BAB II

TINJAUAN PUSTAKA

II.1. Aplikasi

Aplikasi adalah suatu *sub* kelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untuk melakukan suatu tugas yang diinginkan pengguna. Contoh utama aplikasi adalah pengolah kata, lembar kerja, memanipulasi foto, merancang rumah dan pemutar media. Beberapa aplikasi yang digabung bersama menjadi sutau pake disebut sebagai suatu paket atau *suite* aplikasi (*application suite*). Contohnya adalah *Microsoft Office* dan *OpenOffice.org*, yang menggabungkan suatu aplikasi pengolah kata, lembar kerja dan beberapa aplikasi lainnya. Aplikasi-aplikasi dalam suatu paket biasanya memiliki atarmuka pengguna yang memiliki kesamaan sehingga memudahkan pengguna untuk mempelajari dan menggunakan tiap aplikasi. Sering kali, mereka memiliki kemampuan untuk saling berinteraksi satu sama lain sehingga menguntungkan pengguna. Contohnya, suatu lembar kerja dapat dibenamkan dalam suatu dokumen pengolah kata walaupun dibuat pada aplikasi lembar kerja yang terpisah. (*Dahlan Abdullah ; 2013 : 152*)

II.2. Kriptografi

II.2.1. Sejarah Kriptografi

Pada zaman Romawi kuno dikisahkan pada suatu saat, ketika Julius Caesar ingin mengirimkan suatu pesan rahasia kepada seorang Jenderal di medan perang. Pesan tersebut mengandung rahasia, Julius Cesar tidak ingin pesan

tersebut terbuka ditengah jalan. Disini Julius Caesar menemukan suatu cara agar pesan tidak dapat dipahami oleh siapapun kecuali Jenderalnya saja, yaitu dengan mengacak pesan yang akan dikirim.

Kriptografi berasal dari bahasa Yunani, menurut bahasa dibagi menjadi dua, yaitu *kripto* dan *graphia*, *kripto* berarti *secret* (rahasia) dan *graphia* berarti *writing* (tulisan). Kriptografi adalah ilmu dan seni untuk menjaga keamanan pesan ketika pesan dikirim dari suatu tempat ketempat lain. Orang yang melakukan ini disebut "*Cryptographer*". Sebuah pesan atau data yang masih asli disebut *plaintext* atau *cleartext*, maka pesan atau data yang telah diubah atau dienkripsi disebut *chipertext*. Kriptografi ini bertujuan untuk menjaga kerahasiaan informasi yang terkandung dalam data sehingga informasi tersebut tidak dapat diketahui oleh orang yang tidak berhak. (*Tri Puji Rahayu ; 2012 : 144*)

II.2.2. Komponen Kriptografi

Pada dasarnya, *cryptography* terdiri dari beberapa komponen (*Tri Puji Rahayu ; 2012 : 144*) seperti :

1. Enkripsi; enkripsi merupakan hal yang sangat penting dalam *cryptography* sebagai pengamanan atas data yang dikirimkan agar rahasia terjaga. Pesan aslinya disebut *plaintext* yang diubah menjadi kode-kode yang tidak dimengerti.
2. Dekripsi; dekripsi merupakan kebalikan dari enkripsi, pesan yang telah dienkripsi dikembalikan kebentuk asalnya (*plaintext*), yang disebut dekripsi pesan. Algoritma yang digunakan untuk dekripsi tentu berbeda dengan yang digunakan untuk enkripsi.

3. Kunci; kunci yang dimaksud disini adalah kunci yang dipakai untuk melakukan enkripsi dan dekripsi. Kunci terbagi dalam dua bagian yakni kunci pribadi (*private key*) dan kunci umum (*publik key*).
4. *Chipertext*; merupakan suatu pesan yang sudah melalui pesan proses enkripsi. Pesan yang ada pada *chipertext* tidak bisa dibaca karena berisi karakter-karakter yang tidak memiliki makna (arti).
5. *Plaintext*; sering juga disebut *cleartext*; merupakan suatu pesan bernakna yang ditulis atau diketik dan *plaintext* ini yang akan diproses menggunakan algoritma *cryptography* agar menjadi *chipertext*.
6. *Cryptanalysis*; bisa diartikan sebagai analisis sandi suatu ilmu untuk mendapatkan *plaintext* tanpa harus mengetahui kunci secara wajar. Jika suatu *chipertext* berhasil menjadi *plaintext* tanpa menggunakan kunci yang sah. (*Tri Puji Rahayu ; 2012 : 144*)

II.3. Algoritma *Run Length Encoding*

Algoritma *Run Length Encoding* adalah melakukan kompresi dengan memindahkan pengulangan *byte* yang sama berturut-turut atau secara terus menerus. Algoritma ini digunakan untuk mengompresi citra yang memiliki kelompok-kelompok piksel yang berderajat keabuan yang sama. Pada metode ini dilakukan pembuatan rangkaian pasangan nilai (P,Q) untuk setiap baris piksel, dimana nilai P menyatakan nilai derajat keabuan, sedangkan nilai Q menyatakan jumlah piksel berurutan yang memiliki derajat keabuan tersebut. Sebagai contoh sebuah citra dengan nilai piksel ”120, 120, 120, 120, 150, 200, 200, 200, 200, 150, 150, 150, 150, 120, 150, 150, 150, 150”, nilai

piksel pertama 120,kedua 120, karakter ketiga 120, karakter keempat 120, dan karakter kelima 150, dikarenakan pada nilai piksel kelima tidak sama dengan sebelumnya, sehingga 4 nilai piksel pertama yang mengalami perulangan akan dijumlahkan semuanya dan nilai yang dijumlahkan adalah banyaknya nilai piksel yang akan diulang, sehingga *output* keempat nilai piksel yang pertama setelah dikompresi adalah hanya sebuah nilai piksel dan diikuti dengan nilai perulangan yaitu ”120,4”, setelah dilakukan kompresi 4 piksel pertama akan dilanjutkan ke nilai berikutnya yaitu nilai kelima 150, kemudian nilai keenam 200, dikarenakan nilai piksel keenam tidak sama dengan nilai piksel kelima, dan nilai piksel kelima tidak mengalami perulangan sehingga nilai piksel yang tidak mengalami perulangan akan ditambahkan kepada nilai piksel ”120,4”, sehingga menjadi ”120,4150”, dan seterusnya sehingga nilai piksel citra ” 120, 120, 120, 120, 150, 200, 200, 200, 200, 150, 150, 150, 150, 120, 150, 150, 150, 150” setelah dikompresi akan menjadi ”(120,4) (150,1) (200,4) (150,4) (120,1) (150,4)”. Dapat diperhatikan bahwa nilai piksel ” 120, 120, 120, 120, 150, 200, 200, 200, 200, 150, 150, 150, 150, 120, 150, 150, 150, 150” yang berukuran 18 *byte* / nilai piksel. (*Muhammad Sandi ; 2015 : 78-79*)

II.4. SMS (*Short Message Service*)

Pertukaran pesan melalui *Short Message Service* (SMS) merupakan suatu layanan yang popular dikalangan pemakai telepon bergerak di Indonesia. Pengiriman SMS dari satu perangkat ke perangkat lainnya melalui SMS Center untuk menyimpan dan menyampaikan SMS ke perangkat tujuan. (*Hendra ; 2012 : 1*)

II.4.1. Keamanan SMS

GSM menyediakan mekanisme keamanan untuk memastikan kerahasiaan dan integritas dari layanan, mekanisme ini ditempatkan antara perangkat bergerak dengan jaringan operator dengan menggunakan algoritma A5, tetapi berdasarkan percobaan serangan kriptanalisis secara realtime terhadap keluaran algoritma A5/1 selama percakapan dua menit, kunci rahasia dapat dipecahkan dalam waktu satu detik, dan pada serangan kedua membutuhkan output dari algoritma A5/1 selama dua detik dari percakapan, dan kunci rahasia berhasil dipecahkan dalam waktu beberapa menit. (*Hendra ; 2012 : 2*)

II.4.2. Pengiriman SMS

Ada dua metode untuk mengirim sebuah pesan *text* ke suatu perangkat bergerak, yaitu melalui suatu perangkat bergerak ataupun melalui suatu *External Messaging Entities* (ESMEs). ESMEs terdiri dari sejumlah besar perangkat berbeda dan memiliki berbagai antarmuka seperti email, portal *messaging* berbasis web yang terkoneksi pada jaringan telepon bergerak melalui internet maupun kanal *dedicated* tertentu. Pesan awalnya dikirim ke suatu server yang

menangani trafik SMS yang dikenal sebagai *Short Messaging Service Center* (SMSC). Suatu *provider* yang mendukung pesan text harus memiliki paling sedikit satu SMSC pada jaringan mereka. SMSC perlu untuk menentukan bagaimana pesan disampaikan ke perangkat target. SMSC menanyakan kepada suatu basis data *Home Location Register* (HLR) yang menyimpan data pemakai dan informasi lokasi. Melalui interaksi dengan elemen lainnya, HRL menentukan routing informasi ke tujuan. Jika SMSC menerima balasan bahwa target tidak dapat dicapai, maka pesan akan disimpan untuk dikirim nantinya, jika sebaliknya maka akan dibalas dengan alamat *Mobile Switching Center* (MSC) yang tersedia untuk melayani. Ketika suatu pesan tiba dari SMSC ke MSC, MSC menanyakan kepada suatu basis data *Visitor Location Register* (VLR) yang akan mengembalikan suatu duplikat informasi dari perangkat target ketika dia tidak berada pada HLR-nya. MSC kemudian mengirim pesan kepada *Base Station* (BS) untuk disampaikan ke target. (*Hendra ; 2012 : 2*)

II.5. *Java*

Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan *Java* 2 adalah generasi kedua dari *Java platform* (generasi awalnya adalah JDK atau *Java Development Kit*). *Java* inilah yang berdiri diatas mesin *interpreter* yang diberi nama *Java Virtual Machine*(JVM). JVM inilah yang akan membaca *bytecode* dalam *file .class* dari suatu program sebagai representasi langsung program yang berisi bahasa mesin". Oleh karena itu bahasa java disebut juga sebagai bahasa pemrograman yang *portable* karena dapat dijalankan sebagai

sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM. (*Utomo Budiyanto ; 2011 : 27*)

Sun Microsystems telah mendefinisikan tiga *platform java* menurut *Utomo Budiyanto 2011* yang masing – masing diarahkan untuk tujuan tertentu dan untuk lingkungan komputasi yang berbeda-beda:

1. *Java Standard Edition* (J2SE), adalah inti dari bahasa pemrograman *java*. JDK adalah salah satu *tool* dari J2SE untuk mengkompilasi program *java* pada JRE.
2. *Java Enterprise Edition* (J2EE), dengan *built-in* mendukung untuk *servlets*, JSP, dan XML, edisi ini ditujukan untuk aplikasi berbasis *server*.
3. *Java Micro Edition* (J2ME), didesain untuk meletakkan perangkat lunak *java* pada barang elektronik beserta perangkat pendukungnya.

Teknologi *Java* mencangkup 2 elemen penting yaitu bahasa pemrograman (*programming language*) dan lingkungan aplikasi (*application environment*). *Java* sebagai bahasa pemrograman dapat diartikan bahwa *java* sebanding dengan bahasa pemrograman seperti C++, *Pascal, Visual Basic*, dan lainya, sedangkan *Java* sebagai lingkungan aplikasi berarti bahwa *java* dapat berjalan pada berbagai lingkungan seperti *browser*(*Applets*), *server*(*servlets* dan JSP) dan pada *mobile device*(*midlet* dan WAP). *Java* dalam hal ini mengungguli bahasa lainnya yang pernah ada jika dilihat dari sisi teknologi *mobile*. Hal ini dibuktikan dengan banyaknya jenis telepon genggam yang menggunakan *java* sebagai fitur utamanya. *Microsoft.NET mobile* pun kelihatannya belum dapat menyaingi keunggulan *Java* dalam bidang aplikasi *mobile*. Perlu diketahui bahwa *Microsoft* hanya mengandalkan solusi WAP yang mengembangkan ASP.NET untuk

kebutuhan *mobile device*, sedangkan *Java* memiliki 2 solusi yaitu WAP dan MIDP (*Mobile Information Device Profile*). Solusi pertama adalah dengan mengandalkan J2EE (*Java* 2 *Enterprise Edition*) dengan produknya yang bernama JSP (*Java Server Pages*) dan *Java Servlets*. JSP dan *Servlets* ini digunakan untuk membentuk halaman WAP. Solusi kedua dengan menggunakan J2ME(*Java* 2 *Micro Edition*) MIDP dengan produknya yang bernama *Midlets*. *Midlets* inilah yang menjadi fitur andalan oleh beberapa jenis telepon genggam terbaru. (*Utomo Budiyanto ; 2011 : 27*)

II.5.1. J2ME (*Java 2 Micro Edition*)

Java Micro Editon atau yang biasa disebut J2ME adalah bagian dari J2SE, karena itu banyak pustaka yang ada pada J2SE dapat digunakan pada J2ME. Tetapi J2ME mempunyai beberapa pustaka khusus yang tidak dimiliki J2SE. Kelahiran *platform* J2ME timbul karena dibutuhkan adanya sebuah *platform* komputasi yang mengakomodasi piranti komputer elektronik dan *embedded*. Piranti ini dikelompokkan menjadi dua kategori, menurut *Utomo Budiyanto ; 2011 : 27* yaitu :

a. Personal, *piranti mobile* yang dapat digunakan untuk komunikasi melalui jaringan tertentu misalkan ponsel, Personal *Digital Assistant* (PDA), *Palm*, *Pocket* PC dan *organizer*.
b. Piranti informasi yang digunakan bersama dengan jaringan tetap, koneksi jaringan yang tidak putus-putus misalnya TV, internet dan sistem navigasi.

Kategori pertama mengarahkan piranti untuk tujuan khusus atau fungsi-fungsi tertentu yang terbatas dan tidak digunakan untuk mesin komputasi yang

serba guna. Kategori kedua diarahkan untuk piranti yang mempunyai kapabilitas yang lebih besar dengan fasilitas *user interface* yang lebih baik, kemampuan komputasi yang lebih besar. (*Utomo Budiyanto ; 2011 : 27*)

1. Keunggulan J2ME

Salah satu kelebihan *Java* yang paling signifikan adalah *run everywhere*. Dengan kelebihan ini, para pengembang yang sudah terbiasa mengembangkan aplikasi dalam bingkai kerja J2ME dan J2EE akan mampu bermigrasi dengan mudah untuk mengembangkan aplikasi J2ME. Selain itu, *Java* juga merupakan *platform* yang memiliki banyak keunggulan lain, keunggulan *Java* secara umum adalah :

a. *Multiplatform*, aplikasi J2ME bisa berjalan diatas banyak *platform* yang didalamnya terdapat JVM. Beberapa *platform* yang tersedia didalamnya terdapat JVM antara lain *Windows* CR, *Symbian*, *Embedded Linux* dan sebagainya.
b. *Robust*, kode-kode *Java* adalah kode-kode *robust*, karena *virtual machine* mengatur keamanan proses eksekusi aplikasi. *Java virtual machine* menyediakan *garbage collector* yang berfungsi mencegah kebocoran memory.
c. Terintegrasi dengan baik, J2ME bisa terhubung dengan *back-end* J2EE *server* dan *web services* dengan mudah karena menyediakan pustaka-pustaka API RMI dan *web services*.
d. Berorientasi obyek, *Java* merupakan salah satu bahasa pemrograman yang murni berorientasi obyek. Hal ini mempermudah dan mempercepat

pengembangan sistem yang dikembangkan dengan metode analisa dan desain berorientasi obyek. (*Utomo Budiyanto ; 2011 : 28-29*)

II.6. Sistem Operasi *Android*

II.6.1. *Android*

Android adalah sistem operasi untuk telepon seluler yang berbasis Linux, yang mencakup sistem operasi, *middleware* dan aplikasi. *Android* tidak terikat ke satu merek telepon seluler. *Android* menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri hingga dapat digunakan oleh berbagai peranti *mobile*. Beberapa fitur utama dari *Android* antara lain WiFi *hotspot*, *Multi-touch*, *Multitasking*, GPS, *support java*, mendukung banyak jaringan (GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, *Bluetooth*, Wi-Fi, LTE, and WiMAX) dan juga kemampuan dasar telepon seluler pada umumnya. (*Alicia Sinsuw ; 2013 : 2*)

II.6.2 Android OS

Android OS adalah sistem operasi yang berbasis *Linux*, sistem operasi *open source.* Selain *Android Software Development Kit (SDK)* untuk pengembangan aplikasi, *android* juga tersedia bebas dalam bentuk sistem operasi. Hal ini yang menyebabkan *vendor-vendor smartphone* begitu berminat untuk memproduksi *smartphone* dan komputer *tablet* berbasis *Android*. (*Alicia Sinsuw ; 2013 : 2*)

II.6.3 *Android* SDK (*Software Development Kit*)

Android SDK adalah *tools* API *(Aplication Programming Interface)* yang diperlukan untuk mengembangkan aplikasi pada *platform Android* menggunakan bahasa pemrograman *Java*. Beberapa fitur-fitur *Android* yang paling penting adalah mesin *Virtual Dalvik* yang dioptimalkan untuk perangkat *mobile*, *integrated browser* berdasarkan *engine open source WebKit*, Grafis yang dioptimalkan dan didukung oleh *libraries* grafis 2D, grafis 3D berdasarkan spesifikasi *opengl* ES 1.0 (Opsional akselerasi perangkat keras), kemudian *SQLite* untuk penyimpanan data (*database*). Fitur-fitur *android* lainnya termasuk media yang mendukung *audio*, *video*, dan gambar, juga ada *fitur bluetooth*, EDGE, 3G dan WiFi, dengan fitur kamera, GPS, dan kompas. Selanjutnya fitur yang juga turut disediakan adalah lingkungan *Development* yang lengkap dan kaya termasuk perangkat emulator, *tools* untuk *debugging*, profil dan kinerja memori, dan *plugin* untuk IDE *Eclipse*. (*Alicia Sinsuw ; 2013 : 2*)

II.6.4. *AVD (Android Virtual Device)*

Android Virtual Device merupakan emulator untuk menjalankan aplikasi *android*, yang tampilannya dapat dilihat pada gambar 1. Setiap AVD terdiri dari sebuah profil perangkat keras yang dapat mengatur pilihan untuk menentukan *fitur hardware emulator*. Misalnya, menentukan apakah menggunakan perangkat kamera, apakah menggunakan *keyboard* QWERTY fisik atau tidak, berapa banyak memori internal, dan lain-lain. AVD juga memiliki sebuah pemetaan versi *Android*, maksudnya kita menentukan versi dari *platform Android* akan berjalan pada emulator. Pilihan lain dari AVD, misalnya menentukan *skin* yang kita ingin

gunakan pada emulator, yang memungkinkan untuk menentukan dimensi layar, tampilan, dan sebagainya. Kita juga dapat menentukan SD *Card virtual* untuk digunakan dengan di emulator. (*Alicia Sinsuw ; 2013 : 2*)

Gambar II.1. Tampilan AVD
Sumber : (*Alicia Sinsuw ; 2013 : 3*)

II.7. UML (*Unified Modelling Language*)

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 kita 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. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: *Grady Booch OOD* (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: *metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock*, dsb. Masa itu terkenal dengan masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan. Dimulai pada bulan Oktober 1994 *Booch, Rumbaugh dan Jacobson*, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease *draft* pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5

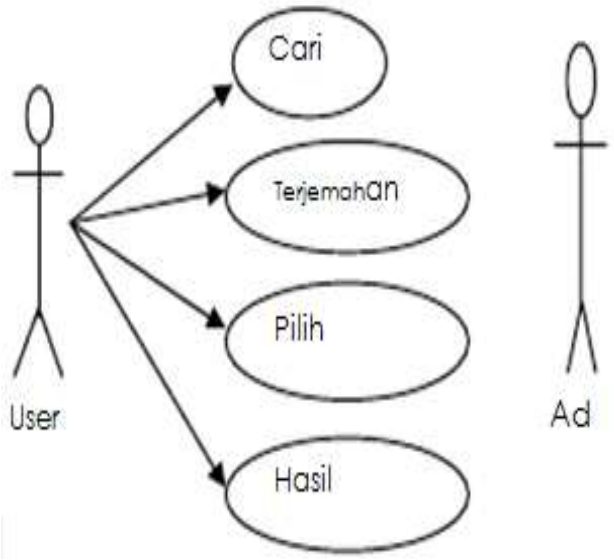
yang dirilis bulan Maret 2003. *Booch, Rumbaugh* dan *Jacobson* menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek. *(Yuni Sugiarti ; 2013 : 33)*

Dalam pembuatan skripsi ini penulis menggunakan diagram *Use Case* yang terdapat di dalam UML. Adapun maksud dari *Use Case* Diagram diterangkan dibawah ini.

1. *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/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case diagram* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. 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*

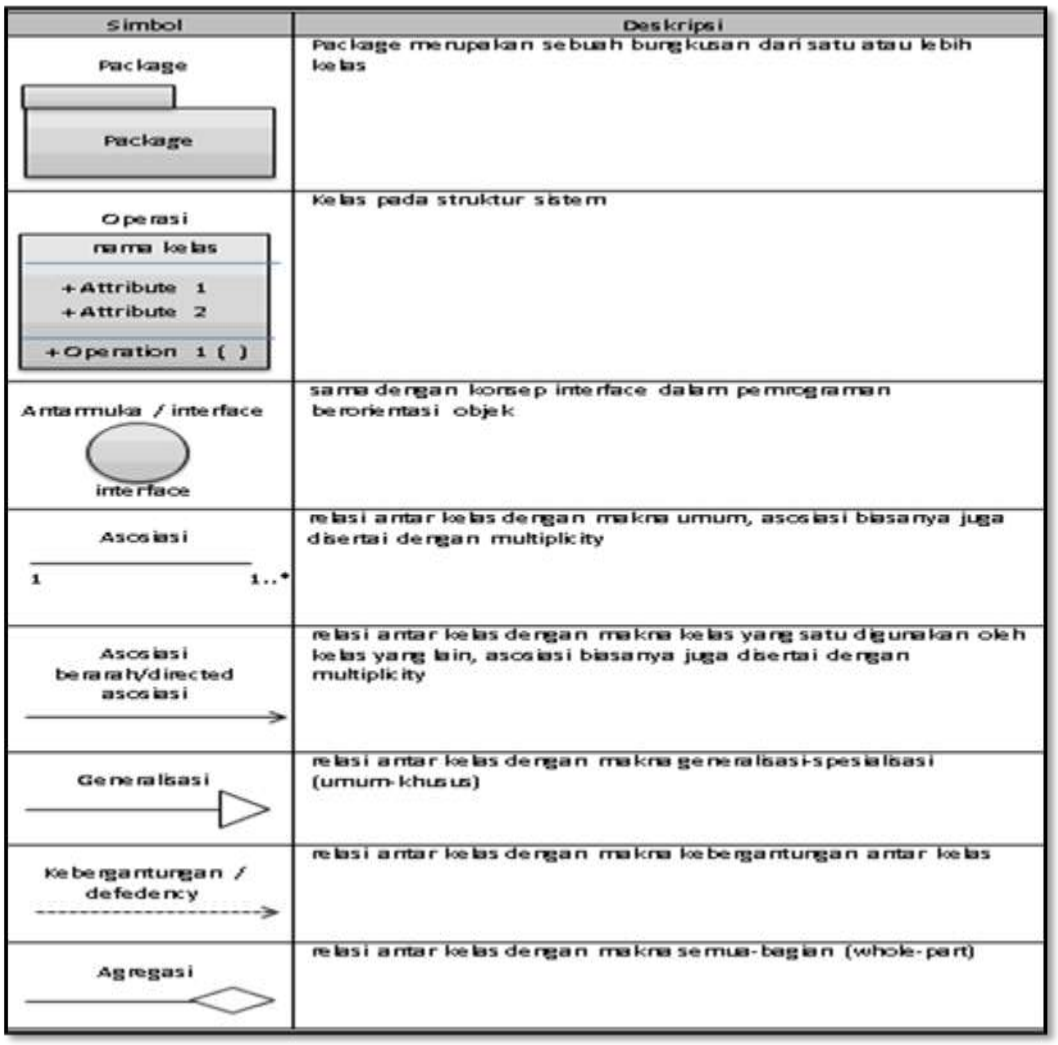
case juga dapat meng-*extend* *use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. *(Yuni Sugiarti ; 2013 : 41)*



Gambar II.2. *Use Case Diagram*
Sumber : (*Junaedi Siregar ; 2013 : 76*)

2. *Class Diagram*

Diagram kelas atau *class* diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Berikut adalah simbol-simbol pada diagram kelas :

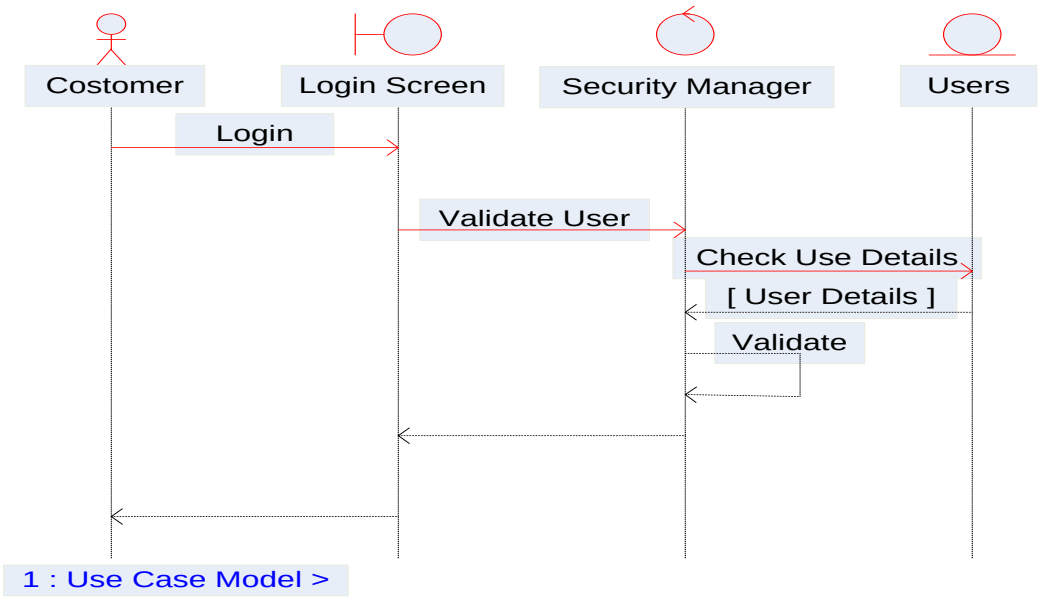


Gambar II.3. *Class* Diagram
Sumber : (Yuni Sugiarti ; 2013 : 59)

3. *Sequence Diagram*

Diagram *Sequence* menggambarkan kelakuan/prilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.



Gambar II.4. *Contoh Sequence Diagram*
Sumber : (Yuni Sugiarti ; 2013 : 63)

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.

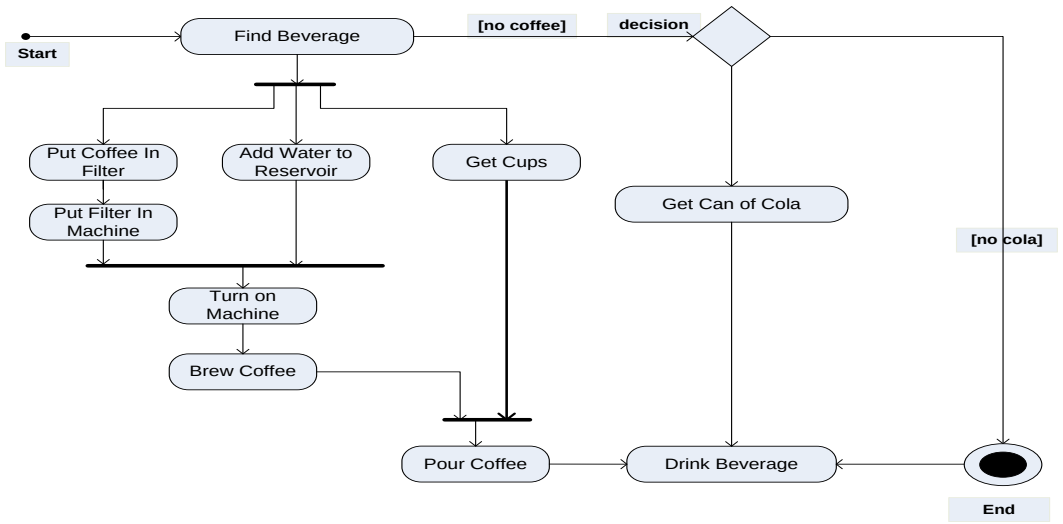
Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak

menggambarkan behaviour 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 dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

Activity diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.



Gambar II.5. *Activify* Diagram
Sumber : (Yuni Sugiarti ; 2013 : 76)