada wartawan di istana" dan kalimat pertanyaan "siapa mengumumkan susunan kabinet baru?".

```
event(x1, umum).
agent(x1,x2).
patient(x1,x3).
object(x2,presiden).
object(x2,susunan).
object(x2,kabinet).
nn(x3,x4).
adjective(x5,baru).
attrib(x4,x5).
object(x6,wartawan).
purpose(x1, x6).
object(x7, istana).
di(x1, x7).
```

```
Query(Ans)   :- event(A, umum),
                agent(A, B),
                patient(A, D),
                object(B, Ans),
                object(D, susunan),
                object(E, kabinet),
                adjective(F, baru),
                attrib(E, F),
                nn(D, E).
```

Gambar 2.2 Contoh representasi semantik [12]

2.3.2 Komponen *Passage Retriever*

Komponen ini berfungsi untuk mengambil dokumen-dokumen yang diperkirakan mengandung kandidat-kandidat jawaban dari pertanyaan yang diberikan berdasarkan informasi yang didapat dari komponen *question analyzer*. Umumnya komponen ini ditunjang oleh teknik *information retrieval* (IR), maka untuk mendapatkan hasil yang baik dibutuhkan teknik IR yang baik. Salah satu metode yang sering dipakai dalam IR dan dipakai pada sistem QA adalah metode ruang vektor atau *vector space model* [11]. Masukan pertanyaan akan digunakan

sebagai *query* pencarian pada mekanisme IR yang kemudian digunakan untk mencari dokumen yang mengandung term dari query tersebut. Pada metode ruang vektor ini, *query* dan dokumen direpresentasikan sebagai vektor-vektor. Kedekatan antara dua vektor disebut dengan *similarity* atau kemiripan. Kemiripan tersebut dihitung dengan rumus kosinus. Semakin dekat jarak kedua vektor, semakin besar kemiripannya maka semakin relevan dokumen tersebut. Untuk mengukur tingkat kerelevanan dokumen-dokumen tersebut digunakan pembobotan yang biasanya menggunakan *term frequency* (TF), *inverse document frequency*(IDF), dan normalisasi. Berikut persamaan (2.1) yang digunakan untuk menghitung *similarity* [14]

$$Sim(Q,\mathcal{D}) = \sum_{i=1}^{T}\left(\frac{w_{qi}}{\sqrt{w_{q1}^2+w_{q2}^2+\cdots+w_{qr}^2}} \times \frac{w_{di}}{\sqrt{w_{d1}^2+w_{d2}^2+\cdots+w_{dr}^2}}\right)$$ Persamaan (2.1)

Similarity

keterangan:

sim(Q,D) = *similarity vektor query* dan vektor dokumen

w_{qi} = bobot *term* i dalam *query*

w_{di} = bobot *term* i dalam dokumen

r = jumlat *term* di query

s = jumlah *term* di dokumen

2.3.3 Komponen *Answer Finder*

Komponen ini berfungsi untuk menemukan kandidat-kandidat jawaban dari dokumen atau bagian dokumen yang berhasil diambil/ditemukan oleh komponen *passage retriever*. Kandidat jawaban kemudian akan diurutkan berdasarkan aturan tertentu.

Untuk penanganan tipe pertanyaan *non-factoid* ataupun *factoid* pendekatan yang dapat digunakan antara lain adalah *pattern matching*/pencocokan pola[7][8],

machine learning/pembelajaran mesin [15], dan analisis semantik [12]. Pada pendekatan pencocokan pola digunakan aturan berdasarkan *surface expression pattern* dan *linguistic clue*. *Surface expression pattern* adalah pola tertentu yang disusun oleh kata-kata tertentu yang sering muncul berdasarkan jenis pertanyaannya, misalnya *definition, why, reason, how*, dsb. Sedangkan *linguistic clue* adalah petunjuk-petunjuk bahasa yang dapat membantu mengidentifikasi posisi atau klasifikasi kalimat. Beberapa contoh penerapan metode *pattern matching* untuk tipe pertanyaan *non-factoid* adalah sebagai berikut:

1. Untuk tipe pertanyaan *definition*, contoh pola yang digunakan dapat dilihat pada tabel 2.3 berikut ini.

Tabel 2.3 Contoh pola untuk pertanyaan *Definition*

Pattern
<keyword> + [is/are] + <answer>
<keyword> + [mean/means] + <answer>
<answer>(<keyword>)

2. Untuk tipe pertanyaan *reason*, contoh pola yang digunakan dapat dilihat pada tabel 2.4 berikut ini.

Tabel 2.4 Contoh pola untuk pertanyaan *reason*

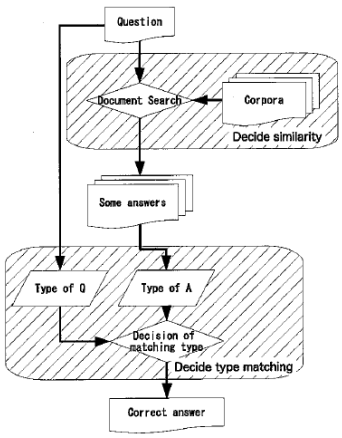
Pattern
<keyword> + [because] + <answer>
<answer> + [is reason of] + <keyword>

3. Untuk tipe pertanyaan *how*, contoh pola yang digunakan dapat dilihat pada tabel 2.5 berikut ini.

Tabel 2.5 contoh pola untuk pertanyaan *how*

Pattern
<step> + <keyword> + <answer>
[to do] + <keyword> + <answer>

Pada pendekatan pembelajaran mesin, salah satu aturan yang dapat digunakan adalah menentukan apakah tipe pertanyaan dengan tipe kandidat jawaban sama atau tidak. Jika tipe jawaban dan tipe kandidat jawaban cocok, maka kandidat jawaban itu dapat dimasukkan sebagai jawaban dari pertanyaan tersebut, dengan syarat topiknya sama. Algoritma pembelajaran mesin yang biasa dipakai adalah *Support Vector Machine* (SVM). Pencocokan tipe ini dapat dilakukan setelah masing-masing tipe dari pertanyaan dan jawaban diklasifikasikan secara terpisah atau dapat dilakukan langsung dalam satu proses. Ilustrasi dari penggunaan metode ini dapat dilihat pada gambar 2.3 berikut.



Gambar 2.3 Ilustrasi konstruksi penentuan jawaban dengan menggunakan *machine learning* [15]

Pada pendekatan analisis semantik, sistem harus diberikan “pengetahuan” terlebih dahulu berupa kalimat pernyataan. Kalimat pernyataan ini akan diproses ke dalam representasi semantik, kemudian “dicocokkan” dengan representasi semantik dari kalimat-kalimat pernyataan yang ada. Penjelasan dapat dilihat pada gambar 2.4.

```
profession(x2, wartawan).   Query(Ans) :- event(B, kurang),
complement(x3,x4).                        agent(B, C),
event(x4, naik).                          location(C, pasokan)
agent(x4, x5).                            object(D, Ans),
object(x5, harga).                        nn(C, D).
object(x6, susu).
nn(x5, x6).
event(x7, kurang).
agent(x7, x8).
object(x8, pasokan).
object(x9, susu).
object(x10, impor).
nn(x9, x10).
nn(x8, x9).
location(x11, indonesia).
di(x7, x11).
cause(x4, x7).
```

Gambar 2.4 Contoh representasi semantik [12]

2.4 *Unified Modeling Language* (UML)

Menurut Dharwiyanti [17], *Unified Modelling Language* (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML, dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa – bahasa berorientasi objek seperti C++, *Java*, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan syntax/semantik.

2.4.1 Diagram UML

UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu :

1. *Use case Diagram* untuk memodelkan proses bisnis
2. *Conceptual Diagram* untuk memodelkan konsep-konsep yang ada di dalam aplikasi
3. *Sequence Diagram* untuk memodelkan pengiriman pesan (message) antar *object*s
4. *Collaboration Diagram* untuk memodelkan interaksi antar *object*s
5. *State Diagram* untuk memodelkan perilaku *object*s di dalam sistem
6. *Activity Diagram* untuk memodelkan perilaku *Use cases* dan *Objects* di dalam sistem
7. *Class Diagram* untuk memodelkan struktur kelas
8. *Object Diagram* untuk memodelkan struktur *object*
9. *Component Diagram* untuk memodelkan komponen *object*
10. Deployment Diagram untuk memodelkan distribusi aplikasi

Menurut Dharwiyanti [17], UML menyediakan 8 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu:

1. *Use case* Diagram untuk menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”.
2. *Sequence* Diagram untuk menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa *message* yang digambarkan terhadap waktu.

3. *Collaboration* Diagram menggambarkan interaksi antar objek seperti *sequence* diagram, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian message.
4. *Statechart* Diagram untuk menggambarkan transisi dan perubahan keadaan (dari satu state ke state lainnya) suatu objek pada sistem sebagai akibat dari stimuli yang diterima.
5. *Activity* Diagram *Activity* diagrams menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir.
6. *Class* Diagram untuk menggambarkan struktur dan deskripsi *class*, package dan objek beserta hubungan satu sama lain seperti containment, pewarisan, asosiasi, dan lain-lain.
7. Component Diagram untuk menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan(dependency) di antaranya.
8. Deployment Diagram untuk menggambarkan detail bagaimana komponen dideploy dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal.

Untuk pengembangan aplikasi dalam penelitian ini akan digunakan 4 macam UML diagram, yaitu :

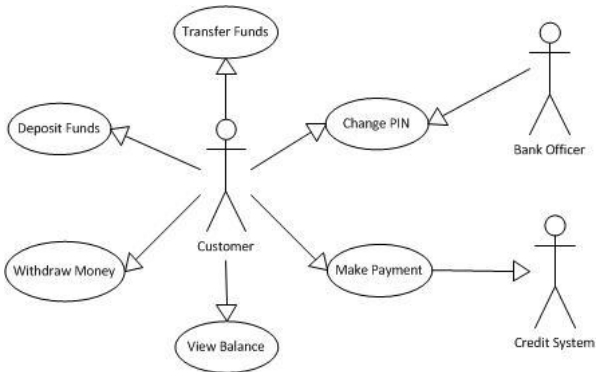
1. *Use case* Diagram
2. *Class* Diagram
3. *Sequence* Diagram
4. *Activity* Diagram

2.4.1.1 Use Case Diagram

Use case mengidentifikasi dan menjelaskan fungsi sistem dari perspektif pengguna luar dan dengan cara dan terminologi yang mereka mengerti. Secara akurat dan menyelesaikan permintaan ini, pengguna tingkat tinggi terlibat yang merupakan ahli subjek materi yang memiliki pengetahuan tentang proses *business* atau peristiwa.

Gambaran *use case* diwakili oleh sebuah elips horizontal dengan nama muncul kasus penggunaan di atas, bawah, atau di dalam elips. sebuah *use case* mewakili tujuan tunggal dari sistem yang menggambarkan urutan kegiatan dan interaksi pengguna dalam usaha mencapai tujuan. Pembuatan *use case* telah terbukti menjadi teknik yang sangat baik untuk lebih memahami dan dokumen persyaratan sistem. *Use case* sendiri adalah tidak dianggap sebagai persyaratan fungsional, tetapi cerita (skenario). *Use case* menceritakan terdiri dari satu atau lebih persyaratan.

Use case diagram terdiri atas diagram untuk proses dan aktor. Aktor merepresentasikan orang yang akan mengoperasikan atau orang yang berinteraksi dengan sistem aplikasi. Proses merepresentasikan operasi-operasi yang dilakukan oleh aktor. Proses digambarkan berbentuk elips dengan nama operasi dituliskan di dalamnya. Aktor yang melakukan operasi dihubungkan dengan garis lurus ke *use case.*

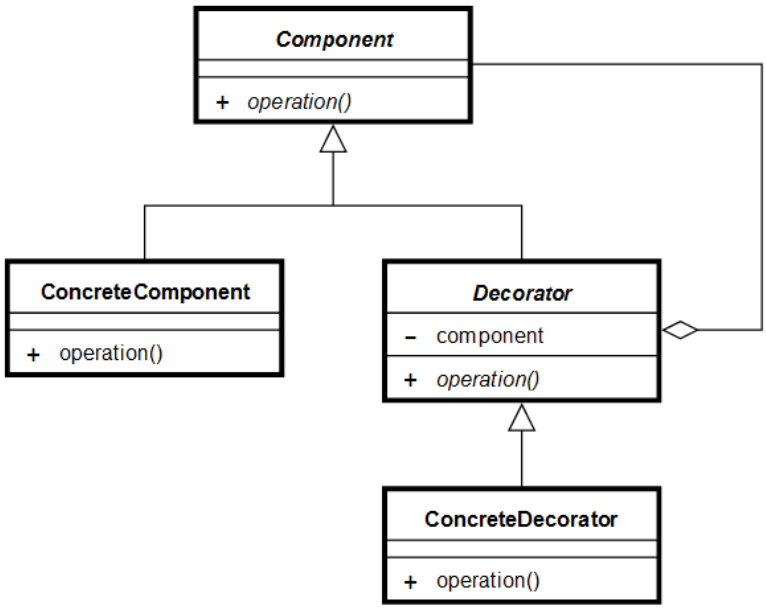


Gambar 2.6 Contoh *Use Case Diagram*

2.4.1.2 Class Diagram

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. *Class* diagram menunjukkan hubungan antar *class* dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan.

Menurut Roger S. Pressman [3], *Class* diagram digunakan untuk memodelkan kelas-kelas, termasuk atribut dan operasinya serta hubungan dan asosiasinya dengan kelas lainnya. *Class* diagram menyediakan sebuah gambaran statis atau struktural dari sebuah sistem. *Class* diagram tidak memperlihatkan komunikasi dinamis antar objek dari kelas yang ada dalam diagram.

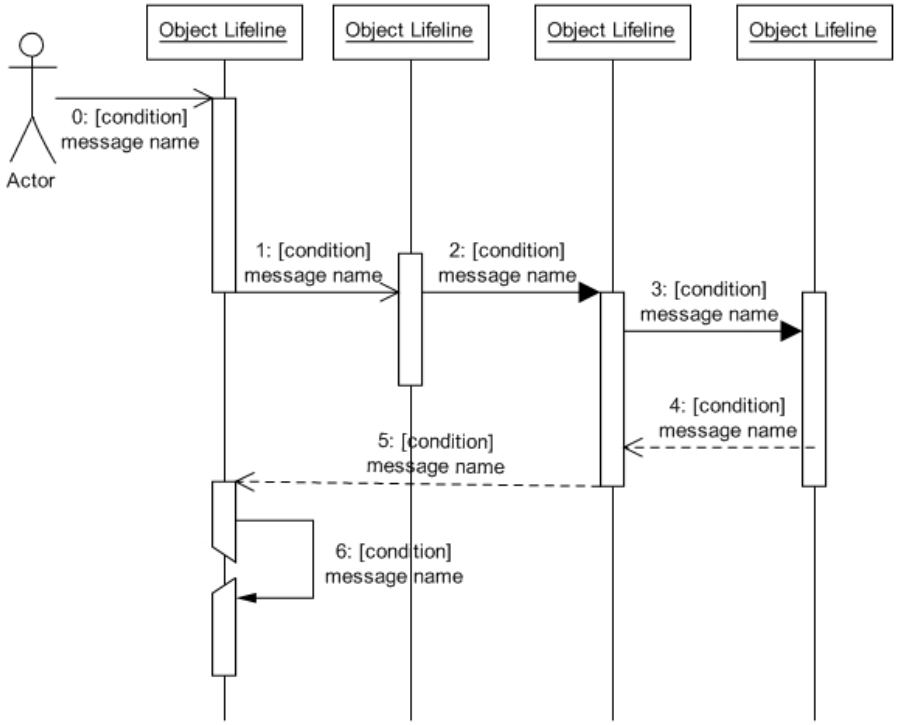


Gambar 2.7 Contoh *Class Diagram*

2.4.1.3 Sequence Diagram

Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi.

Sequence diagram adalah diagram yang menggambarkan interaksi antara aktor dan sistem untuk sebuah skenario *use case*. Sequence diagram menggambarkan bagaimana objek saling berinteraksi satu sama lain melalui messages dalam eksekusi dari sebuah *use case* atau operasi. Diagram ini menggambarkan langkah-langkah pesan dikirim dan diterima antara objek [17].



Gambar 2.8 Contoh *Sequence Diagram*

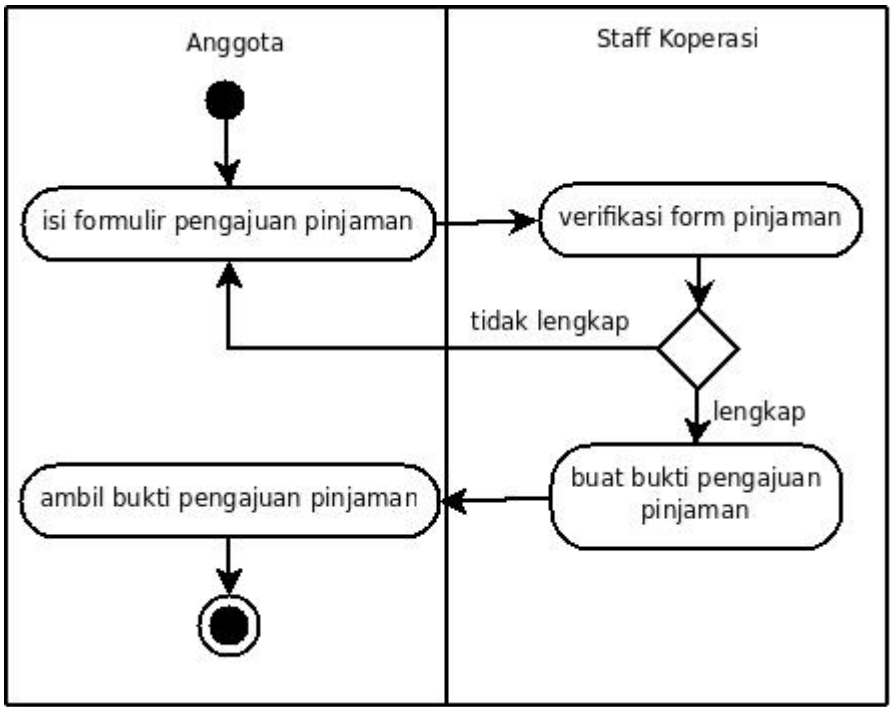
Berikut ini merupakan macam-macam notasi dari sequence diagram:

1. *Actor*: dilambangkan dengan gambar actor dari *use case*.
2. *System*: dilambangkan dengan sebuah kotak.
3. *Lifelines*: garis putus-putus vertikal, yang merupakan perpanjangan dari actor menuju dan simbol sistem, yang mengindikasikan rentetan.

4. *Activation bars*: sebuah bar yang mengindikasikan periode waktu kapan partisipan aktif di dalam interaksi.
5. *Input messages*: panah horizontal dari actor menuju sistem yang mengindikasikan pesan masuk.
6. *Output messages*: panah horizontal dari sistem menuju actor, ditunjukkan dengan panah bergaris putus-putus.

2.4.1.4 Activity Diagram

Activity diagram, secara grafis digunakan untuk menggambarkan rangkaian aliran aktivitas baik proses bisnis atau *use case*. Diagram ini juga dapat digunakan untuk memodelkan aksi yang akan dilakukan saat sebuah operasi dieksekusi, dan memodelkan hasil dari aksi tersebut.



Gambar 2.9 Contoh *Activity diagram*

Berikut ini merupakan notasi dari *Activity Diagram*:

1. *Initial node*: menunjukan permulaan dari progres, dilambangkan dengan sebuah lingkaran solid.

2. *Actions*: dilambangkan dengan sebuah kotak dengan ujung membulat merepresentasikan setiap langkah individual. Urutan dari aksi membentuk keseluruhan dari aktivitas yang ditunjukkan oleh diagram.

3. *Flow*: dilambangkan dengan panah yang mengindikasikan progres melalui *actions*. Setiap panah tidak diperlukan kata untuk mengidentifikasikannya kecuali untuk menentukan pilhan.

4. *Decision*: dilambangkan dengan lambang wajik. Terdapat satu panah yang masuk dan terdapat 2 atau lebih panah yang keluar dari pilihan wajik. Setiap panah terdapat keterangan yang menerangkan kondisi.

5. *Merge*: dilambangkan dengan lambang wajik, terdapat 2 atau lebih panah yang masuk namun hanya satu panah yang keluar.

6. *Fork*: dilambangkan dengan kotak garis warna hitam, dimana hanya satu panah yang masuk, namun terdapat dua atau lebih panah yang keluar.

7. *Join*: dilambangkan dengan kotak garis warna hitam seperti fork, namun terdapat dua atau lebih panah yang masuk dan satu panah yang keluar.

8. *Activity final*: menunjukkan akhiran dari suat progress, dilambangkan dengan suatu lingkaran solid dilingkari oleh lingkaran kosong.

2.5 Mobile Android

Android merupakan sebuah *platform* yang merupakan produk dari Open Handset Alliance yang dipimpin oleh Google. Kelompok ini terdiri dari operator selular, produsen perangkat handset, produsen komponen, solusi perangkat dan penyedia *platform* dan perusahaan pemasaran[18].

Android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware*, dan aplikasi. Android menyediakan *platform* yang terbuka bagi para pengembang untuk menciptakan aplikasi mereka. Awalnya, Google Inc. membeli Android Inc. yang merupakan pendatang baru yang membuat peranti lunak untuk ponsel/*smartphone*. Kemudian untuk

mengembangkan Android, dibentuklah Open Handset Alliance, konsorsium dari 34 perusahaan peranti keras, peranti lunak, dan telekomunikasi, termasuk Google, HTC, Intel, Motorola, Quacomm, T-*Mobile*, dan Nvidia.

Pada saat perilisan perdana Android, 5 November 2007, Android bersama Open Handset Alliance menyatakan mendukung pengembangan open source pada perangkat *mobile*. Di lain pihak, Google merilis kode-kode android di bawah lisensi Apache, sebuah lisensi perangkat lunak dan open *platform* perangkat seluler.

Sekitar september 2007 Google mengenalkan Nexus One, salah satu jenis *smartphone* yang menggunakan Android sebagai sistem operasinya. Telepon seluler ini diproduksi oleh HTC Corporation dan tersedia di pasaran pada Januari 2008. Pada 9 Desember 2008, diumumkan anggota baru yang bergabung dalam program kerja Android ARM Holdings, Atheros Communication, diproduksi oleh Asustek Computer Inc, Garmin Ltd, Softbank, Sonny Ericsson, Toshiba Corp, dan Vodafone Group Plc. Seiring pembentukan *Open Handset Alliance*, OHA mengumumkan produk perdana mereka Android, perangkat *mobile* yang merupakan modifikasi kernel Linux 2.6. Sejak Android dirilis telah dilakukan berbagai pembaruan berupa perbaikan bug dan penambahan fitur baru [18].

Versi-versi dari android:

1. Android versi 1.1

9 Maret 2009, Google merilis Android versi 1.1. Android versi ini dilengkapi dengan pembaruan estetis pada aplikasi, jam alarm, *Voice Search* (pencarian suara), pengiriman pesan dengan Gmail, dan pemberitahuan email. Android versi 1.5 (*Cupcake*) Pada pertengahan Mei 2009, Google kembali merilis telepon seluler dengan menggunakan Android dan SDK (*Software Development* Kit) dengan versi 1.5 (*Cupcake*). Terdapat beberapa pembaruan termasuk juga penambahan beberapa fitur dalam seluler versi ini yakni kemampuan merekam dan menonton video dengan modus kamera, mengunggah 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.

2. Android versi 1.6 (*Donut*)

Donut (versi 1.6) dirilis pada September dengan menampilkan proses pencarian yang lebih baik dibanding sebelumnya, penggunaan baterai indikator dan kontrol applet VPN. Fitur lainnya adalah galeri yang memungkinkan pengguna untuk memilih foto yang akan dihapus; kamera, camcorder dan galeri yang dintegrasikan; CDMA / EVDO, 802.1x, VPN, 16 Gestures, dan *Text-tospeech engine*; kemampuan dial kontak; teknologi *text to change speech* (tidak tersedia pada semua ponsel; pengadaan resolusi VWGA).

3 Android versi 2.0/2.1 (*Eclaire*)

Pada 3 Desember 2009 kembali diluncurkan ponsel Android dengan versi 2.0/2.1 (*Eclair*), perubahan yang dilakukan adalah pengoptimalan *hardware*, peningkatan Google Maps 3.1.2, perubahan *user interface* dengan *browser* baru dan dukungan HTML5, daftar kontak yang baru, dukungan *flash* untuk kamera 3,2 MP, digital Zoom, dan *Bluetooth* 2.1.

4. Android versi 2.2 (*Froyo: Frozen Yoghurt*)

Android *Froyo* yaitu android versi 2.2 diluncurkan pada tanggal 20 Mei 2010, dengan perubahan terhadap versi android sebelumnya antara lain dukungan Adobe Flash 10.1, kecepatan kinerja dan aplikasi 2 sampai 5 kali lebih cepat, intergrasi V8 *JavaScript* engine yang dipakai Google Chrome yang mempercepat kemampuan rendering pada *browser*, pemasangan aplikasi dalam *SD Card*, kemampuan WiFi Hotspot portabel, dan kemampuan auto update dalam aplikasi Android Market.

5. Android Gingerbread 2.3

Android Gingerbread diluncurkan pada tanggal 6 Desember 2010. Perubahan-perubahan umum yang didapat dari Android versi ini antara lain peningkatan kemampuan permainan (*gaming*), peningkatan fungsi *copy copy*, layar antar muka (*User interface*) didesain ulang, dukungan format video VP8 dan *Web*M, efek audio baru (*reverb, equalization, headphone virtualization* dan *bass boost*),

dukungan kemampuan *Near Field Communication* (NFC), dan dukungan jumlah kamera yang lebih dari satu.

6. Android *HoneyComb* 3.0/3.1

Android *HoneyComb* dirancang khusus untuk *tablet*, disini anda dapat melihat *tablet* yang memakai android untuk OS beserta harga *tablet*. Android versi ini mendukung ukuran layar yang lebih besar. *User interface* pada *HoneyComb* juga berbeda karena sudah didesain untuk *tablet*. *HoneyComb* juga mendukung multi prosesor dan juga akselerasi perangkat keras (*hardware*) untuk grafis. *Tablet* pertama yang dibuat dengan menjalankan *HoneyComb* adalah Motorola Xoom. Perangkat *tablet* dengan *platform* Android 3.0 akan segera hadir di Indonesia. Perangkat tersebut bernama *Eee Pad Transformer* produksi dari Asus. Rencana masuk pasar Indonesia pada Mei 2011.

7. Android *Ice Cream Sandwich* (ICS) 4.0

Android *Ice Cream Sandwich* diumumkan pada tanggal 19 Oktober 2011, membawa fitur *HoneyComb* untuk *smartphone* dan menambahkan fitur baru termasuk membuka kunci dengan pengenalan wajah, jaringan data pemantauan penggunaan dan kontrol, terpadu kontak jaringan sosial, perangkat tambahan fotografi, mencari email secara *offline*, dan berbagi informasi dengan menggunakan NFC.

8. Android versi 4.1 (*Jelly Bean*)

Sampai tulisan ini di tulis Versi Android *Jelly Bean* adalah versi Android yang terbaru. Android *Jelly Bean* yaang diluncurkan pada acara Google I/O lalu membawa sejumlah keunggulan dan fitur baru. Penambahan baru diantaranya meningkatkan input *keyboard*, desain baru fitur pencarian, UI yang baru dan pencarian melalui *Voice Search* yang lebih cepat.Tak ketinggalan Google Now juga menjadi bagian yang diperbarui. Google Now memberikan informasi yang tepat pada waktu yang tepat pula. Salah satu kemampuannya adalah dapat mengetahui informasi cuaca, lalu-lintas, ataupun hasil pertandingan olahraga.

Adapun *Features* yang tersedia pada *platform* Android saat ini antara lain:

a. *Framework* Aplikasi yang mendukung penggantian komponen dan *reusable*.
b. Mesin *Virtual* Dalvik berjalan diatas Linux kernel dan dioptimalkan untuk perangkat *mobile*.
c. Integrated *browser* berdasarkan *open source engine WebKi*t.
d. Grafis yang dioptimalkan dan didukung oleh *library* grafis 2D yang terkostumisasi, grafis 3D berdasarkan spesifikasi openGL ES 1,0 (*Opsional* akselarasi *hardware*).
e. SQLite untuk penyimpanan data.
f. Media support yang mendukung audio, video, dan gambar (MPGE, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
g. GSM *Telephony* (tergantung *hardware*).
h. *Bluetooth*, EDGE, 3G, GPS, dan WIFI (tergantung *hardware*).
i. Lingkungan *Development* yang lengkap dan kaya termasuk perangkat *emulator, tools* untuk *debugging*, profil dan kinerja memori, dan *plugin* untuk Eclips IDE.
j. Multi-*touch*, kemampuan layaknya handset modern yang dapat menggunakan dua jari atau lebih untuk berinteraksi dengan perangkat.
k. *Market*, Seperti kebanyakan *Handphone* yang memiliki tempat penjualan aplikasi, market pada android merupakan katalog aplikasi yang dapat di *download* dan di *install* pada *Handphone* melalui internet.

Ada beberapa batasan dalam pengembangan aplikasi *mobile*, yaitu.

1. Sumber daya yang terbatas.
2. Keterbatasan koneksi internet.
3. *Processor, memory* dan baterai.

2.5.1 Arsitektur Pada Android

Sistem operasi Android memiliki 4 lapisan (layer) yang merupakan komponen *system* Android. Gambar di halaman selanjutnya merupakan lapisan arsitektur *system* operasi android.



Gambar 2.10 *Overview* dari arsitektur Android [19]

Google mengibaratkan Android sebagai sebuah tumpukan *software*. Setiap lapisan dari tumpukan ini menghimpun beberapa program yang mendukung fungsi-fungsi spesifik dari *system* operasi. Berikut ini susunan dari lapisan – lapisan tersebut dilihat dari lapisan dasar hingga lapisan teratas.

2.5.1.1 Linux kernel

Tumpukan paling bawah pada arsitektur Android ini adalah kernel. Google menggunakan kernel Linux versi 2.6 untuk membangun *system* Android, yang mencakup memory management, security setting, power management dan beberapa driver *hardware*. Kernel berperan sebagai abstr*action* layer antara *hardware* dan keseluruhan *software*. Sebagai contoh : HTC GI dilengkapi dengan kamera. Kernel Android terdapat driver kamera yang memungkinkan pengguna mengirimkan perintah kepada *hardware* kamera.

Android dibangun di atas kernel Linux 2.6. Namun secara keseluruhan android bukanlah linux, karena dalam android tidak terdapat paket standar yang dimiliki oleh linux lainnya. Linux merupakan sistem operasi terbuka yang handal

dalam manajemen memori dan proses. Oleh karenanya pada android hanya terdapat beberapa servis yang diperlukan seperti keamanan, manajemen memori, manajemen proses, jaringan dan driver. Kernel linux menyediakan *driver* layar, kamera, WiFi, *Flash Memory*, *audio* dan IPC (*Interprocess Communication*) untuk mengatur aplikasi dan lubang keamanan.

2.5.1.2 Libraries

Bertempat di level yang sama dengan android runtime adalah libraries. Android menyertakan satu set *library-library* dalam bahasan C/C++ yang digunakan oleh berbagai komponen yang ada pada sistem android. Kemampuan ini dapat diakses oleh programmer melewati android *application* framework. Sebagai contoh android mendukung pemutaran format audio, *video* dan gambar.

Berikut beberapa core *library* tersebut:

1. *System* C *Library* – variasi dari implemetasi standard C sistem *library* (libc) milik BSD, dioptimasi untuk piranti *embedded* berbasis linux.

2. Media *Libraries* – berdasarkan packet *video*‟s openCORE, *library-library* ini mendukung *playback* dan *recording* dari berbagai format audio dan *video* popular, meliputi MPEG4, H.264, MP3, AAC, AMR, JPG, dan PNG.

3. *Surface Manager* – Mengelola atau mengatur akses pada *display* dan lapisan composites 2D dan grafis 3D dari berbagai aplikasi.

4. *LibWebCore* – web *browser engine* modern yang mensupport android *browser* maupun *embeddable* web *view*.

5. SGL – Mendasari grafis 2D

6. 3D *libraries* – implementasi berdasarkan OpenGL ES 1.0 APIs, *library* ini menggunakan perangkat keras akselerasi 3D dan pengoptimalan 3D *software* rasterizer.

7. *FreeType* – bitmap dan *vector font rendering*

8. SQLite – mesin *database* yang kuat dan ringan tersedia untuk semua aplikasi.

Library - *library* tersebut bukanlah aplikasi yang berjalan sendiri, namun hanya dapat digunakan oleh program yang berada di level atasnya. Sejak versi android 1.5 pengembang dapat membuat dan menggunakan pustaka sendiri menggunakan NDK (Native Development Toolkit).

2.5.1.3 Android Runtime

Lapisan berikutnya adalah android runtime yang berisi core libraries dan dalvik virtual machine. Core libraries mencakup serangkaian inti *library java*, artinya android menyertakan satu set *library-library* dasar yang menyediakan sebagian besar fungsi-fungsi yang ada pada *library-library* dasar bahasa pemprograman *java*. Dalvik adalah *java* virtual machine yang memberikan kekuatan pada sistem android. Dalvik VM ini di optimalkan untuk telepon seluler. Setiao aplikasi yang berjalan pada android berjalan pada prosesnya sendiri, dengan instance dari dalvik virtual machine. Dalvik telah dibuat sehingga sebuah piranti yang memakainya dapat menjalankan multi virtual machine dengan efisien. Dalvik VM dapat mengeksekusi file dengan format .dex (Dalvik Executable) yang telah dioptimasi untuk menggunakan minimal memory footprint. Virtual machine ini register-based, dan menjalankan *class-class* yang decompile menggunakan "dx" tool yang telah disertakan. Dalvik virtual machine menggunakan kernel linux untuk menjalankan fungsi – fungsi seperti threading dan low level memory management.

2.5.1.4 Framework Aplikasi

Lapisan selanjutnya adalah framework aplikasi, yang mencakup program untuk mengatur fungsi – fungsi dasar smartphone. *Application* framework merupakan serangkaian tool dasar seperti alokasi resources smartphone, aplikasi telepon, pergantian antar proses atau program dan pelacakan lokasi fisik telepon. Para pengembang aplikasi memiliki aplikasi penuh kepada tool – tool dasar tersebut dan memanfaatkannya untuk menciptakan aplikasi yang lebih kompleks.

Programmer mendapatkan akses penuh untuk memanfaatkan API (Android Protocol Interface) yang juga digunakan core *applications*. Arsitektur aplikasi didesain untuk menyederhanakan pemakaian kembali komponen – komponen, dimana setiap aplikasi dapat menunjukkan kemampuannya dan aplikasi lain dapat memakai kemampuan tersebut. Mekanisme yang sama memungkinkan pengguna mengganti komponen yang dikehendaki. Di dalam semua aplikasi terdapat servis dan sistem yang meliputi :

a. Satu *set Views* yang dapat digunakan untuk membangun aplikasi meliputi *lists, grids, text boxes, buttons,* dan *embeddable web browser*.

b. *Content Providers* yang memungkinkan aplikasi untuk mengakses data dari aplikasi lain misalnya *contact* atau membagi data yang dimiliki.

c. *Resources Manager*, menyediakan akses ke *non-code resources* misalnya *localized strings, graphics* dan *layout files.*

d. *Notification Manager* yang memungkinkan semua aplikasi untuk menampilkan *custom alerts* pada status bar.

e. *Activity Manager* yang mengatur *life cycle* dari aplikasi dan menyediakan *common navigation backstack*.

2.5.1.5 Applications

Di lapisan teratas terdapat aplikasi itu sendiri. Di lapisan inilah dapat ditemukan fungsi-fungsi dasar smartphone seperti menelepon dan mengirim pesan singkat, menjalankan web browser, mengakses daftar kontak, dan lain-lain. Sebagian penggunan pada lapisan inilah yang paling sering diakses, dimana pengaksesan fungsi-fungsi dasar tersebut melalui user interface.

2.5.2 XML pada Android

XML (*Extensible Markup Language*) adalah bahasa *markup* yang mendefinisikan seperangkat aturan untuk encoding dalam dokumen yang bersifat *human-readable* dan *machine-readable*. Pada android sendiri berkas berjenis XML memiliki banyak fungsi, diantaranya :

1. Berkas Android Manifest

   Setiap aplikasi android harus memiliki berkas ini (dengan nama AndroidManifest.xml) pada direktori root. Pada berkas ini berisi informasi penting tentang aplikasi untuk sistem android.

2. *Layout*

   Android menyediakan kosakata XML sederhana yang sesuai dengan kelas View dan sub*class*-nya, seperti untuk widget dan *layout*.

3. *Database*

   Database XML adalah persistensi data perangkat lunak sistem yang memungkinkan data yang akan disimpan dalam format XML. Data ini kemudian dapat dilihat, diekspor dan serialisasi ke dalam format yang diinginkan. XML *database* biasanya berhubungan dengan dokumen berorientasi *database*.

2.6 Java

Menurut definisi yang diberikan oleh Sun MicroSystem, JAVA adalah bahasa pemrograman yang *simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded,* dan *dynamic.*

Java bahasa yang *simple:* Rancangan bahasa java dibuat sedemikian rupa sehingga dengan cepat dapat dikenali dan dipahami oleh pemrogram pemula sekalipun karena mirip dengan bahasa C/C++ dan yang kedua lebih sederhana dari C/C++ dalam beberapa hal yang sulit pada C/C++ sudah tidak ada pada Java, misalnya pemakaian pointer.

Java adalah bahasa yang *object oriented*: Konsep objek diterapkan pada pembuatan program Java, bahkan java menggunakan konsep objek secara murni.

Java adalah bahasa yang *distributed*: Bahasa Java dirancang sedemikian rupa sehingga mendukung pembuatan aplikasi dalam jaringan computer.

Java adalah bahasa yang *robust*: rancangan java dibuat sedemikian rupa hingga mengurangi kemungkinan menjadi beku *(freeze)* ketika dijalankan.

Java adalah bahasa yang s*ecure*: secara otomatis java menerapkan pengamanan terhadap aplikasi sehingga mengurangi kemungkinan terjadinya serangan dari pengguna jaringan.

Java adalah bahasa yang *architecture neutral*: program java yang telah dikompilasi dapat dijalankan

Java adalah bahasa yang *portable*: program java sangat mudah dipindahkan dari satu mesin ke mesin lain tanpa harus dikompilasi ulang.

Java adalah bahasa yang h*igh-performance*: pada hakikatnya program C yang telah dikompilasi berjalan jauh lebih cepat dari Java, namun hasil kompilasi ini belum tentu bisa jalan pada computer yang berbeda arsitektur.Program java dikompilasi menjadi *bytecode* yang bisa jalan diberbagai mesin, namun ketika dijalankan *bytecode* ini harus diterjemahkan oleh JVM ke dalam bahasa mesin yang sesuai, sehingga lebih lambat dari C, namun karena java dapat digunakan pada jaringan computer yang kenyataannya tidak terlalu cepat karena hambatan saluran komunikasi maka program Java masih terasa cukup cepat.

Java adalah bahasa yang m*ultithreaded*: Java dirancang untuk menangani berbagai aplikasi yang berjalan secara bersamaan, misalnya memainkan lagu, sambil melakukan *download*. Kejadian ini disebut dengan *multithreaded*.

Java adalah bahasa yang d*ynamic*: bahasa java dirancang untuk beradaptasi dengan lingkungan yang dapat berubah dengan cepat[20].

2.7 Database (Basis data)

Database adalah kumpulan file-file yang saling berelasi sehingga membentuk satu bangunan data, relasi tersebut biasa ditunjukan dengan kunci dari tiap file yang ada. Satu *database* menunjukkan satu kumpulan data yang dipakai dalam lingkup perusahaan, instansi. Bila terdapat file yang tidak dapat dipadukan atau dihubungkan dengan file lainnya berarti file tersebut bukanlah kelompok dari satu *database*, ia akan membentuk satu *database* sendiri [21] .

Komponen dalam *database*:

1. *Entity*: orang, tempat, kejadian atau konsep yang informasinya direkam. Pada bidang Administrasi Siswa misal entitynya adalah siswa, buku, pembayaran, dan nilai test.
2. *Attribute*: Setiap *entity* mempunyai attribute atau sebutan untuk mewakili suatu *entity*. Seorang siswa dapat dilihat dari attributenya, misalnya nama, nomor siswa, alamat. *Attribute* juga disebut juga sebagai data elemen, data *field*, atau data *item*.
3. *Data Value* (nilai atau isi data): data aktual atau informasi yang disimpan pada tiap data elemen atau *attribute*.
4. *Record/tuple:* kumpulan elemen yang saling berkaitan menginformasikan tentang suatu *entity* secara lengkap.
5. *File:* Kumpulan *record* sejenis yang mempunyai panjang elemen yang sama, *attribute* yang sama, namun berbeda beda data *value*nya.

2.7.1 Entity Relationship Diagram (ERD)

Pemodelan awal basis data yang paling banyak digunakan adalah menggunakan ERD. ERD ang digunakan untuk menggambarkan data dalam bentuk entitas, atribut dan hubungan antarentitas [22]. Model ini dinyatakan dalam bentuk diagram. Perlu diketahui bahwa ERD tidak mencerminkan bentuk fisik yang nantinya akan disimpan dalam *database*, melainkan hanya bersifat konseptual.

2.8 Object Oriented Programming (OOP)

Pemrograman berorientasi objek merupakan pemrograman yang menjadikan objek sebagai komponen utama dalam sistem. Pemrograman berorientasi objek merupakan satu ide luar biasa dalam bidang pemrograman yang setelah diterapkan ternyata melahirkan bahasa yang lebih maju seperti C++, Java, Phyton, dsb. Perkembangan dari *OOP* inipun melahirkan konsep pemrograman yang lain berbasis Windows atau GUI *(Graphical User Intarface)* yang kemudian popular sebai bahasa visual, seperti Visual Basic, Delphi, Visual C, Visual Java, dsb[23].

2.8.1 Konsep Objek

Objek dalam pengertian sehari-hari adalah benda, baik yang secara fisik dapat diketahui keberadaannya maupun yang bersifat khayal (*virtual*). Contohnya, kamera, kalkulator, ponsel, ide, gagasan, dan sebagainya.

Objek dalam pemrograman *(soft object)* adalah suatu komponen dari suatu sistem yang dibangun, misalnya objek matematis, objek jendela *(windows),* objek tombol, objek kotak gambar, objek kotak teks, dsb [20].

2.8.2 Konsep Kelas

Setiap objek memiliki suatu kelompok. Kelompok ini dalam konsep OOP disebut "kelas" atau "*class*". Misalnya Toyota, Daihatsu, dan Honda adalah termasuk kelompok "mobil". Kelas adalah pemodelan dari objek yang berisi informasi (aturan) tentang sifat karakteristik (data) dan tingkah laku (method) yang dimiliki oleh objek tersebut [20].

Dalam suatu kelas terdapat dua hal: sifat-sifat *(attribute/field/property)* dan fungsi *(method)* atau kelakuan *(behavior).* Misalnya kelas mobil memiliki atribut warna *body*, jumlah tempat duduk, ukuran ban, dsb, kemudian terdapat fungsi seperti pengapian, pengisian aki, percepatan *(accelerator)*, rem, dsb. Pada tingkat pemrograman suatu atribut bisa berupa definisi suatu varabel/field atau definisi fungsi, sedangkan kelakuan bisa berupa implementasi dari suatu fungsi.

Sifat-sifat dari suatu kelas bisa diwariskan *(inheritance)* ke sub-kelasnya, misal sub-kelas mobil sedan mewarisi atribut dari kelas mobil. Fungsi-fungsi kelas juga dapat diwariskan bilamana perlu.

2.8.3 Kelas Konstruktor

Konstruktor adalah suatu fungsi khusus yang diberi nama sama dengan nama kelasnya. Pemberian nilai pada atribut kelas dapat dilakukan pada saat objek didefinisikan melalui kelas konstruktor *(class constructor)*.

2.8.4 Enkapsulasi

Enkapsulasi *(encapsulation)* adalah pengemasan data dan prosedur dalam objek. Pengkapsulan merupakan upaya penyembunyian informasi yang diterapkan

pada sehingga data tidak dapat diakses secara langsung oleh pihak di luar objek [24]. Akses terhadap kelas bergantung pada *modifier* yang diberikan pada kelas dan isinya. Ada tiga macam modifier yang biasa digunakan, yaitu:

1. *Public*, berarti dapat diakses secara bebas baik dari dalam maupun dari luar kelas.
2. *Private*, berarti akses sangat terbatas, hanya dari dalam kelas itu sendiri, tidak bisa dari luar.
3. *Protected*, berarti akses terbatas, hanya diberikan pada sub-kelas, dan dari dalam kelas.

Enkapsulasi memberi jaminan pada kelas objek agar tidak diganggu oleh pemrogram kecuali yang disediakan untuknya. Dengan demikian program akan lebih bersih dari gangguan meupun kesalahan, termasuk kesalahan dalam pemakaian objek. Contoh enkapsulasi:

```
class belajar{ public String x =”Pintar”; private
String y = “Java”;
protected String z = “OOP”; }
```

2.8.5 Pewarisan

Peristiwa pemakaian kelas objek dengan parameter-parameter lokal yang sesuai dengan definisi antarmuka objek dapat dipandang sebagai pewarisan *(inheritance)* sifat-sifat objek ke program klien. Dengan kata lain pewarisan merupakan sifat dalam bahasa berorientasi objek yang memungkinkan sifat-sifat dari suatu kelas diturunkan ke kelas lain [8]. Sebagai contoh disediakan objek bingkai atau frame dengan antarmuka seperti *height* (tinggi) dan *width* (lebar).

Dalam hal ini program klien dapat memanggil obejk frame dan men-set tinggi dan lebarnya sesuai kebutuhan. Inheritance berarti pewarisan pada sub-kelas dari suatu kelas, yang bisa dilaksanakan melalui kata kunci *extends.*

Contoh pewarisan:

```
public class binatang{
protected String makanan;
public void makan(){
System.out.println("makan".+this.makanan);}
}
Public class singa extends binatang{
Public void daging(){
System.out.println(this.makanan + "daging"); }
}
```

2.8.6 Polimorfisme

Polimorfisme adalah sifat dari kelas objek yang memungkinkan beberapa fungsi dapat memiliki nama yang sama walaupun isi fungsi tersebut berbeda. Sebagai contoh, fungsi konstruktor suatu kelas *circle* tetapi dengan parameter berbeda, *circle* dengan parameter x, *circle* dengan parmeter x,y, dan *circle* dengan parameter x,y,z. Jadi setiap kelas mempunyai parameter yang berbeda dan dapat digunakan sesuai kebutuhannya masing-masing.

2.9 Tools Pendukung Pembuatan Aplikasi Android

2.9.1 Java Development Kit (JDK)

JDK adalah *software* development kit merupakan seperangkat kode, *library*, form dan paket yang berguna untuk membuat aplikasi atau applet *java*. Di dalam JDK terdapat *Java* Runtime Environment (JRE) yang akan memproses byte code yang dihasilkan dari interpenter *java*. Tugas dari interpenter *java* ini adalah untuk mengcompile source.*java* menjadi source.*class* yang selanjutnya source.*class* inilah yang akan dijalankan oleh JRE.

2.9.2 Software Development Kit (SDK)

SDK adalah suatu kumpulan dari *library* dan toolkit seperti emulator untuk mengembangkan atau menciptakan aplikasi untuk suatu perangkat lunak, perangkat computer, *system* operasi atau platform. Di dalam SDK terdapat tools yang dibutuhkan dalam pengembangan android, diantaranya yaitu adb shell yang merupakan singkatan dari android development bridge yang dapat menjalankan

terminal android seperti terminal pada *system* operasi linux, dan command yang terdapat dalam adb shell sendiri seperti command linux pada umumnya.

2.9.3 Eclipse

Eclipse adalah sebuah Integrated Development Environment (IDE) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua platform (platform-independent). Berikut ini adalah sifat dari Eclipse :

1. Multi-platform : Target sistem operasi eclipse adalah Microsoft Windows, Linux, Solaris, AIX, HP-UX dan Mac OS X.

2. Mulit-language: Eclipse dikembangkan dengan bahasa pemrograman *Java*, akan tetapi Eclipse mendukung pengembangan aplikasi berbasis bahasa pemrograman lainnya, seperti C/C++, Cobol, Python, Perl, PHP, dan lain sebagainya.

3. Multi-role: Selain sebagai IDE untuk pengembangan aplikasi, Eclipse pun bisa digunakan untuk aktivitas seperti dokumentasi, test perangkat lunak, pengembangan web, dan lain sebagainya.

Secara standar Eclipse selalu dilengkapi dengan *Java* Development Tools (JDE), plug-in yang membuat Eclipse kompatibel untuk mengembangkan program *Java*, dan Plug-in Development Environment (PDE) untuk 27 mengembangkan plug-in baru. Eclipse beserta plug-in-nya diimplementasikan dalam bahasa pemrograman *Java*. Konsep Eclipse adalah IDE yang terbuka (open), mudah diperluas (extensible) untuk apa saja, dan tidak untuk sesuatu yang spesifik. Apabila ingin mengembangkan program C/C++ terdapat plug-in CDT (C/C++ Development Tools).

2.9.4 Android Development Tools (ADT)

Android Development Tools (ADT) adalah *plugin* untuk Eclipse *Intergrated Development Environment* (IDE) yang dirancang untuk memberikan lingkungan yang terpadu di mana untuk membangun aplikasi Android. ADT memperluas kemampuan Eclipse untuk membiarkan para developer lebih cepat dalam

membuat proyek baru Android, membuat aplikasi UI, menambahkan komponen berdasarkan Android *Framework* API, debug aplikasi dalam pengunaan Android SDK, dan membuat file APK untuk mendistribusikan aplikasi. Mengembangkan aplikasi di Eclipse dengan ADT sangat dianjurkan dan merupakan cara tercepat untuk memulai membuat aplikasi android, karena banyak kemudahan-kemudahan sebagai *tools* yang terintegrasi seperti, *custom* XML editor, dan *debug panel ouput*. Selain itu ADT memberikan dorongan luar biasa dalam mengembangkan aplikasi Android.