# BAB II
# DASAR TEORI

## 2.1 Pemintas (*shortcut*)

Pemintas atau istilah asing dikenal dengan kata *shortcut* pada *smartphone* merupakan sebuah tombol aplikasi yang menyerupai tombol aplikasi aslinya yang dipasang di halaman depan (*Home Screen*). Tujuannya memudahkan pengguna untuk mengakses aplikasi yang dituju tanpa harus masuk kedalam menu aplikasi. Berikut adalah gambar tampilan dari menu aplikasi pada sebuah *smartphone*:




Gambar 2.1. Tampilan menu

Untuk mengakses tombol-tombol aplikasi seperti yang ditampilkan pada gambar di atas yaitu tombol aplikasi Email, *Camera*, Maps dan lain - lain pengguna diharuskan masuk kedalam menu aplikasi, dan memilih aplikasi mana yang ingin diakses. Aplikasi – aplikasi yang sering diakses pengguna biasanya akan dibuat sebuah pemintasnya pada halaman depan.

Berikut merupakan tampilan halaman depan (*home screen*) pada sebuah *smartphone* :




Gambar 2.2. tampilan halaman depan

Tombol – tombol yang sering diakses pengguna cenderung akan ditampilkan di halaman depan. Seperti pada gambar 2.2 pemintas Email, *Camera*, Maps dan lain – lain merupakan contoh aplikasi yang sering diakses pengguna.

## 2.2 Sistem Operasi Android

Android adalah sebuah sistem operasi berbasis Linux yang dirancang untuk *mobile device* seperti *smartphone* dan *computer tablet* yang dikembangkan oleh Google. Android merupakan software berbasis kode komputer yang bisa didistribusikan secara terbuka (*open source*) sehingga programmer bisa membuat aplikasi baru di dalamnya.




Gambar 2.3. Logo Android

Beberapa fitur Android antara lain :

1. ***Application framework,*** yakni digunakan untuk membangun aplikasi Android.
2. ***Integrated Browser,*** Android menyertakan *browser* berbasis WebKit sebagai aplikasi standar.
3. ***Optimized graphics,*** Android mempunyai pustaka grafik 2D dan menyertakan pustaka grafik 3D OpenGL ES.
4. ***SQLite,*** adalah aplikasi basis data SQLite yang disertakan dalam Android.
5. ***Media Support,*** dukungan untuk memutar format multimedia yang banyak.
6. ***GSM telephony support,*** adalah kemampuan Android untuk mengakses langsung *hardware* untuk komunikasi GSM.
7. ***Bluetooth, EDGE, 3G,*** **dan** ***WiFi,*** dukungan untuk banyak jenis koneksi *wireless*.
8. ***Camera, GPS, compass, NFC*** **dan** ***accelerometer,*** dukungan untuk *hardware* tersebut, tersedia API untuk mengakses *hardware* tersebut.
9. ***Rich development environment,*** tersedia *software development* yang lengkap.
10. ***Dalvik virtual machine,*** tiap aplikasi pada Android memiliki *instance virtual machine* yang dapat bekerja secara efisien dalam lingkungn memori yang terbatas.

### 2.2.1 Versi Android

Versi dari Android disini hanya membahas tentang versi apa saja yang telah diliris resmi oleh Google. berikut adalah rangkuman dari versi tersebut.

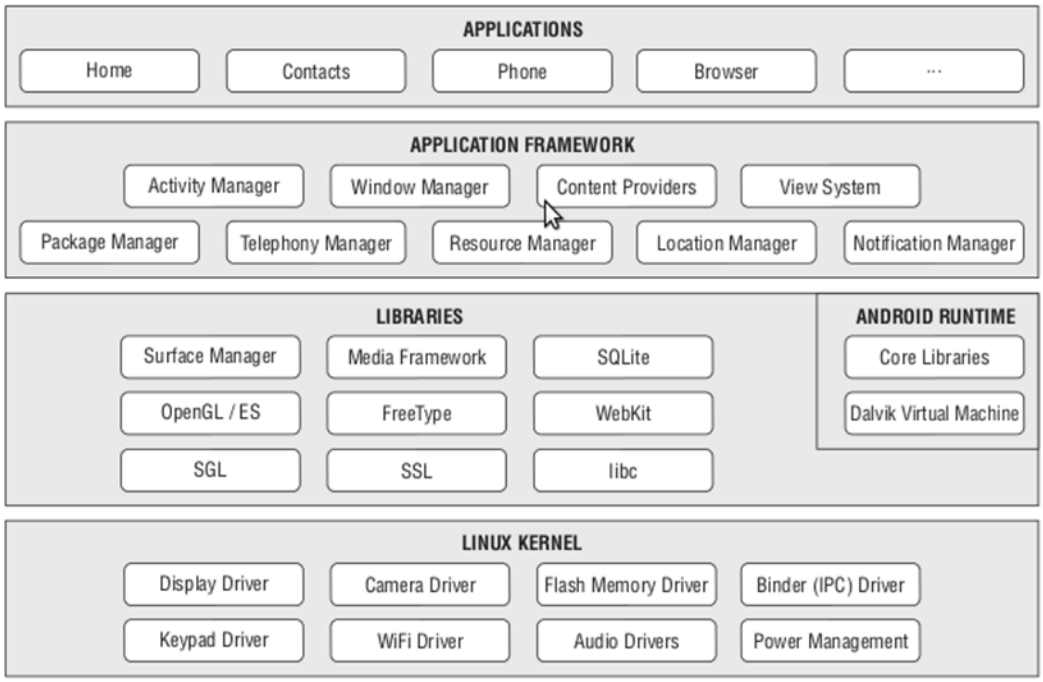
1. Android versi 1.1 (*Codename* tidak diketahui), dirilis 9 Maret 2009
2. Android versi 1.5 (*Codename* Cupcake), dirilis Pertengahan Mei 2009
3. Android versi 1.6 (*Codename* Donut), dirilis September 2009
4. Android versi 2.0/2.1 (*Codename* Eclair), dirilis 3 Desember 2009
5. Android versi 2.2 (*Codename* Froyo: Frozen Yoghurt), dirilis 20 Mei 2010

6. Android versi 2.3 (*Codename* Gingerbread), dirilis 6 Desember 2010
7. Android versi 3.0/3.1 (*Codename* Honeycomb), dirilis Mei 2011
8. Android versi 4.0 (*Codename* ICS: Ice Scream Sandwich), dirilis 19 Oktober 2011
9. Android versi 4.1 (*Codename* Jelly Bean) API level 16
10. Android versi 4.3 (jelly Bean) dengan API level 18 dirilis 24 Juli 2013 dan
11. Android versi 4.4 (Kitkat) dengan Api level 19 (tertinggi) dirilis 31 Oktober 2013.

Perbedaan versi android dari versi satu ke versi lainnya seiring dengan penambahan fitur dan API nya.

### 2.2.2 Arsitektur Android

Berikut merupakan gambaran mengenai arsitektur sistem operasi Android yang terbagi menjadi lima bagian pada empat lapisan utama.




Gambar 2.4. Arsitektur Android

Penjelasan tiap lapisan :

1. *Kernel* Linux : merupakan inti dari sistem operasi Android. Lapisan ini meliputi seluruh *driver* pada tingkatan *device* untuk berbagai jenis komponen *hardware* dari perangkat Android.
2. *Libraries* : Meliputi seluruh kode yang merupakan fitur utama dari sistem Android. Contohnya *Library* SQLite yang merupakan *library* untuk menyediakan layanan *database* sehingga aplikasi dapat menggunakannya sebagai tempat penyimpanan data. Contoh lainnya *Library* Webkit yang menyediakan segala fungsi yang berkaitan dengan *web browsing.*
3. Android *Runtime* : Berada pada lapisan yang sama sebagai *library*. Android *runtime* menyediakan inti *libraries* yang memungkinkan pengembang untuk menulis aplikasi Android dengan bahasa pemrograman Java. Android *runtime* juga menyertakan Dalvik *virtual machine* (aplikasi Android dikompilasi kedalam file Dalvik yang dapat dieksekusi). Dalvik merupakan mesin virtual spesial yang dirancang secara khusus untuk Android dan dioptimalkan untuk perangkat *mobile* sumber batere dengan memory dan CPU yang terbatas.
4. *Application Framework* : Memperlihatkan berbagai kemampuan dari sistem Android. Disediakan untuk pengembang aplikasi, sehingga dapat digunakan dalam aplikasi yang dibuat.
5. *Application* : Merupakan lapisan teratas. Segala macam aplikasi yang dibuat akan ditempatkan pada lapisan ini.

### 2.2.3 Struktur Aplikasi Android

Arsitektur Android mengadopsi konsep *component reuse* yang memungkinkan untuk mempublikasikan dan berbagai *activities, service* dan data dengan aplikasi lain, dengan batasan akses kelola. Berikut layanan aplikasi yang merupakan landasan arsitektur dari semua aplikasi Android :

1. *Activity Manager* : mengontrol siklus hidup suatu Aktivitas, termasuk manajemen tumpukan aktivitas

2. *Views* :Digunakan untuk membuat antarmuka *user* untuk aktivitas yang dibuat.
3. *Notification Manager* : Menyediakan mekanisme konsisten dan *noninstrusive* yang memberi peringatan kepada *user.*
4. *Content Providers* : Membiarkan aplikasi untuk berbagi data.
5. *Resource Manager* : Mendukung non-kode sumber seperti *string* dan grafis sebagai *tool* eksternal.

## 2.3 Android SDK

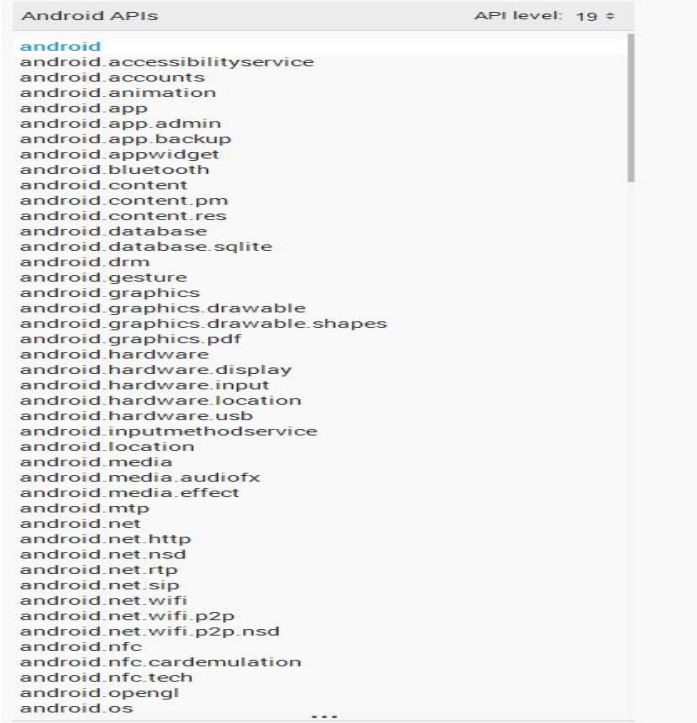
Android SDK (*Software Development Kit*) menyediakan *library* API (*Application Programming Interface*) dan *development tools* yang diperlukan untuk membangun, menguji dan *debug* untuk aplikasi Android.

Bagi pengembang aplikasi Android akan memerlukan ADT (Android *Developer Tools*). Dan google selaku pengembang sistem operasi Android sudah menyediakan ADT *Bundle* yang memudahkan pengembang aplikasi Android. ADT *Bundle* terdiri dari :

1. Eclipse + ADT *plugin*
2. Android SDK *tools*
3. Android *Platforms-tools*

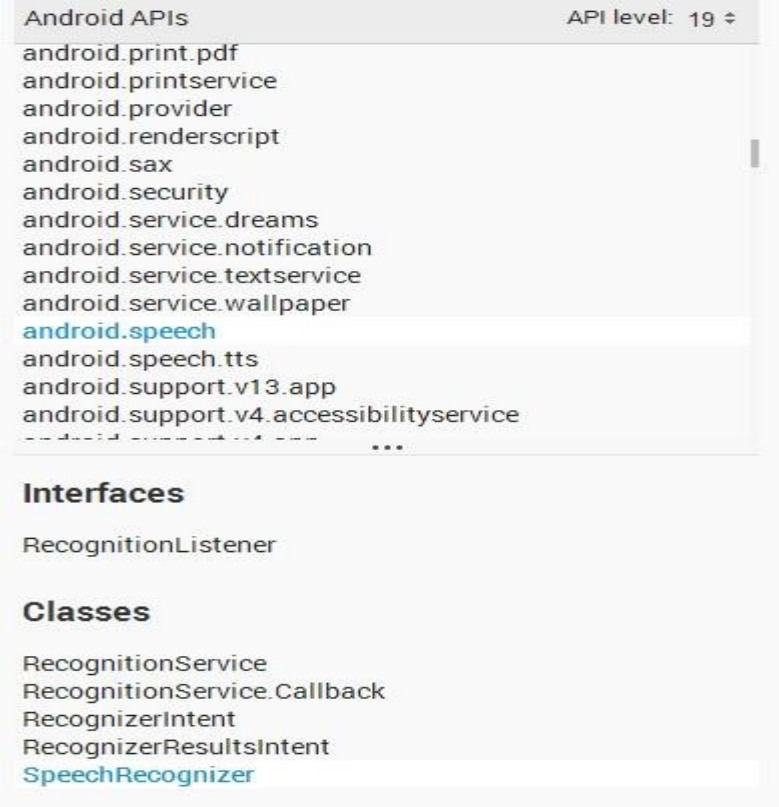
### 2.3.1 Android API

Pada pemrograman, sebuah API menentukan bagaimana komponen perangkat lunak harus berinteraksi satu sama lain. Pada praktiknya API berbentuk *library* meliputi *data structure, object class, routine* dan variabel. API dapat digunakan untuk mempermudah pemrograman. Pada sistem operasi Android API berupa *package* yang disiapkan, berisi *class – class* untuk mendukung pemrograman aplikasi Android. Berikut gambaran mengenai API pada Sistem operasi Android :




Gambar 2.5. Android API

Dalam sebuah API ada *class* – *class* yang lebih spesifik fungsinya untuk membantu *programmer.* Berikut contoh gambaran *class* dari API *android.speech* :




2.6. Contoh Class

Pada gambar 2.6. API *android.speech* terdiri dari beberapa *classes* yaitu

1. *RecognitionService*
2. *RecognitionService.Callback*
3. *Recognizerintent*
4. *RecognizerResultIntent*
5. *SpeechRecognizer*

Saat ini versi Android 4.4 (Kitkat) sudah memiliki fitur Android API sampai level 19. API utama yang diperlukan penulis untuk aplikasi yang akan dibuat yaitu API *android.speech* dan API *android.content.pm*.

## 2.4 Speech Recognition

*Speech Recognition* ialah pengenalan suara namun berbeda dengan *Voice Recognition. Speech Recognition* merupakan proses menterjemahkan suara atau ucapan yang dilontarkan dan diterjemahkan kedalam bentuk teks. Bisa juga disebut *Voice to Text*. Pada saat seseorang berbicara, proses *Voice Recognition* selain dapat mengenali ucapan juga dapat membedakan karakter suara orang tersebut, bahkan dapat mengenali suara siapa. Namun proses *Speech Recognition* hanya bisa mengenali apa yang diucapkan dan diterjemahkan dalam bentuk teks. Jadi intinya *Speech Recognition* adalah bagian dari *Voice Recognition*.

*Speech Recognition* bekerja pada level *middleware*, artinya bekerja di dalam sistem operasinya. Pada sistem operasi Android fitur *Speech Recognition* telah disematkan google pada Android 4.1 (Jelly Bean generasi pertama).

*Speech Recognition* pada Android awal mulanya difungsikan sebagai *search engine* milik google yaitu Google *search*, dengan menggunakan perintah suara untuk membantu memudahkan pengguna mencari kata kunci pada *browser* tanpa harus mengetik. Pada saat tugas akhir ini dibuat fitur *Speech Recognition* pada google selain bisa sebagai *search engine* juga sebagai perintah set alarm, perintah pencarian lokasi pada map atau peta, dan perintah informasi cuaca serta informasi *traffic*. Semua fungsi tersebut memerlukan koneksi internet melalui sistem *cloud* layanan Google bernama Google Now.

### 2.4.1 Layanan Google Now

Google Now merupakan sebuah layanan *cloud* dari perusahaan internet bernama Google, dimana Google Now ini difungsikan sebagai *Intelligent Personal Assistant*. Pada mulanya layanan Google Now ini diterapkan pada sistem *search engine*. Dimana pengguna yang ingin mencari kata kunci pada sebuah *search engine*, maka pengguna mengucapkan kata kunci tersebut lalu dikenal oleh *speech recognition engine* yang ada pada *cloud* Google Now. Layanan Google NOW baru diterapkan dengan baik pada Android 4.1.

Ketika kita memanfaatkan layanan Google Now pada Android, perintah suara yang kita ucapkan, akan langsung dikodekan oleh sistem operasi android

dan dikirim ke internet. Disini kode yang dikirim akan diolah dan dikenali polanya sebagai sebuah kata atau kalimat yang diucapkan dan dikirim kembali berupa teks.

Pada sistem operasi Android untuk menggunakan fitur S*peech Recognition* dan bagaimana agar dapat terhubung dengan layanan Google Now tentu memerlukan sebuah API. API tersebut ialah API *android.speech.*

## 2.5 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (*Object-oriented programming* / OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Dimana semua data dan fungsi didalam paradigma ini dikemas kedalam *class-class* atau *object-object.* Setiap *class* atau *object* saling berinteraksi dengan memberikan informasi antara *object* yang satu dengan yang lain. Setiap *object* dapat menerima pesan, memproses data, mengirim, menyimpan dan memanipulasi data. Masing-masing *object* berisikan informasi mengenai dirinya sendiri dan dapat terhubung dengan *object* lain. Konsep dasar dari pemrogramman berorientasi objek antara lain :

1. Kelas (*Class*)

   Merupakan model yang berisi kumpulan *attribute* dan *method* dalam suatu unit untuk tujuan tertentu. Sebagai contoh *class* manusia memiliki *attribute* berat, tinggi, usia kemudian memiliki *method* makan, minum, tidur. *Method* dalam sebuah *class* dapat merubah *attribute* yang dimiliki oleh *class* tersebut. Sebuah *class* merupakan dasar dari modularitas dan struktur dalam pemrograman berorientasi objek.

2. Atribut (*Attribute*)

   *Attribute* dari sebuah *class* adalah variabel global yang dimiliki sebuah *class*. *Attribute* dapat memiliki hak akses *privat, public* maupun *protected*.

3. Metode (*Method*)

   *Method* adalah fungsi atau prosedur yang dibuat oleh seorang programmer di dalam suatu *class*. Dengan kata lain, *method* pada sebuah *class* hampir

sama dengan fungsi atau prosedur pada pemrograman prosedural. Pada sebuah *method* di dalam sebuah *class*, *class* juga memiliki hak akses seperti halnya *attribute.*

4. Objek (*Object*)

   Merupakan perwujudan dari *class*, setiap *object* akan mempunyai *attribute* dan *method* yang dimiliki oleh *class*-nya, contoh : andi, budi, yani merupakan *object* dari *class* manusia. Setiap *object* dapat berinteraksi dengan *object* lainnya meskipun berasal dari *class* yang berbeda.

5. Abstraksi (*Abstraction*)

   Adalah suatu cara untuk melihat suatu *object* dalam bentuk yang lebih sederhana. Sebagai contoh kita tidak melihat sebuah mobil sebagai ibuan kompunen elektronik, sistem mekanik dan empat buah ban, jok, stir, kemudi dan sebagainya. Dengan abstraksi, suatu sistem yang kompleks dapat dipandang sebagai kumpulan subsistem-subsistem yang lebih sederhana, seperti halnya mobil yang merupakan suatu sistem yang terdiri atas berbagai subsitem, seperti subsitem kemudi, subsistem pengereman, dan sebagainya.

6. Enkapsulasi (*Encapsulation*)

   Merupakan suatu mekanisme untuk menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar sistemdan sekaligus menyederhanakan penggunaan sistem tersebut. Memastikan pengguna sebuah *object* tidak dapat mengganti keadaan sebuah *object* dengan cara tidak layak. Hanya metode dalam *object* tersebut yang diberi ijin untuk mengakses keadaannya. Setiap *object* mengakses *interface* yang menyebutkan bagaimana *object* lainnya dapat berinteraksi dengannya. *Object* lainnya tidak akan mengetahui dan tergantung pada representasi dalam *object* tersebut.

7. Polimorfisme (*polymorphism*)

   Merupakan konsep yang memungkinkan digunakannya suatu *interface* yang sama untuk memerintah suatu *object* agar melakukan suatu tindakan yang mungkin secara prinsip sama tetapi secara proses berbeda. Contoh, bila seekor burung menerima pesan “gerak cepat”, dia akan menggerakkan

sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut.

8. Pewarisan (*inheritance*)

   Merupakan konsep mewariskan *attribute* dan *method* yang dimiliki oleh sebuah *class* kepada *class* turunannya. Dengan konsep ini, *class* yang dibuat cukup mendefinisikan *attribute* dan *method* yang spesifik di dalamnya, sedangkan *attribute* dan *method* yang lebih umum akan didapatkan dari *class* yang menjadi induknya. Mengatur polimorfisme dan enkapsulasi dengan mengijinkan *object* didefinisikan dan diciptakan dengan jenis khusus dari *object* yang sudah ada. *Object-object* ini dapat membagi (dan memperluas) perilaku mereka tanpa harus mengimplementasikan ulang perilaku tersebut.

## 2.6 UML

*Unified Modeling Language* (UML) sebuah bahasa yang berdasarkan grafik atau gambar untuk memvisualisasi, menspesifikasikan, membangun, dan pendokumentasian dari sebuah sistem pengembangan software berbasis OO (Object-Oriented). UML merupakan himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML merupakan metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat (*tool*) untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

UML adalah suatu bahasa yang digunakan untuk menentukan memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian

UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi . Penggunaan UML dalam industri terus meningkat, karena UML sudah menjadi standar sebagai bahasa pemodelan yang umum dalam dunia industri perangkat lunak dan pengembangan sistem.

### 2.6.1 Diagram UML

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

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

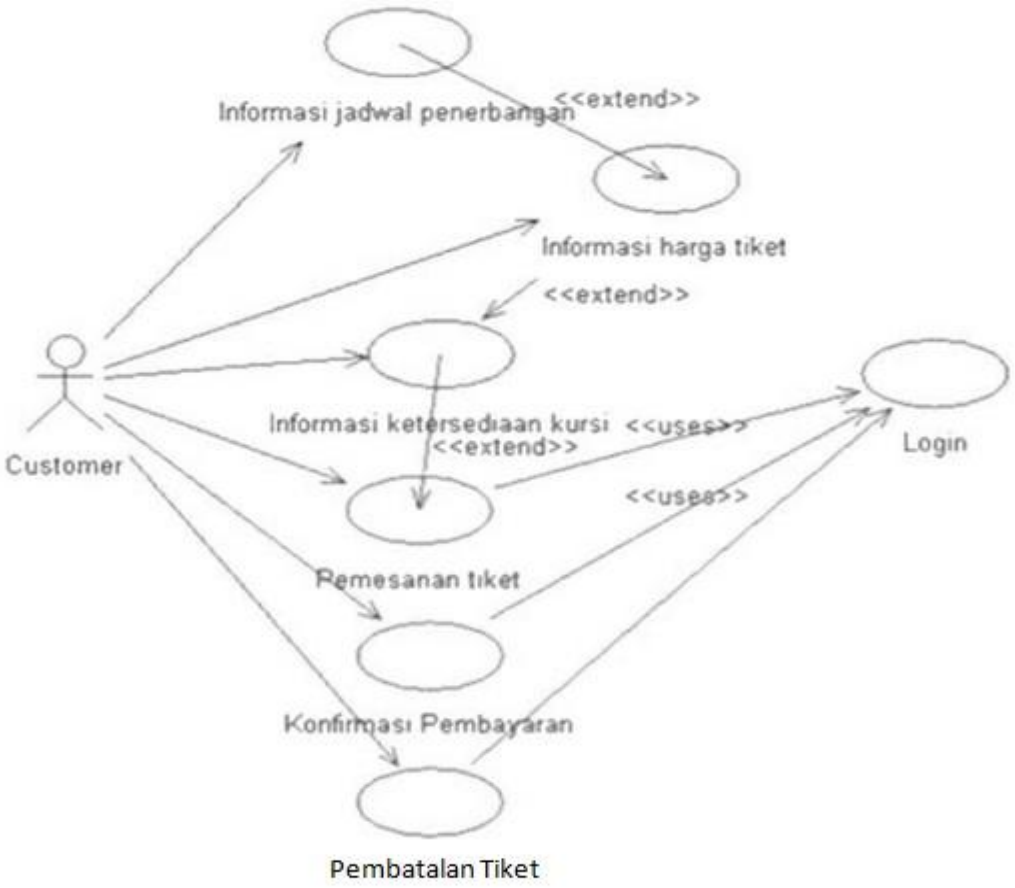
### 2.6.2 *Use Case* Diagram

*Use case diagram* menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja dan sebagainya. Seorang atau sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Sebuah contoh mengenai *use case* pemesanan tiket pesawat. Seorang *customer* ketika ingin membeli tiket pesawat mengharuskan mencari informasi mengenai informasi jadwal penerbangan, informasi harga tiket dan informasi ketersediaan kursi. Informasi – informasi tersebut merupakan bagian dari satu kesatuan informasi secara umum. Artinya *customer* bisa melakukan pemesanan tiket hanya jika informasi mengenai ketersediaan kursi, harga tiket dan jadwal penerbangan telah dilakukan. Pemesanan tiket, konfirmasi pembayaran dan pembatalan tiket mengharuskan *customer* untuk melakukan proses *login*.




Gambar 2.7. *Use Case* Diagram pemesanan tiket pesawat

### 2.6.3 *Class* Diagram

*Class* diagram sangat membantu dalam visualisasi struktur *class* dari suatu sistem. Hal ini disebabkan karena *class* adalah deskripsi kelompok objek-objek dengan properti, perilaku (operasi) dan relasi yang sama. Disamping itu *class* diagram bisa memberikan pandangan global atas sebuah sistem. Hal tersebut tercermin dari *class-class* yang ada dan relasinya satu sama lainnya. Itulah sebabnya *class* diagram menjadi paling popular di UML.

*Class* dalam notasi UML digambarkan dengan kotak. Nama *class* menggunakan huruf besar di awal kalimatnya dan diletakkan di atas kotak. Bila *class* mempunyai nama yang terdiri dari dua suku kata atau lebih, maka semua suku kata digabungkan tanpa spasi dengan huruf awal tiap suku kata

menggunakan huruf besar. Terdapat tiga bagian dalam suatu *class* yaitu nama *class*, atribut dan *operation*