BAB 2
LANDASAN TEORI

2.1 Hadiah

Dituliskan dalam Wikipedia hadiah atau kado adalah pemberian uang, barang, jasa dll yang dilakukan tanpa ada kompensasi balik seperti yang terjadi dalam perdagangan, walaupun dimungkinkan pemberi hadiah mengharapkan adanya imbal balik, ataupun dalam bentuk nama baik (*prestise*) atau kekuasaan. Dalam hubungan manusia, tindakan pertukaran hadiah berperan dalam meningkatkan kedekatan sosial [1], Istilah hadiah dapat juga dikembangkan untuk menjelaskan apa saja yang membuat orang lain merasa lebih bahagia atau berkurang kesedihannya, terutama sebagai kebaikan, termasuk memaafkan, dan sebuah pemberian karena adanya acara atau kegiatan tertentu.

Maksud serta tujuan pemberian hadiah adalah pernyataan cinta atau persahabatan, pernyataan terima kasih untuk hadiah yang diterima sebelumnya, perasaan kasihan, dalam bentuk amal, pernyataan kebersamaan, dalam bentuk bantuan bersama, membagi harta yang dimiliki, menolong yang ditimpa kemalangan, memberikan souvenir perjalanan, kebiasaan pada keadaan (biasanya perayaan).

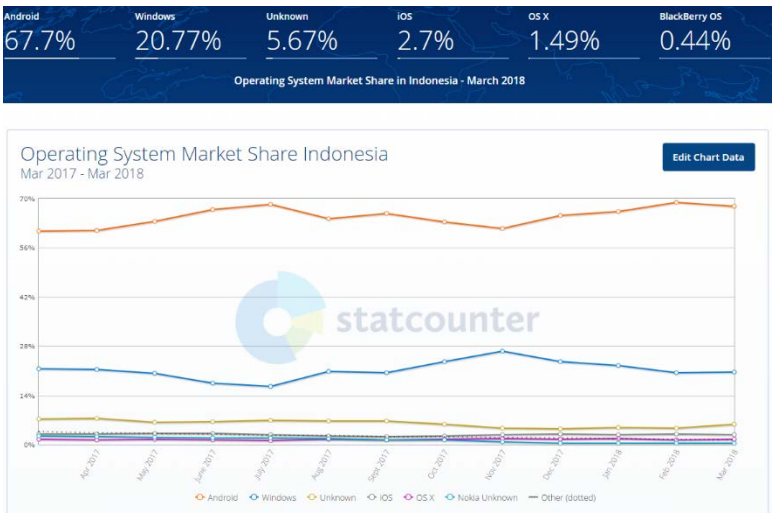
Sedangkan menurut para ahli, dalam buku sherry tahun 1983 oleh Van Baal hadiah merupakan suatu pemberian yang bisa diinterpretasikan sebagai bentuk keterlibatan pemberi dalam peristiwa suka maupun duka yang dialami oleh penerima, sebagai sebuah ajakan pertemanan, membangun relasi, sebgai ucapan permintaan maaf atau terima kasih, sebagai simbol cinta dan berbagai bentuk ekspresi emosional lainnya.

2.2 Android

Android merupakan *platform mobile* yang memberikan pengembang untuk melakukan pengembangan sesuai dengan yang diharapkannya. Sistem operasi yang mendasari Android dilisensikan di bawah GNU, *general Public Lisensi* Versi 2

(GPLv2), yang sering dikenal dengan istilah "*copyleft*" lisensi dimana setiap perbaikan pihak ketiga harus jatuh di bawah terms. Android didistribusikan di bawah Lisensi *Apache Software* (ASL/Apache2), yang memungkinkan untuk distribusi kedua dan seterusnya [9].

Android di Indonesia sendiri memiliki segmentasi pasar yang besar dengan survey salah satu situs online StatCounter GlobalStats menyatakan bahwa pengguna android di Indonesia mencapai 67.7 % selama 1 tahun kebelakang, dapat dilihat pada Gambar 2.1. Dengan begitu bahwa platform yang tepat digunakan di Indonesia menggunakan platform android. Android dipuji sebagai "*platform mobile* pertama yang lengkap, terbuka dan bebas" [2].



Gambar 2.1 Survey *Operation System* di Indonesia

Gambar 2.2 Survey *Operation System* Android di Indonesia

1. Lengkap (*Complete Platform*)

Para desainer dapat melakukan pendekatan yang komprehensif ketika mereka sedang mengembangkan platform Android. Android merupakan sistem operasi yang aman dan banyak menyediakan tools dalam membangun software dan memungkinkan untuk peluang pengembangan aplikasi [9].

2. Terbuka (*Open Source Platform*)

Platform Android disediakan melalui lisensi *open source*. Pengembang dapat dengan bebas untuk mengembangkan aplikasi. Android sendiri menggunakan Linux Kernel 2.6 [9].

3. Free (*Free Platform*)

Android adalah *platform*/aplikasi yang bebas untuk *develop*. Tidak ada lisensi atau biaya royalti untuk dikembangkan pada *platform* Android. Tidak ada biaya keanggotaan diperlukan. Tidak diperlukan biaya pengujian. Tidak ada kontrak yang diperlukan. Aplikasi untuk android dapat didistribusikan dan diperdagangkan dalam bentuk apapun [9].

2.2.1 Sejarah Android

Android adalah sebuah sistem informasi untuk perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware* dan aplikasi. Android menyediakan *platform* 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, dibentuk lah *Open Handset Alliance*, konsorsium dari 34 perusahaan peranti keras, peranti lunak, dan telekomunikasi, termasuk Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, dan Nvidia [10].

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.

Di dunia ini terdapat dua jenis distributor sistem operasi Android. Pertama yang mendapat dukungan penuh dari Google atau *Google Mail Service* (GMS) dan kedua adalah yang benar-benar bebas distribusi nya tanpa dukungan langsung Google atau di kenal sebagai *Open Handset Distribution* (OHD) [10].

Sekitar September 2007 Google mengenalkan Nexus One, salah satu jenis smartphone yang menggunakan Android sebagai sistem operasinya. Telepon seluler ini di produksi oleh HTC *corporation* dan tersedia di pasaran pada 5 Januari 2010. Pada 9 Desember 2008, diumumkan anggota baru yang Communications, di produksi oleh Asustek Computer Inc, Garmin Ltd, Softbank, Sony 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 [10].

Pada masa saat ini sebagian besar vendor-vendor *smartphone* sudah memproduksi *smartphone* berbasis Android, vendor-vendor itu antara lain HTC, Motorola, Samsung, LG, HKC, Huawei, Archos, Webstation Camangi, Dell, Nexus, SciPhone, WayteQ, Sony Ericsson, LG, Acer, Philips, T-Mobile, Nexian, IMO, Asus dan masih banyak lagi vendor smartphone di dunia yang memproduksi android. Hal ini, karena android itu adalah sistem operasi yang open source sehingga bebas didistribusikan dan dipakai oleh vendor manapun [10].

Tidak hanya menjadi sistem operasi di *smartphone*, saat ini Android menjadi pesaing utama dari Apple pada sistem operasi Tablet PC. Pesatnya pertumbuhan Android selain faktor yang disebutkan diatas adalah karena Android itu sendiri adalah platform yang sangat lengkap baik itu sistem operasinya, Aplikasi dan Tool pengembangan, Market aplikasi Android serta dukungan yang sangat tinggi dari komunitas *Open Source* di dunia, sehingga Android terus berkembang pesat baik dari segi teknologi maupun dari segi jumlah *device* yang ada di dunia. [10]

2.2.2 Arsitektur Android

Android adalah tumpukan perangkat lunak berbasis Linux sumber terbuka yang dibuat untuk berbagai perangkat dan faktor bentuk. Diagram berikut menunjukkan komponen besar dari *platform* Android [11].

Gambar 2.3 Arsitektur Android

Google sebagai pencipta Android yang kemudian diasuh oleh Open Handset Alliance mengibaratkan Android sebagai sebuah tumpukan software. Setiap lapisan dari tumpukan ini menghimpun beberapa program yang mendukung fungsi-fungsi spesifik dari sistem operasi. Tumpukan paling bawah adalah kernel. Google menggunakan kernel Linux versi 2.6 untuk membangun Android, yang mencakup memory management, security setting, power management, dan beberapa driver hardware. Bertempat di level yang sama dengan library adalah lapisan runtime yang mencakup serangkaian inti library Java. Dengannya, para programmer dapat mengembangkan aplikasi untuk Android menggunakan bahasa pemrograman Java. Lapisan selanjutnya adalah application framework, yang mencakup program untuk mengatur fungs i- fungsi dasar smartphone [11] .

2.2.2.1 Linux Kernel

Fondasi platform Android adalah kernel Linux. Sebagai contoh, Android Runtime bergantung pada kernel Linux untuk fungsionalitas dasar seperti threading dan manajemen memori tingkat rendah. Menggunakan kernel Linux memungkinkan Android untuk memanfaatkan fitur keamanan inti dan memungkinkan produsen perangkat untuk mengembangkan driver perangkat keras untuk kernel yang cukup dikenal [11].

2.2.2.2 *Hardware Absrtraction Layer*

Hardware Abstraction Layer (HAL) menyediakan antarmuka standar yang mengekspos kemampuan perangkat keras di perangkat ke kerangka kerja Java API yang lebih tinggi. HAL terdiri atas beberapa modul pustaka, masing-masing mengimplementasikan antarmuka untuk komponen perangkat keras tertentu, seperti modul kamera atau bluetooth. Bila API kerangka kerja melakukan panggilan untuk mengakses perangkat keras, sistem Android memuat modul pustaka untuk komponen perangkat keras tersebut [11].

2.2.2.3 Android Runtime

Untuk perangkat yang menjalankan Android versi 5.0 (API level 21) atau yang lebih tinggi, setiap aplikasi menjalankan proses masing-masing dengan tahap *Android Runtime* (ART). ART ditulis guna menjalankan beberapa mesin virtual pada perangkat bermemori rendah dengan mengeksekusi file DEX, format

bytecode yang didesain khusus untuk Android yang dioptimalkan untuk footprint memori minimal. Buat rantai aplikasi, misalnya Jack, mengumpulkan sumber Java ke bytecode DEX, yang dapat berjalan pada platform Android. Beberapa fitur utama ART mencakup:

1. Kompilasi mendahului waktu (AOT) dan tepat waktu (JIT).
2. Pengumpulan sampah (GC) yang dioptimalkan.
3. Dukungan debug yang lebih baik, mencakup profiler sampling terpisah, pengecualian diagnostik mendetail dan laporan kerusakan dan kemampuan untuk mengatur titik pantau guna memantau bidang tertentu.

Sebelum ke Android versi 5.0 (API level 21), Dalvik adalah waktu proses Android. Jika aplikasi Anda berjalan baik pada ART, semestinya berfungsi baik juga pada Dalvik, tetapi mungkin tidak sebaliknya. Android juga menyertakan serangkaian pustaka waktu proses inti yang menyediakan sebagian besar fungsionalitas bahasa pemrograman Java, termasuk beberapa fitur bahasa Java 8, yang digunakan kerangka kerja Java API [11].

2.2.2.4 C/C++

Banyak komponen dan layanan sistem Android inti seperti ART dan HAL dibuat dari kode asli yang memerlukan pustaka asli yang tertulis dalam C dan C++. Platform Android memungkinkan kerangka kerja Java API mengekspos fungsionalitas beberapa pustaka asli pada aplikasi. Misalnya, Anda bisa mengakses OpenGL ES melalui kerangka kerja Java OpenGL API Android guna menambahkan dukungan untuk menggambar dan memanipulasi grafik 2D dan 3D pada aplikasi Anda. Jika Anda mengembangkan aplikasi yang memerlukan kode C atau C++, Anda bisa menggunakan Android NDK untuk mengakses beberapa pustaka platform asli langsung dari kode asli [11].

2.2.2.5 Kerangka Kerja Java API

Keseluruhan rangkaian fitur pada Android OS tersedia untuk Anda melalui API yang ditulis dalam bahasa Java. API ini membentuk elemen dasar yang Anda perlukan untuk membuat aplikasi Android dengan menyederhanakan penggunaan kembali inti, komponen dan layanan sistem modular, yang menyertakan berikut ini:

1. Tampilan Sistem yang kaya dan luas bisa Anda gunakan untuk membuat UI aplikasi, termasuk daftar, kisi, kotak teks, tombol, dan bahkan browser web yang dapat disematkan.
2. Pengelola Sumber Daya, memberikan akses ke sumber daya bukan kode seperti string yang dilokalkan, grafik, dan file layout.
3. Pengelola Notifikasi yang mengaktifkan semua aplikasi guna menampilkan lansiran khusus pada bilah status.
4. Pengelola Aktivitas yang mengelola daur hidup aplikasi dan memberikan back-stack navigasi yang umum.
5. Penyedia Materi yang memungkinkan aplikasi mengakses data dari aplikasi lainnya, seperti aplikasi Kontak, atau untuk berbagi data milik sendiri

Developer memiliki akses penuh ke API kerangka kerja yang sama dengan yang digunakan oleh aplikasi sistem Android [11].

2.2.2.6 Sistem Aplikasi

Android dilengkapi dengan serangkaian aplikasi inti untuk email, perpesanan SMS, kalender, menjelajahi internet, kontak, dll. Aplikasi yang disertakan bersama platform tidak memiliki status khusus pada aplikasi yang ingin dipasang pengguna. Jadi, aplikasi pihak ketiga dapat menjadi browser web utama, pengolah pesan SMS atau bahkan keyboard utama (beberapa pengecualian berlaku, seperti aplikasi Settings sistem).

Aplikasi sistem berfungsi sebagai aplikasi untuk pengguna dan memberikan kemampuan kunci yang dapat diakses oleh developer dari aplikasi mereka sendiri. Misalnya, jika aplikasi Anda ingin mengirimkan pesan SMS, Anda tidak perlu membangun fungsionalitas tersebut sendiri—sebagai gantinya Anda bisa menjalankan aplikasi SMS mana saja yang telah dipasang guna mengirimkan pesan kepada penerima yang Anda tetapkan [11].

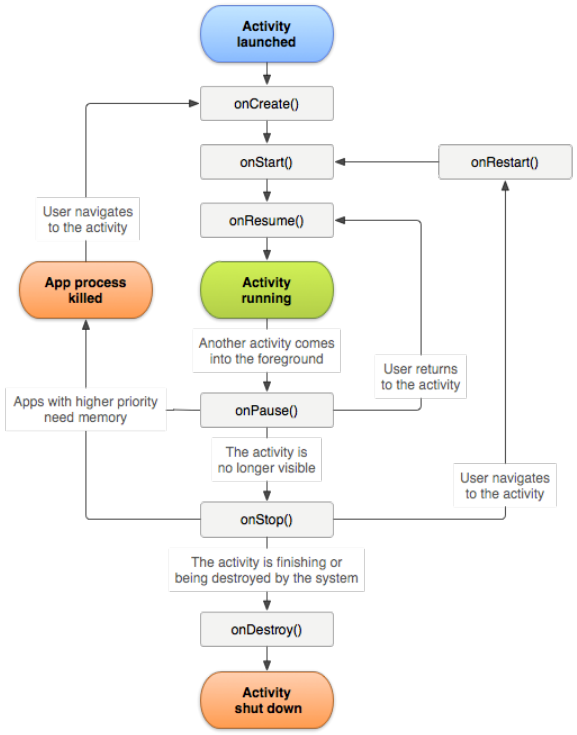
2.2.3 *Android Activity Lifecycle*

Selama aplikasi berjalan, sebuah Activity dalam melalui salah satu dari keempat status berikut:

Table 2.1 Status Activity

Status	**Deskripsi**
Running	Activity terlihat dan dapat berinteraksi dengan pengguna
Paused	Activity masih terlihat, tapi tidak dapat berinteraksi dengan pengguna
Stopped	Activity sudah tidak terlihat tapi masih ada di memory
Killed	Activity sudah tidak terlihat dan dihapus dari sistem karena kebutuhan memory atau method finish() dipanggil.

Diagram berikut ini menampilkan status Activity dengan method yang akan dipanggil sebelum memasuki masing-masing status. Tiap kotak menampilkan method yang dipanggil [12].



Gambar 2.4 Android Lifecycle

Berikut penjelasan fungsi dari tiap method di atas:

1. onCreate()

Di-method ini Activity sudah dimulai tapi belum terlihat oleh pengguna. Inisialisasi sebagian besar dimulai di sini. Misalnya memanggail setContentView () untuk membaca layout, membaca View, dll.

2. onStart()

Activity sudah terlihat tapi belum bisa berinteraksi. Method ini jarang dipakai, tapi bisa sangat berguna untuk mendaftarkan sebuah *BroadcastReceiver* untuk mengamati perubahan yang dapat mempengaruhi UI.

3. onResume()

Activity sudah terlihat dan pengguna sudah dapat berinteraksi. Di sini adalah tempat terbaik untuk menjalankan animasi, membuka akses seperti camera, mengupdate UI, dll.

4. onPause()

Kebalikan dari onResume (). Activity sudah akan bersiap-siap meninggalkan layar (masih terlihat) dan sudah tidak berinteraksi dengan pengguna. Biasanya bila perlu melakukan undo untuk pekerjaan yang dilakukan di onResume () kita lakukan di sini.

5. onStop()

Kebalikan dari onStart () Activity sudah tidak terlihat. Biasanya kita melakukan undo untuk pekerjaan yang dilakukan di dalam onStart ().

6. onDestroy()

Kebalikan dari onCreate (). Method ini dapat terpanggil karena memanggil method finish () atau karena sistem membutuhkan memori lebih. Di dalam onDestroy () kita biasanya membersihkan proses-proses yang ada di belakang layar. Misalnya pengunduhan data dari internet yang mungkin masih berjalan jika tidak dihentikan di onDestroy ().

7. onRestart()

Dipanggil saat activity sudah melalui onStop () tapi akan diaktifkan lagi. Method ini jarang di implementasi.

2.2.4 Fitur Android Berdasarkan Versi

Sejak kemunculannya, sistem operasi Android terus mengalami peningkatan baik dari segi kemampuan maupun performanya. Hal ini diawali dengan versi 1.0 selanjutnya meningkat, hingga versi 8.1 dengan beberapa fitur unggulannya. Berikut ini fitur-fitur dilihat dari perkembangan versi nya:

1. Android 1.6 Donut

Donut membawa informasi dunia dengan kotak telusur kilat. Donut juga menjadi cikal bakal Android yang hadir dalam semua bentuk dan ukuran sementara Android Market kian beranjak dewasa. Kotak telusur kilat Android telah diperkenalkan sejak di sistem Android Donut. Donut dapat menghasilkan penelusuran dari web dan dari konten lokal ponsel dari kotak telusur di layar utama. Android dapat hadir dalam berbagai bentuk dan ukuran berkat kemampuan sistem operasi Android Donut, sehingga dapat berfungsi di berbagai resolusi layar dan rasio aspek. Ponsel juga dapat menggunakan resolusi layar lebih besar dari mode potret berukuran 320 x 480 piksel. Sebelum Google Play muncul, ada Android Market. Diluncurkan pada 2008, Android Market didesain ulang di sistem Android donut untuk menampilkan aplikasi gratis dan aplikasi berbayar teratas ketika katalog aplikasi pihak ketiga Android mulai popular [10].

2. Android 2.1 Eclair

Pada sistem Android Eclair, layar dengan kepadatan piksel tinggi menampilkan wallpaper animasi yang memukau dan merespon sentuhan. Kita juga dapat mengemudi ke mana saja dengan navigasi belokan demi belokan dan informasi lalu lintas *real-time*, semuanya dari ponsel.Google maps navigasi memberi makna baru pada definisi ponsel cerdas. Petunjuk arah belokan demi belokan yang menggunakan data Google maps menyertakan banyak fitur yang biasanya ada di sistem navigasi mobil seperti tampilan 3D hadap depan, panduan suara, dan informasi lalu lintas dan semuanya dapat digunakan secara gratis. Di Android, kita dapat menyesuaikan layar utama sesuai selera. Kita juga dapat menyesuaikan nada dering, wallpaper, serta menyusun aplikasi dan widget di banyak layar atau dalam folder. Eclair memperkenalkan wallpaper animasi yang tampak lebih hidup di layar berukuran 854 x 480 piksel yang luar biasa.

Sebelum perintah suara “Ok Google” populer digunakan, kita dapat menge-tap ikon mikrofon untuk mendiktekan langsung ke ponsel. Sistem Android Eclair mengganti tombol koma di keyboards dengan ikon mikrofon. Cukup sekali tap, kata-kata yang diucapkan muncul langsung di layar [10].

3. Android 2.2 Froyo

Froyo memperkenalkan ponsel secepat kilat yang dapat dikontrol oleh bunyi suara, dan dilengkapi dengan fitur hotspot agar kita selalu terhubung di mana saja. Froyo semakin memper canggih kemampuan suara Android dengan *Voice Action*, sehingga kita dapat melakukan fungsi tombol di ponsel seperti menelusuri, mendapatkan petunjuk arah, membuat catatan, menyetel alarm, dan lain-lain hanya dengan suara. Dengan tethering, Froyo memungkinkan kita mengubah ponsel menjadi hotspot Wi-Fi portable sehingga kita dapat tetap terhubung meski saat dalam perjalanan. Froyo memperkenalkan pengompilasi Dalvik JIT yang memberikan peningkatan performa hingga 5 kali lipat dalam kode yang terikat ke CPU. Froyo juga menghadirkan JavaScript ke browser android, sehingga meningkatkan performa JavaScript sebesar 2-3 kali lipat [10].

4. Android 2.3 Gingerbread

Gingerbread membuat pengalaman menggunakan Android lebih sederhana dan lebih cepat bagi pengguna dan pengembang. Bermain *game* menjadi lebih canggih, baterai lebih tahan lama, dan dukungan NFC memunculkan beragam aplikasi baru. Gingerbread mendorong *game* Android ke jenjang yang lebih tinggi. Pengembang aplikasi dapat membangun *game* 3D yang kaya grafis berkat tingkat akses baru yang lebih rendah untuk kontrol perangkat, grafis, dan penyimpanan. Beberapa tahun sebelum metode pembayaran seluler lazim digunakan di toko, Android meluncurkan dukungan *Near Field Communication* (NFC) agar pengguna dapat mentransmisikan informasi antar perangkat cukup dengan mendekatkannya. Melalui NFC, Gingerbread membuka lebih banyak peluang untuk memberikan layanan dengan sekali menge-tap perangkat. Gingerbread membantu kita mengoptimalkan masa pakai baterai dengan mengetahui cara perangkat menggunakan tenaga baterai, mencari tahu banyaknya baterai yang dikonsumsi oleh tiap aspek perangkat, mulai dari kecerahan layar hingga aplikasi aktif [10].

5. Android 3.0 Honeycomb

Honeycomb diluncurkan di era tablet dengan desain antarmuka fleksibel yang memamerkan gambar besar dan navigasi mulus di layar. Honeycomb mengoptimalkan ruang di layar tablet. Pola tata letaknya yang lebih besar menyempurnakan pengalaman membaca buku, menonton video, menjelajahi peta,

dan lain-lain. Tidak ada lagi tombol fisik utama, kembali, dan menu. Dengan Honeycomb, bilah sistem baru memungkinkan kontrol navigasi dipasang di layar perangkat Android. Setelan cepat baru memudahkan kita mengakses informasi penting seperti melihat waktu, tanggal, masa pakai baterai, dan status sambungan perangkat, semua di satu tempat [10].

6. Android 4.0 Ice Cream Sandwich

Ice Cream Sandwich memudahkan penyesuaian dan kontrol pengguna, sesuaikan layar utama, tentukan banyaknya data yang digunakan, dan langsung bagikan konten kapan pun kita mau. Ice Cream Sandwich memperkenalkan folder aplikasi dan baki favorit. Widget yang menyematkan konten aplikasi aktif di layar utama menjadi lebih fleksibel seperti luaskan widget untuk menampilkan lebih banyak konten atau ciut kan untuk menghemat tempat. Kelola penggunaan data jaringan untuk mengontrol biaya seluler. Lacak jumlah data yang kita gunakan, setel tingkat peringatan dan batas penggunaan, serta non aktifkan layanan data jika sudah mencapai batas. Fitur yang futuristik pada masanya, Android Beam memungkinkan dua ponsel berbagi konten melalui NFC cukup dengan menyentuhkan nya secara bersamaan. Bagikan aplikasi, kontak, musik, dan video kepada orang lain dan semua tanpa perlu membuka menu, aplikasi, atau menyandingkan ponsel [10].

7. Android 4.1 Jelly Bean

Kecerdasan meresap ke semua aspek Jelly Bean yang diluncurkan di masa bantuan personal bersama Google Now. Lebih mudah merespons notifikasi dan kita bisa memakai beberapa akun pengguna di satu perangkat. Dapatkan informasi yang dibutuhkan pada saat yang tepat dengan Google Now. Sebagai pelopor bantuan seluler jenis baru, Google Now memberi tahu cuaca hari ini saat kita sedang bersiap-siap dan waktu perjalanan sebelum kita keluar rumah. Pada sistem Android Jelly Bean, notifikasi menjadi lebih luas agar dapat menampilkan lebih banyak informasi. Kita juga dapat langsung merespons dari notifikasi. Jelly Bean memungkinkan fitur multi pengguna di satu perangkat. Tiap akun memiliki ruang sendiri yang dapat disesuaikan mulai dari layar utama hingga wallpaper, widget, dan aplikasi agar kita dapat menggunakan perangkat bersama-sama tanpa berbagi

info. Fitur multi pengguna diluncurkan dengan tablet dan kemudian diluncurkan ke ponsel dengan Lollipop [10] .

8. Android 4.4 KitKat

Android KitKat membantu kita melakukan banyak hal cukup dengan suara. Ucapkan “Ok Google” untuk meluncurkan penelusuran suara, mengirim SMS, mendapatkan petunjuk arah, atau bahkan memutar lagu. Jika kita mahir menggunakan perangkat, desain baru yang imersif akan membuat kita semakin menikmati konten. Saat membaca buku, bermain *game* atau menonton film, desain imersif KitKat menyembunyikan item yang tidak ingin kita lihat. KitKat memprioritaskan kontak yang paling sering kita telepon dan kita dapat menelusuri tempat terdekat langsung dari ponsel. Saat mendapat panggilan telepon dari nomor tak dikenal, ponsel akan mencari pencocokan dari direktori lokal di Google Maps [10].

9. Android 5.1 Lollipop

Android telah hadir di layar besar dan kecil. Mulai dari ponsel, tablet, jam tangan, TV dan mobil. Lollipop dilengkapi gaya visual berani dan respons yang mulus dari Desain Material. Tampilan dan nuansa gres terbaru dari Android yang memudahkan navigasi perangkat. Didasarkan pada bayangan dan gerakan, Desain Material menggabungkan prinsip dasar desain yang keren dengan segala inovasi teknologi. Pada Android Lollipop, kita dapat berpindah dengan mudah dari ponsel ke tablet, jam tangan Android Wear, atau Android TV. Karena Lollipop bekerja di semua perangkat, kita dapat melanjutkan dari saat terakhir mengakses lagu, aplikasi, foto, dan bahkan penelusuran terbaru. Notifikasi akan berpindah ke layar kunci dan muncul dalam format kartu yang tersusun rapi. Lihat sekilas, atau bahkan lihat dan tanggapi pesan langsung dari layar kunci. Kontrol terperinci memungkinkan kita menyesuaikan konten yang muncul di layar kunci [10].

10. Android 6.0 Marshmallow

Pada Android Marshmallow, kita mendapatkan fitur Now on Tap, masa pakai baterai yang lebih tahan lama, dan izin aplikasi baru untuk kontrol yang lebih leluasa. Dengan Now on Tap, kita mendapatkan bantuan tanpa harus keluar dari aplikasi ataupun website. Cukup tap dan tahan lama tombol utama. Pada

Marshmallow, kita dapat menentukan fitur apa saja yang kita izinkan untuk sebuah aplikasi. Kita juga bisa mengubah perizinan kapan saja. Pada Android Marshmallow, kita juga dapat menikmati teknologi baterai yang lebih cerdas dan efisien. Marshmallow mengoptimalkan baterai untuk hal-hal paling penting dengan fitur istirahatkan dan aplikasi siaga [10].

11. Android 7.0 Nougat

Android 7.0 Nougat memperkenalkan beragam fitur baru dan kemampuan bagi pengguna dan developer, yaitu:

a. Nougat memiliki emoji-emoji baru dan kemampuan menggunakan dua atau lebih bahasa secara bersamaan.
b. Dapat beralih antar aplikasi dengan tap dua kali dan menjalankan dua aplikasi secara berdampingan. Sehingga, kita dapat terus menonton film sambil menulis pesan, atau membaca resep sementara timer terbuka.
c. Teknologi API Vulkan™ yang dapat menampilkan grafis 3D performa tinggi. Pada perangkat yang didukung, aplikasi tampak seolah hidup dengan grafis lebih tajam dan efek menawan.
d. Mode VR (*Virtual Reality*)
e. Fitur Istirahatkan dapat membantu menghemat daya baterai selagi dalam perjalanan. Sehingga, perangkat akan terus berada dalam mode daya rendah selagi dibawa di dalam saku atau dompet.
f. Tata ulang Setelan Cepat Khusus untuk akses lebih cepat ke aplikasi yang diinginkan.
g. Balas Langsung Notifikasi memungkinkan kita untuk membalas secara langsung tanpa membuka aplikasi apapun.
h. Gabungan Notifikasi
i. Penghemat Data dapat membatasi banyaknya data yang digunakan perangkat dengan Penghemat Data. Saat Penghemat Data diaktifkan, aplikasi di latar belakang tidak akan dapat mengakses data seluler.
j. Kontrol Notifikasi saat muncul notifikasi, cukup tekan dan tahan untuk mengubah setelan. Sehingga kita dapat menonaktifkan notifikasi mendatang langsung dari setelan ini.

k. Mengubah ukuran teks pada perangkat dan mengubah ukuran ikon [10].

12. Android 8.0 Oreo

Beragam pembaharuan fitur-fitur Android diberikan pada Android Oreo, yaitu:

a. Swift bergerak, di belakang layar lebih cepat. Memulai tugas favorit lebih cepat dengan kecepatan booting 2x saat menyalakannya (waktu booting yang diukur di Google Pixel). Android Oreo membantu meminimalkan aktivitas latar belakang pada aplikasi yang paling digunakan.
b. Dengan seizin pengguna, Autofill mengingat login untuk membawa kita ke aplikasi favorit dengan kecepatan supersonik.
c. *Picture in Picture* memungkinkan kita melihat dua aplikasi sekaligus.
d. Tekan titik pemberitahuan untuk segera melihat apa yang baru, dan mudah menghapusnya dengan menggesek pergi.
e. Teleport langsung ke aplikasi baru langsung dari browser Anda, tidak perlu instalasi.
f. Perlindungan Google Play bekerja agar perangkat dan data tetap aman dari aplikasi yang tidak sesuai dengan memindai lebih dari 50 miliar aplikasi per hari, bahkan yang belum terpasang.
g. Baterai lebih hemat.
h. Tambahan emoji baru [10].

Smartphone Android dirasakan sangat cocok untuk membanguna aplikasi karena memiliki fitur dan fasilitas yang banyak. Selain itu juga, pengguna android di Indonesia cakupannya sangat besar yaitu 67% [2].

2.3 JSON

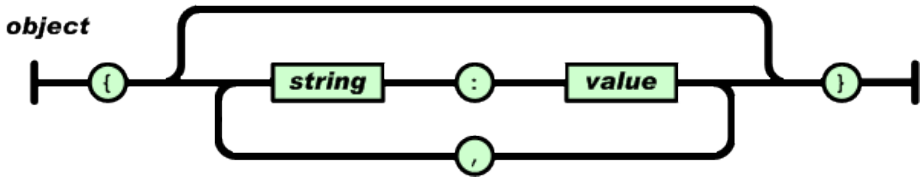
JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (generate) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemprograman JavaScript, Standar ECMA-262 Edisi ke-3 - Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh programmer keluarga C termasuk C, C++, C#, Java, JavaScript, Perl, Python dll [13] .

Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data. JSON terbuat dari dua struktur:

1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau *associative array*.
2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).

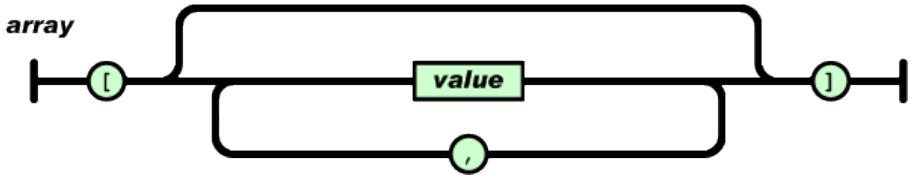
Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemprograman moderen mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemprograman yang juga berdasarkan pada struktur data ini. JSON menggunakan bentuk sebagai berikut:

1. Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan: (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh, (koma).



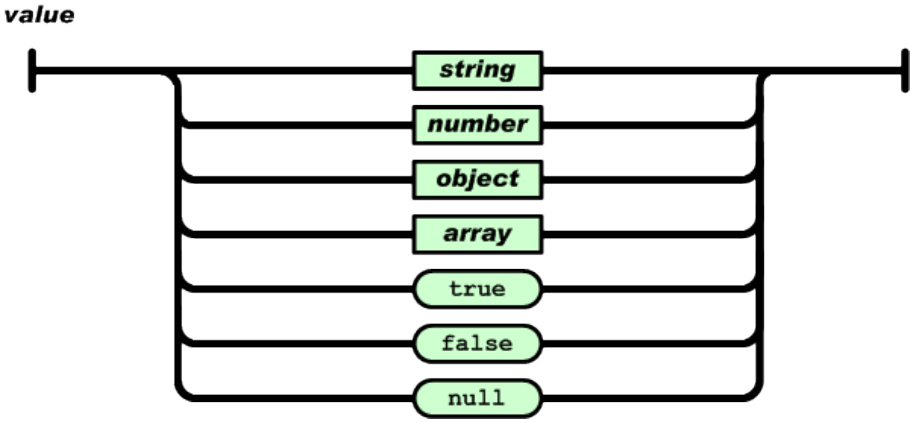
Gambar 2.5 Object JSON

2. Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh, (koma).



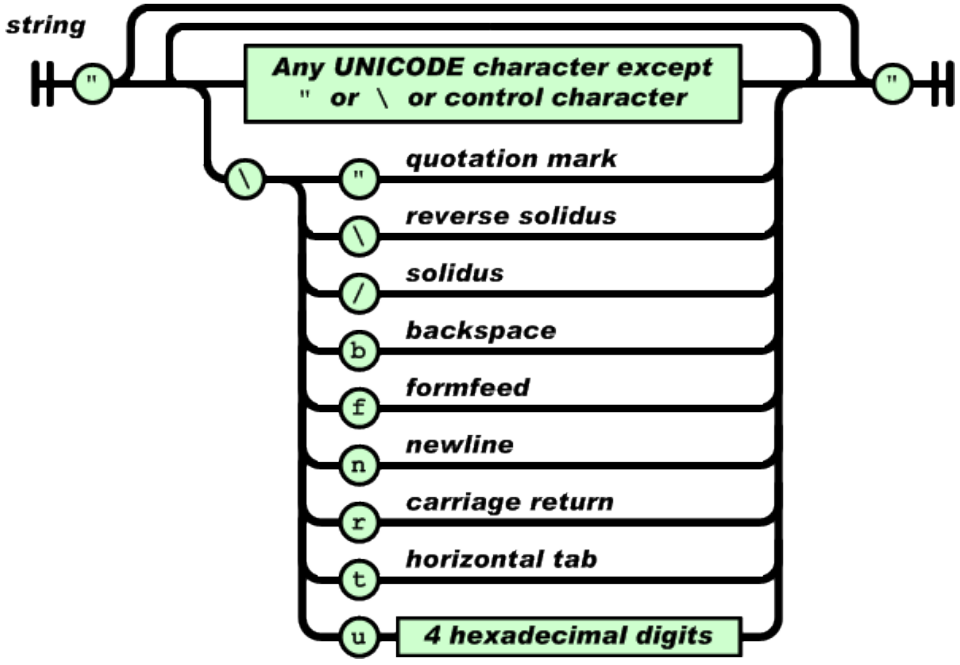
Gambar 2.6 Array JSON

3. Nilai (value) dapat berupa sebuah string dalam tanda kutip ganda, atau angka, atau true atau false atau null, atau sebuah objek atau sebuah larik. Struktur-struktur tersebut dapat disusun bertingkat.



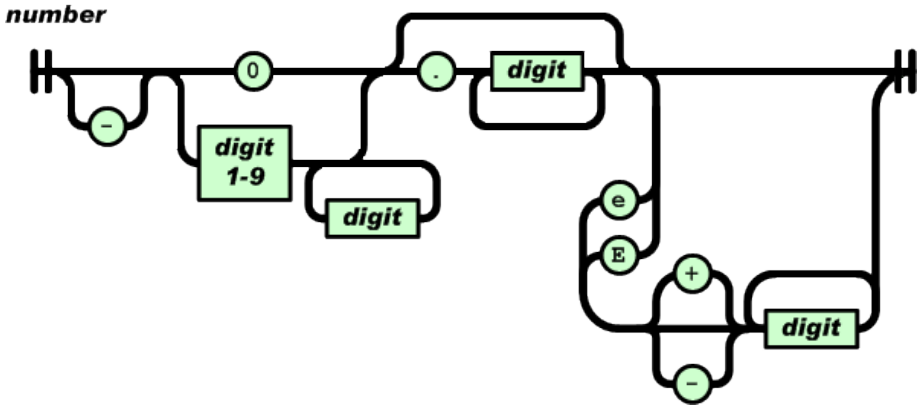
Gambar 2.7 Value JSON

4. String adalah kumpulan dari nol atau lebih karakter Unicode, yang dibungkus dengan tanda kutip ganda. Di dalam string dapat digunakan backslash escapes "\" untuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada string. String sangat mirip dengan string C atau Java.



Gambar 2.8 String JSON

5. Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.

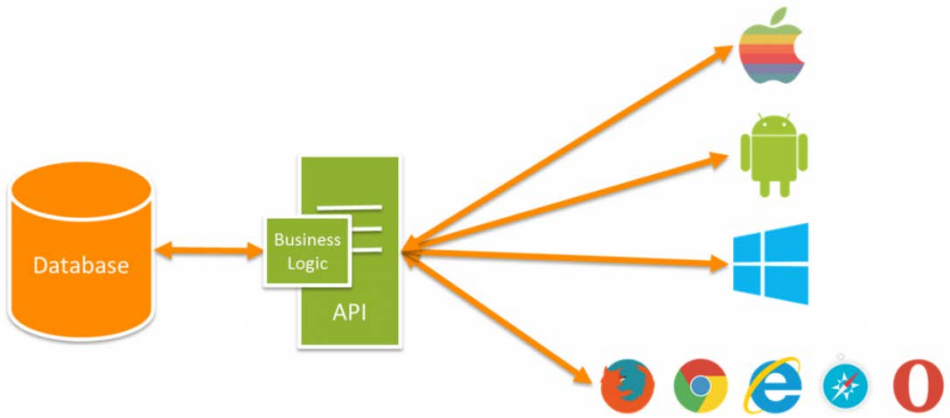


Gambar 2.9 Number JSON

Spasi kosong (*whitespace*) dapat disisipkan di antara pasangan tanda-tanda tersebut, kecuali beberapa detil encoding yang secara lengkap dipaparkan oleh bahasa pemprograman yang bersangkutan [13].

2.4 *Application Programming Interface*

API adalah singkatan dari Application Programming Interface, dan memungkinkan developer untuk mengintegrasikan dua bagian dari aplikasi atau dengan aplikasi yang berbeda secara bersamaan. API terdiri dari berbagai elemen seperti function, protocols, dan tools lainnya yang memungkinkan developer untuk membuat aplikasi. Tujuan penggunaan API adalah untuk mempercepat proses development dengan menyediakan function secara terpisah sehingga developer tidak perlu membuat fitur yang serupa. Penerapan API akan sangaat terasa jika fitur yang diinginkan sudah sangat kompleks, tentu membutuhkan waktu untuk membuat yang serupa dengannya. Misalnya: integrasi dengan *payment gateway*. Terdapat berbagai jenis sistem API yang dapat digunakan, termasuk sistem operasi, library, dan web [6].



Gambar 2.10 Alur API

API yang bekerja pada tingkat sistem operasi membantu aplikasi berkomunikasi dengan layer dasar dan satu sama lain mengikuti serangkaian protokol dan spesifikasi. Contoh yang dapat menggambarkan spesifikasi tersebut adalah POSIX (*Portable Operating System Interface*). Dengan menggunakan standar POSIX, aplikasi yang di-compile untuk bekerja pada sistem operasi tertentu juga dapat bekerja pada sistem lain yang memiliki kriteria yang sama. Software library juga memiliki peran penting dalam menciptakan compatibility antar sistem yang berbeda [6].

Aplikasi yang berinteraksi dengan library harus mengikuti serangkaian aturan yang ditentukan oleh API. Pendekatan ini memudahkan software developer untuk membuat aplikasi yang berkomunikasi dengan berbagai library tanpa harus memikirkan kembali strategi yang digunakan selama semua library mengikut API yang sama. Kelebihan lain dari metode ini menunjukkan betapa mudahnya menggunakan library yang sama dengan bahasa pemrograman yang berbeda [6].

Seperti namanya, Web API dalam diakses melalui protokol HTTP, ini adalah konsep bukan teknologi. Kita bisa membuat Web API dengan menggunakan teknologi yang berbeda seperti PHP, Java, .NET, dll. Misalnya Rest API dari Twitter menyediakan akses read dan write data dengan mengintegrasikan twitter kedalam aplikasi kita sendiri [6].

2.5 OOAD (*Object Oriented Analysis and Design*)

Di dalam membangun sebuah sistem berorientasi objek akan menjadi lebih baik apabila langkah awalnya didahului dengan proses analisis dan perancangan

yang berorientasi objek. Tujuannya adalah untuk mempermudah programmer di dalam mendesain program dalam bentuk objek-objek dan hubungan antara objek tersebut untuk kemudian dimodelkan dalam sistem nyata [14]

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Metodologi berorientasi objek merupakan suatu cara bagaimana system perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek, perancangan objek, pemrograman berorientasi objek, dan pengujian berorientasi objek. Keuntungan menggunakan metodologi berorientasi objek adalah sebagai berikut:

1. Meningkatkan produktivitas

   Karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*reusable*).
2. Kecepatan pengembangan

   Karena system yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean
3. Kemudahan pemeliharaan.

   Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dan pola-pola yang mungkin sering berubah-ubah.
4. Adanya konsistensi

   Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.
5. Meningkatkan kualitas perangkat lunak

   Karena pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

Pendekatan berorientasi objek merupakan suatu teknik atau cara pendekatan dalam melihat permasalahan dan sistem (sistem perangkat lunak, sistem informasi,

atau sistem lainnya). Pendekatan berorientasi objek akan memandang system yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.

Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut, mulai dan abstraksi objek, kelas, hubungan antarkelas sampai abstraksi system. Saat mengabstraksikan dan memodelkan objek, data dan proses-proses yang dipunyai oleh objek akan dienkapsulasi menjadi satu kesatuan.

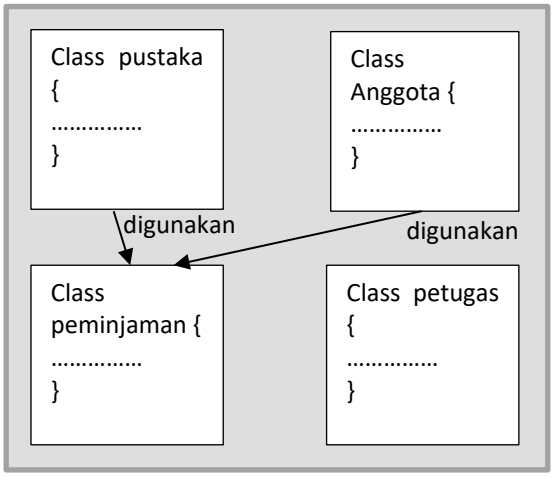
Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemrograman, dan pengujian perangkat lunak. Ada berbagai teknik yang dapat digunakan pada masing-masing tahap tersebut, dengan aturan dan alat bantu pemodelan tertentu.

Sistem berorientasi objek merupakan sebuah system yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah system yang komponennya dibungkus menjadi kelompok data dan fungsi. Setiap komponen dalam system tersebut dapat mewarisi atribut dan sifat dan komponen lainnya. Dan dapat berinteraksi satu sama lain. Berikut ini adalah beberapa konsep dasar yang harus dipahami tentang metodologi berorientasi objek:

1. Kelas (*class*)

Kelas adalah kumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi static dan himpunan objek yang sama yang mungkin lahir atau diciptakan dan kelas tersebut. Sebuah kelas akan mempunyai sifat (*atribut*). Kelakuan (operasi/metode), hubungan (*relationship*) dan arti. Suatu kelas dapat diturunkan dan kelas yang lain, dimana atribut dan kelas semula dapat diwariskan ke kelas yang baru.

Secara teknis, kelas adalah sebuah struktur tertentu dalam pembuatan perangkat lunak. Kelas merupakan bentuk struktur pada kode program yang menggunakan metodologi berorientasi objek. Ilustrasi dari sebuah kelas dapat dilihat pada gambar berikut.



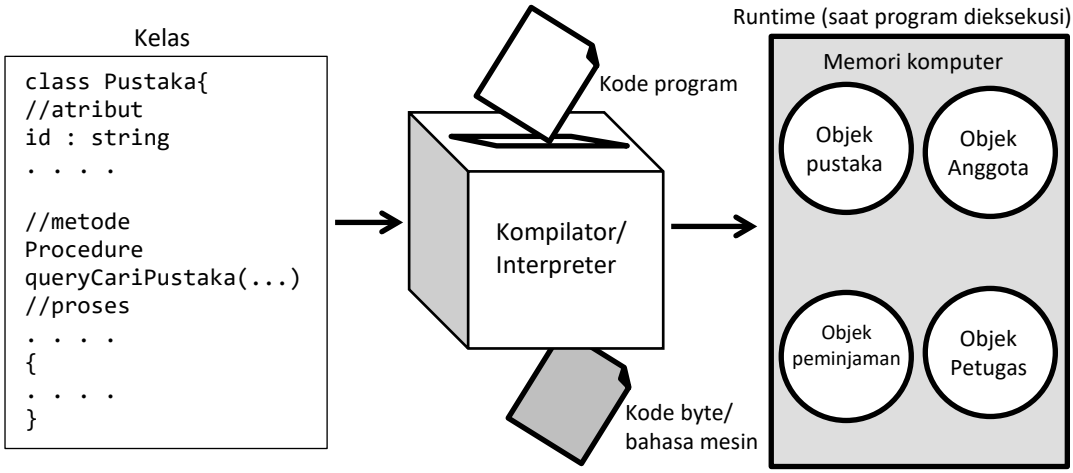
Gambar 2.11 Ilustrasi Kelas

Kelas secara fisik adalah berkas atau file yang berisi kode program, dimana kode program merupakan semua hal yang terkait dengan nama kelas.

2. Objek (*object*)

Objek adalah abstraksi dan sesuatu yang mewakili dunia nyata seperti benda, manusia, satuan organisasi, tempat, kejadian, struktur, status, atau hal - hal lain yang bersifat abstrak. Objek merupakan suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi, dan dihancurkan.

Secara teknis, sebuah kelas saat program dieksekusi maka akan di buat sebuah objek. Objek dilihat dari segi teknis adalah elemen pada saat runtime yang akan diciptakan, dimanipulasi, dan dihancurkan saat eksekusi sehingga sebuah objek hanya ada saat sebuah objek hanya ada saat sebuah program dieksekusi, jika masih dalam bentuk kode, disebut sebagai kelas jadi pada saat runtime (saat sebuah program dieksekusi), yang kita punya adalah objek, di dalam teks program yang kita lihat hanyalah kelas.



Gambar 2.12 Ilustrasi Kelas dan Objek

3. Metode (*method*)

Operasi atau metode atau method pada sebuah kelas hamper sama dengan fungsi atau prosedur pada metodologi struktural. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri. Operasi atau metode merupakan fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek.

Metode atau operasi dapat berasal dari event, aktivitas atau aksi keadaan, fungsi, atau kelakuan dunia nyata. Atau kelakuan dunia nyata. Contoh metode atau operasi misalnya *read, write, move copy*, dan sebagainya. Kelas sebaiknya memiliki metode get dan set untuk setiap atribut agar konsep enkapsulasi tetap terjaga. Metode get digunakan untuk memberikan akses kelas ini dalam mengakses atribut, dan set adalah metode yang digunakan untuk mengisi atribut, agar kelas ini tidak mengakses atribut secara langsung.

4. Atribut (*attribute*)

Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut dipunyai secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya. Atribut sebaiknya bersifat privet untuk menjaga konsep enkapsulasi.

5. Abstraksi (*abstraction*)

Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.

6. Enkapsulasi (*encapsulation*)

Pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerjanya.

7. Pewarisan (*inheritance*)

Mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dan dirinya.

8. Antarmuka (*interface*)

Antarmuka atau *interface* sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah *interface* dapat diimplementasikan oleh kelas lain. Sebuah kelas dapat mengimplementasikan lebih dari satu antarmuka dimana kelas ini akan mendeklarasikan metode pada antarmuka yang dibutuhkan oleh kelas itu sekaligus mendefinisikan isinya pada kode program kelas itu. Metode pada antarmuka yang diimplementasikan pada suatu kelas harus sama persis dengan yang ada pada antarmuka. Antarmuka atau *interface* biasanya digunakan agar kelas yang lain tidak mengakses langsung ke suatu kelas, mengakses antarmukanya.

9. Reusability

Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permisalan pada permasalahan lainnya yang melibatkan objek tersebut. Misalkan dalam sebuah aplikasi peminjaman buku diperlukan kelas anggota, maka ketika membuat aplikasi penyewaan VCD, kelas anggota ini bisa digunakan kembali dengan sedikit perubahan untuk aplikasi penyewaan VCD tanpa harus membuat dari awal kembali.

10. Generalisasi dan Spesialisasi

Menunjukkan hubungan antara kelas dan objek uang umum dengan kelas dan objek yang khusus. Misalnya kelas yang lebih umum (*generalisasi*) adalah kendaraan darat dan kelas khususnya (spesialisasi) adalah mobil, motor, dan kereta.

11. Komunikasi Antar Objek

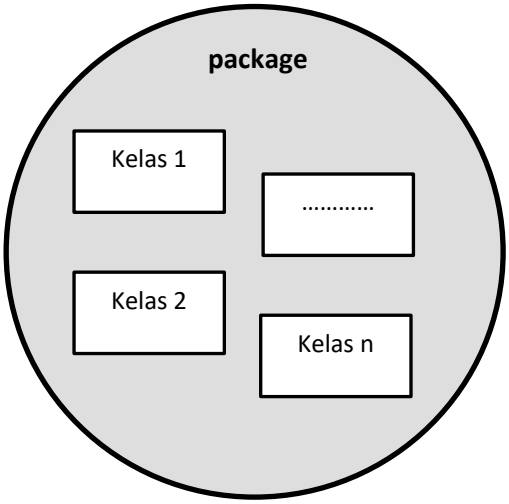
Komunikasi antar-objek dilakukan lewat pesan (*message*) yang dikirim dan satu objek ke objek lainnya.

12. Polimorfisme (*polymorphism*)

Kemauan suatu objek untuk digunakan di banyak tujuan yang berbeda dengan nama yang sama sehingga menghemat baris program.

13. Package

Package adalah sebuah container atau kemasan yang dapat di gunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam package yang berbeda. Ilustrasi dari sebuah package dapat dilihat pada gambar berikut [15].



Gambar 2.13 *Package*

Pemrograman berorientasi objek digunakan agar lebih mudah dalam perawatan perangkat lunak, pemrograman berorientasi objek akan mudah dalam merawat dan memodifikasi kode yang sudah ada. Objek yang baru dapat dibuat tanpa mengubah kode yang sudah ada. Dalam membangun aplikasi *frontend* sistem alat bantu ubinan menggunakan pemrograman berorientasi objek.

2.6 UML (*Unified Modelling Language*)

Unified Modelling Language (UML) adalah sebuah bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang dan

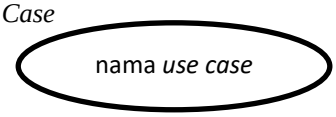
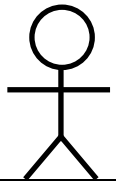
mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem [3].Untuk membuat sebuah program berorientasi objek, UML cocok digunakan untuk merancang model-model yang akan dibangun nantinya.

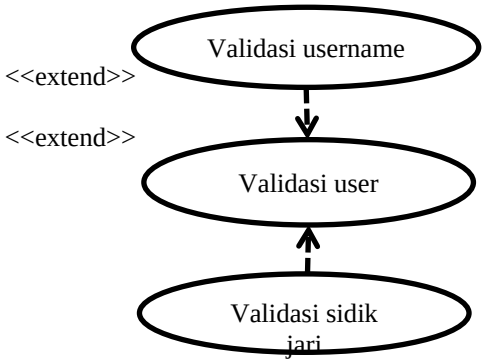
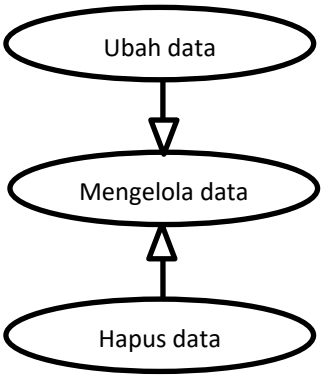
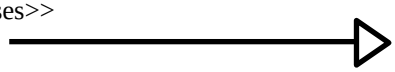
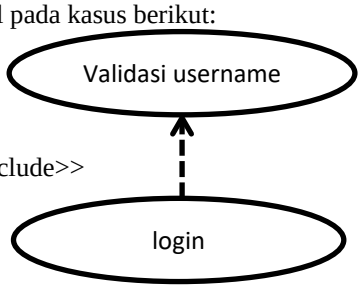
UML memiliki beberapa diagram antara lain: *use case diagram, class diagram, statechart diagram, activity diagram, sequence diagram, collaboration diagram, component diagram, deployment diagram* [3]. Berikut ini penjelasan untuk beberapa diagram yang akan digunakan dalam penelitian ini:

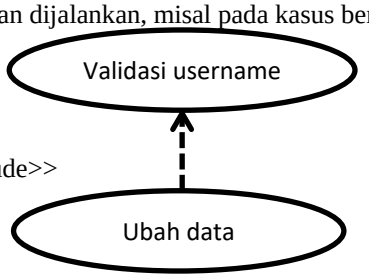
2.6.1 Use Case Diagram

Use case diagram merupakan sebuah gambaran fungsionalitas sebuah sistem. Sebuah *use case* merepresentasikan interaksi antara aktor dengan sistem. *Use case* sangat menentukan karakteristik sistem yang sedang dibuat. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu [3].Berikut adalah simbol-simbol yang ada pada *use case diagram:*

Tabel 2.1 Simbol-simbol diagram use case

Simbol	Deskripsi
Use Case 	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor; biasanya dinyatakan dengan menggunakan kata kerja di awal di awal frase nama use case
Aktor/*actor*  Nama aktor	Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, sehingga walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor
Asosiasi / *association* 	Komunikasi antara aktor dan use case yang berpartisipasi pada use case atau use case memiliki interaksi dengan aktor.
Ekstensi / *extend* <<extend>> 	Relasi *use case* tambahan ke sebuah use case dimana use case yang ditambahkan dapat berdiri sendiri walaupun tanpa *use case* tambahan itu; mirip dengan prinsip *inheritance* pada pemrograman berorientasi obyek; biasanya *use case* tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan. Misalnya arah panah mengarah pada *use case* yang ditambahkan, biasanya *use case* yang menjadi *extend*-nya merupakan jenis yang sama dengan *use case* yang menjadi induknya

Simbol	Deskripsi
	
Generalisasi / *generalization* 	Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua buah use case dimana fungsi yang satu adalah fungsi yang lebih umum dari yang lainnya, misalnya:
	 arah panah mengarah pada use case yang menjadi generalisasi nya (umum).
Menggunakan / *include* / *uses* <<include>>  <<uses>> 	Relasi use case tambahan ke sebuah use case yang ditambahkan memerlukan use case ini untuk menjalankan fungsinya atau sebagai syarat dijalankan *use case* ini Ada dua sudut pandang yang cukup besar mengenai *include* di *use case*: *Include* berarti *use case* yang ditambahkan akan selalu dipanggil saat *use case* tambahan dijalankan, misal pada kasus berikut: 

Simbol	Deskripsi
	Include berarti *use case* yang tambahan akan selalu melakukan pengecekan apakah *use case* yang ditambahkan telah dijalankan sebelum *use case* tambahan dijalankan, misal pada kasus berikut:  Kedua interpretasi di atas dapat dianut salah satu atau keduanya tergantung pada pertimbangan dan interpretasi yang dibutuhkan.

2.6.2 Class Diagram

Class merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class, package* dan objek yang berhubungan satu sama lain seperti *containment*, asosiasi, dan lain-lain [14]. Berikut adalah simbol-simbol yang ada pada *class diagram.*

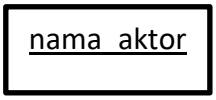
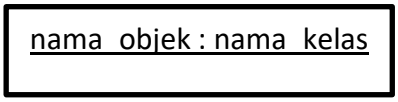
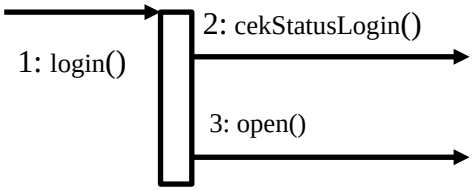
Tabel 2.2 Simbol-simbol class diagram

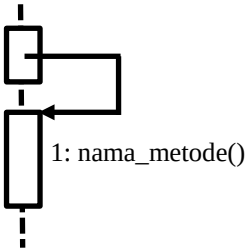
Simbol	Deskripsi
Kelas **nama_kelas** +atribut +operasi()	Kelas pada struktur sistem
Antarmuka / *interface* **nama_interface**	Sama dengan konsep *interface* dalam pemrograman berorientasi objek
Asosiasi / *association*	Relasi antarkelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity*
Asosiasi berarah / *directed association*	Relasi antarkelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity*.
Generalisasi	Relasi antarkelas dengan makna generalisasi-spesialisasi (umum khusus).
Kebergantungan / *dependency*	Relasi antarkelas dengan makna kebergantungan antarkelas
Agregasi / *aggregation*	Relasi antarkelas dengan maksa semua bagian (*whole part*)

2.6.3 Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri antar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan [14]. Berikut adalah simbol-simbol yang ada pada *sequence diagram*:

Tabel 2.3 Simbol-simbol yang ada pada sequence diagram

Simbol	Deskripsi
Aktor  Nama aktor Atau  Tanpa waktu aktif	Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, sehingga walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama actor
Garis hidup / *lifeline* 	Menyatakan kehidupan suatu objek, untuk menggambarkan kelas dan objek.
Objek 	Menyatakan objek dalam keadaan aktif dan berinteraksi, semua yang terhubung dengan waktu aktif ini adalah sebuah tahapan yang dilakukan di dalamnya, misalnya
Waktu aktif 	 Maka cekStatusLogin() dan open() dilakukan di dalam metode login() Aktor tidak memiliki waktu aktif

Simbol	Deskripsi
Pesan tipe create <<create>>	Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat.
Pesan tipe call 1: nama_metode()	Menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri,  Arah panah mengarah pada objek yang memiliki operasi/metode, karena ini memanggil operasi/metode maka operasi/metode yang dipanggil harus ada pada diagram kelas sesuai dengan kelas objek yang berinteraksi
Pesan tipe send 1: masukan()	Menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri, sesuai dengan kelas objek yang berinteraksi.
Pesan tipe return 1: keluaran()	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian.
Pesan tipe destroy <<destroy>>	Menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhiri, sebaiknya jika ada create maka ada destroy.

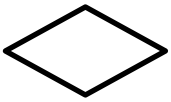
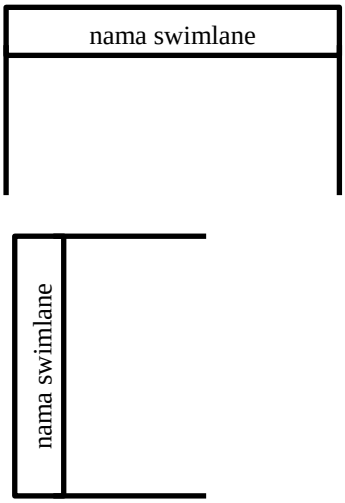
2.6.4 Activity Diagram

Activity Diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi [14].

Activity Diagram merupakan state diagram khusus, dimana sebagian besar state adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya state

sebelum (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behavior internal* sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum. Sebuah aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas [14]. Berikut adalah simbol-simbol yang ada pada *activity diagram*:

Tabel 2.4 Simbol-simbol *activity diagram*

Simbol	Deskripsi
Status awal 	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal
Aktivitas 	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja
Percabangan / *decision* 	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu.
Penggabungan / *join*	Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu
Status akhir 	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir.
Swimlane 	Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi

2.7 Metode *K-Nearest Neighbor*

Sistem rekomendasi adalah suatu sistem yang dirancang untuk memprediksi satu item yang sesuai dengan minat pengguna, yang mana item tersebut akan direkomendasikan pada pengguna. Prediksi informasi minat pengguna dapat diperoleh berdasarkan pola aksi prilaku pengguna atau sering dikatakan sebagai profil pengguna. Pada sistem rekomendasi, terdapat beberapa item yang akan disaring untuk direkomendasikan kepada pengguna berdasarkan profil pengguna, skala rating, dan lain-lain sehingga menghasilkan beberapa item yang direkomendasikan kepada pengguna. Saat ini banyak sistem yang telah mengadopsi sistem rekomendasi pada penerapannya. Seperti halnya pada sosial media twitter, facebook, youtube.com, amazon.com dan lain-lain [5].

Algoritma *K-Nearest Neighbor* (K-NN) adalah suatu metode yang menggunakan algoritma *supervised*. K-NN termasuk kelompok *instance-based learning*. Algoritma ini juga merupakan salah satu teknik *lazy learning*. *K-Nearest Neighbor* dilakukan dengan mencari kelompok k-objek dalam data training yang paling dekat (mirip) dengan objek pada data baru atau data testing.

Untuk menghitungnya digunakan rumus jarak *Euclidean*. Jarak *Euclidean* adalah jarak yang paling umum digunakan pada data numerik

$$dxy = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Gambar 2.14 Rumus jarak *Euclidean*

Keterangan:

x_1 = Sampel Data

y_1 = Data Uji / Testing

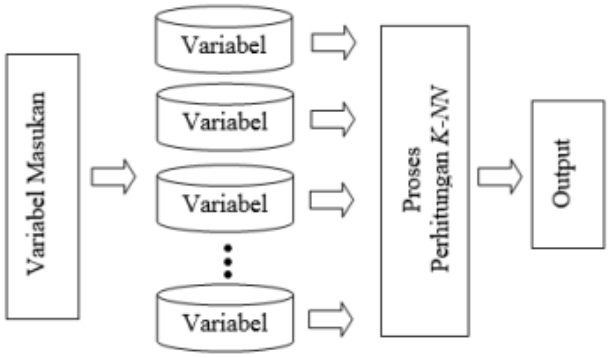
i = Variabel Data

dxy = Jarak *Euclidean*

n = Dimensi Data

Penerapan Metode *K-Nearest Neighbor*. Adapun penerapan metode K-NN melalui beberapa langkah:

1. Tentukan parameter k
2. Hitung jarak antara data yang akan dievaluasi dengan semua pelatihan
3. Urutkan jarak yang terbentuk (urut naik)
4. Tentukan jarak terdekat sampai urutan k
5. Pasangkan kelas yang bersesuaian
6. Cari jumlah kelas dari tetangga yang terdekat dan tetapkan kelas tersebut sebagai kelas data yang akan dievaluasi.



Gambar 2.15 Penerapan KNN

Tahap pertama dimulai dari inputan user yang terdiri dari jawaban dan variabel-variabel yang diberikan oleh sistem. Setelah jawaban diinputkan, proses selanjutnya adalah evaluasi data ujidengan data sample menggunakan perhitungan K-NN. Dalam hal ini yang dijadikan evaluasi data uji adalah kebutuhan calon pembeli hadiah yang kemudian dibandingkan dengan data sample yang sudah ada. Tahapan dari metode *K-Nearest Neighbor*

1. Tahap pertama adalah penentuan parameter K, misalnya K=6. Ini berarti 6 sample data yang terdekat akan dijadikan rekomendasi.
2. Setelah semua kriteria tersebut lengkap, barulah dilakukan perhitungan jarak antara data training dan data sample yang sudah ada. Dalam perhitungan jarak nanti akan digunakan rumus *Euclidean Distance*.