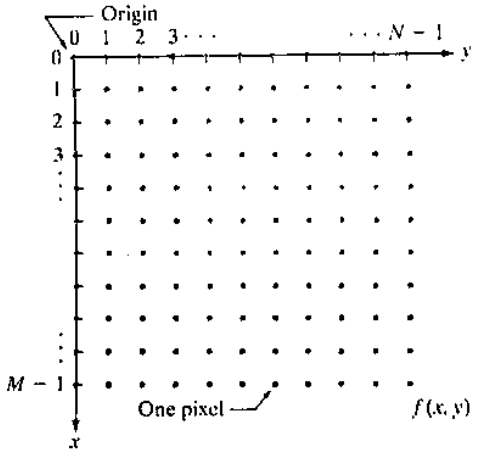
BAB II
TINJAUAN PUSTAKA

2.1 Pengolahan Citra Digital

Kebutuhan untuk pengolahan citra secara cepat dan mudah sangat diperlukan. Banyak sekali aplikasi program untuk editing gambar (*citra*) yang menyediakan fungsi-fungsi yang sudah tersedia, artinya pengguna tidak dapat melakukan kostumisasi terhadap fungsi-fungsi spesifik yang dibutuhkannya.
Penelitian ataupun penerapan di lapangan yang melibatkan proses pengolahan citra, kadang-kadang menyulitkan dalam bidang pemrograman karena rutin program yang berhubungan dengan komputer grafik membutuhkan keahlian khusus dalam implementasinya.

Suatu citra dapat didefinisikan sebagai fungsi *f(x,y)* berukuran M baris dan N kolom, dengan x dan y adalah koordinat spasial dan amplitudo *f* di titik koordinat *(x,y)* dinamakan intensitas atau tingkat keabuan dari citra pada titik tersebut. Apabila nilai x, y dan nilai amplitudo f secara keseluruhan berhingga (*finite*) dan bernilai diskrit maka dapat dikatakan bahwa citra tersebut adalah citra digital. Gambar 2.1 menunjukkan posisi koordinat citra digital.



Gambar 2.1. Koordinat citra digital

Citra digital dapat ditulis dalam bentuk matrik sebagai berikut[5].

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,0) & \cdots & f(0,N-1) \\ f(0,0) & f(0,0) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix} \quad (2.1)$$

Nilai pada suatu irisan antara baris dan kolom (pada posisi x,y) disebut dengan *picture elements, image elements, pels,* atau pixels. Istilah terakhir (pixel) paling sering digunakan pada citra digital.

2.1.1 Jenis Citra

Nilai suatu pixel memiliki nilai dalam rentang tertentu, dari nilai minimum sampai nilai maksimum. Jangkauan yang digunakan berbeda-beda tergantung dari jenis warnanya. Namun secara umum jangkauannya adalah 0-255. Citra dengan penggambaran seperti ini digolongkan ke dalam citra integer. Berikut adalah jenis-jenis citra berdasarkan nilai pixelnya[5].

2.1.1.1 *Citra Warna*

Setiap pixel dari citra warna (8 bit) hanya diwakili oleh 8 bit dengan jumlah warna maksimum yang dapat digunakan adalah 256 warna. Ada dua jenis citra warna 8 bit. Pertama, citra warna 8 bit dengan menggunakan palet warna 256 dengan setiap paletnya memiliki pemetaan nilai (*colormap*) RGB tertentu. Model ini lebih sering digunakan. Kedua, setiap pixel memiliki format 8 bit sebagai berikut[5].

Bit-7	Bit-6	Bit-5	Bit-4	Bit-3	Bit-2	Bit-1	Bit-0
R	R	R	G	G	G	B	B



Gambar 2.2. Citra warna

2.1.1.2 *Citra Grayscale*

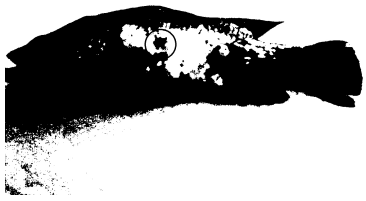
Citra *grayscale* merupakan citra digital yang hanya memiliki satu nilai kanal pada setiap pixelnya, dengan kata lain nilai bagian red = green = blue. Nilai tersebut digunakan untuk menunjukkan tingkat intensitas. Warna yang dimiliki adalah warna dari hitam, keabuan dan putih. Tingkatan keabuan disini merupakan warna abu dengan berbagai tingkatan dari hitam hingga mendekati putih. Citra *grayscale* berikut memiliki kedalaman warna 8 bit (256 kombinasi warna keabuan)[5].



Gambar 2.3. Citra *grayscale*

2.1.1.3 *Citra Biner*

Citra biner adalah citra digital yang hanya memiliki dua kemungkinan nilai pixel yaitu hitam dan putih. Citra biner juga disebut sebagai citra B&W (*black and white*) atau citra monokrom. Hanya dibutuhkan 1 bit untuk mewakili nilai setiap pixel dari citra biner. Citra biner sering kali muncul sebagai hasil dari proses pengolahan seperti segmentasi, pengambangan, morfologi, ataupun dithering[5].



Gambar 2.4. Citra biner

2.1.2 Segmentasi Citra

Segmentasi merupakan teknik untuk membagi suatu citra menjadi beberapa (*region*) dimana setiap daerah memiliki kemiripan atribut. Salah satu teknik segmentasi ialah pengambangan (*thresholding*).

2.1.3 Pengambangan (*Thresholding*)

Proses pengambangan akan menghasilkan citra biner, yaitu citra yang memiliki dua nilai tingkat keabuan hitam dan putih. Secara umum proses pengambangan citra *grayscale* untuk menghasilkan citra biner adalah sebagai berikut[5].

$$g(x,y) = \begin{cases} 1 \; if \; f(x,y) \geq T \\ 0 \; if \; f(x,y) < T \end{cases} \tag{2.2}$$

Dengan $g(x,y)$ adalah citra biner dari citra *grayscale* $f(x,y)$ dan T menyatakan nilai ambang. Nilai T memegang peranan yang sangat penting dalam proses pengambangan. Kualitas hasil citra biner sangat tergantung pada nilai T yang digunakan.

Terdapat dua jenis pengambangan, yaitu pengambangan global (*global thresholding*) dan pengambangan secara lokal adaptif (*locally adaptive thresholding*). Pada pengambangan global, seluruh pixel pada citra dikonversikan menjadi hitam atau putih dengan satu nilai ambang T. Kemungkinan besar pada pengambangan global akan banyak informasi hilang karena hanya menggunakan satu nilai T utuk keseluruhan pixel. Untuk mengatasi ini dapat digunakan pengembangan secara lokal adaptif. Pada pengambangan lokal, suatu citra dibagi menjadi blok-blok kecil yang kemudian dilakukan pengambangan lokal pada setiap blok dengan nilai T yang berbeda.

2.1.4 Pengambangan dengan metode Otsu

Pengambangan dengan Metode Otsu yaitu proses menghitung nilai ambang T secara otomatis berdasarkan citra masukan. Pendekatan yang dilakukan oleh metode Otsu adalah dengan melakukan analisis diskriminan yaitu menentukan

suatu variabel yang dapat membedakan antara dua atau lebih kelompok yang muncul secara alami. Analisis Diskriminan akan memaksimumkan variabel tersebut agar dapat memisahkan objek dengan latar belakang.

Misalkan nilai ambang yang akan dicari dari suatu citra *graycale* dinyatakan dengan k. Nilai k berkisar antara 1 sampai dengan L, dengan nilai L = 255. Probabilitas setiap *pixel* pada *level* ke i dapat dinyatakan dengan[6]:

$$P_i = n_i/N \tag{2.3}$$

dengan:

n_i menyatakan jumlah *pixel* pada *level* ke *i*

N menyatakan total jumlah *pixel* pada citra.

Nilai momen kumulatif ke nol, momen kumulatif ke satu dan nilai rata-rata berturut-turut dinyatakan sebagai berikut[6].

$$\omega(k) = \sum_{i=1}^{k} P_i \tag{2.4}$$

$$\mu(k) = \sum_{i=1}^{k} i.P_i \tag{2.5}$$

$$\mu_T = \sum_{i=1}^{L} i.P_i \tag{2.6}$$

Nilai ambang k dapat ditentukan dengan memaksimalkan persamaan[6]:

$$\sigma_B^2(k^*) = \max_{1 \le k \le L} \sigma_B^2(k) \tag{2.7}$$

dengan

$$\sigma_B^2(k) = \frac{[\mu_T \omega(k) - \mu(k)]^2}{\omega(k)[1-\omega(k)]} \tag{2.8}$$

2.2 Ikan Kakap

Kakap putih sebenarnya adalah ikan liar yang hidup di laut. Namun setelah di lakukan penelitian kakap putih memiliki habitat yang sangat luas. Kakap putih dapat hidup di daerah laut yang berlumpur, berpasir, serta di ekosistem mangrove.

2.2.1 Jenis Penyakit Ikan Kakap

Jenis penyakit bakterial yang ditemukan pada ikan kakap, diantaranya adalah penyakit borok pangkal sirip ekor dan penyakit insang. Hasil isolasi dan identifikasi bakteri ditemukan beberapa jenis bakteri yang diduga berkaitan erat dengan kasus penyakit bakterial, yaitu *Vibrio alginolyticus*, *V. algosus*, *V. anguillarum* dan *V. fuscus* (Jhonny, 2002). Adapun Jenis penyakit ikan kakap, diantaranya;

1. ***Monogenia,***

   Monogenia adalah parasit jenis kutu ikan dari golongan *crustacean* biasanya menyerang pada permukaan tubuh ikan cirinya yaitu luka pada tubuh. Cara penanganannya ikan yang terkena parasit dapat direndam dalam air tawar selama 10 – 15 menit, lalu direndam dengan larutan akriflavin 10 ppm selama 1 jam,prefuran 1 ppm selama 1 jam. Seperti yang dapat dilihat pada gambar 2.5.



Gambar 2.5 *Monogenia.*

2. *Trematoda*

Trematoda adalah cacing pipih, biasanya menyerang insang cirinya nya warna insang merah pucat. Cara penanganannya ikan yang sakit direndam larutan formalin 30-50 ppm selama 24 s/d48 jam atau larutan formalin 100-200 ppm selama 30 menit s/d 1 jam dan perendaman H_2O_2 200 ppm selama 1 jam. Seperti yang dapat dilihat pada gambar 2.6.



Gambar 2.6 *Trematoda.*

3. *Cryptocaryon*

Cryptocaryon penyakit bintik putih. Bagian yang diserang di bagian ekor. Cara penanganannya di rendam dalam air laut yang sudah diberi larutan formalin selama 5-7 hari, dosis 25 ppm. Seperti yang dapat dilihat pada gambar 2.7.



Gambar 2.7 *Cryptocaryon.*

4. *Tricodiniasis*

Tricodiniasis penyakit ini biasanya menyerang sirip ikan penyebarannya melalui perairan atau menular dari ikan lain. Cirinya sirip sobek-sobek dan lender meningkat. Cara penanganannya dengan cara direndam dalam air laut yang sudah diberi formalin 25-30 ppm selama 1-2 hari. Seperti yang dapat dilihat pada gambar 2.8.



Gambar 2.8 *Tricodiniasis.*

5. Bakteri *Vibrio* sp

Bakteri *Vibrio* sp penyakit yang menyebabkan kematian terbesar saat pembesaran cirinya terdapat radang berwarna merah pada anus. Cara penanganannya ikan yang sakit direndam dengan prefuran dengan acriflavine. Seperti yang dapat dilihat pada gambar 2.9.



Gambar 2.9 Bakteri *Vibrio* sp.

2.3 Android

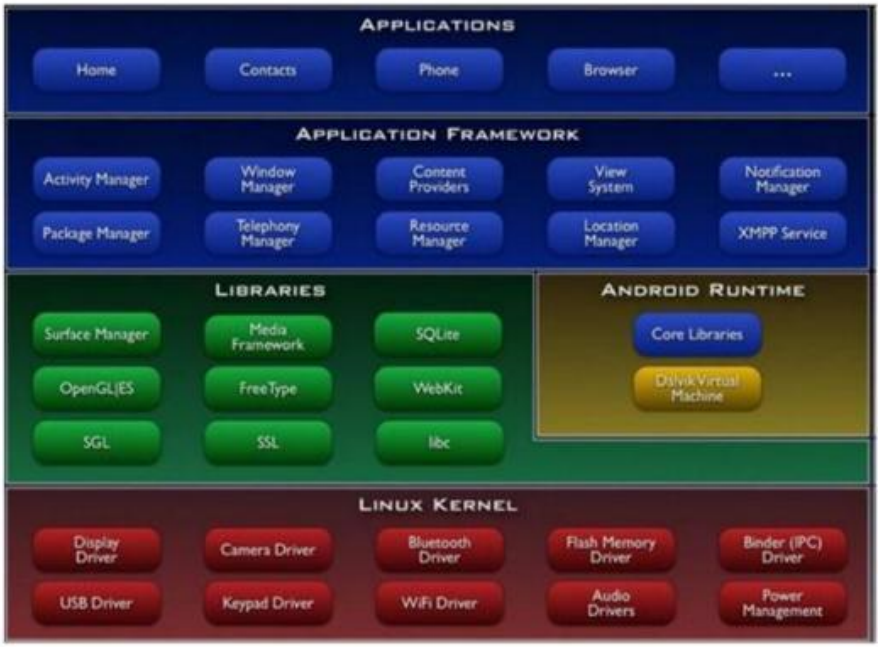
Android merupakan suatu *software stack* untuk *perangkat* mobile. Di dalamnya terdapat sistem operasi, *middleware,* dan *key application*. Aplikasi pada *platform* ini dikembangkan dalam bahasa pemrograman java. Android sendiri memiliki banyak fitur, diantaranya[7]:

1. Merupakan sebuah *Application Framework* sehingga *programmer* dapat menggunakan beberapa fungsi yang telah disediakan.
2. Dalvik *virtual machine*. Tiap aplikasi dalam Android memiliki *instance virtual machine* yang dapat bekerja secara efisien dalam lingkungan memori yang terbatas.
3. *Integrated browser.* Web browser berbasis WebKit e*ngine* terdapat pada browser *default* Android ataupun dapat diintegrasikan dengan aplikasi lain.
4. *Optimized graphic. Library* grafis 2D yang kaya dan 3D berbasis OpenGL ES 1.0 yang mendukung akselerasi hardware.
5. SQLite. Basis data relasional yang ringan namun sangat *powerful.*
6. *Media Support.* Mendukung berbagai format audio, video dan gambar (MPEG, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
7. GSM *Telephony.* Mendukung fungsi komunikasi GSM.
8. Blutooth, EDGE, 3G dan WiFi. Mendukung komunikasi pada jaringan (tergantung hardware).
9. Kamera, GPS, kompas dan *accelerometer.* Mendukung berbagai fitur yang disediakan oleh *hardware*.

10. *Tools Developer* yang lengkap. Termasuk *device emulator, tools* untuk *debugging, profiling* memori dan performa, *plugin* untuk Eclipse IDE.

2.3.1 Arsitektur Android

Sistem operasi Android dibagi menjadi lima bagian dalam empat lapisan utama, seperti yang dapat di lihat pada gambar 2.5 di bawah ini.



Gambar 2.10 Arsitektur Android

Penjelasan masing-masing lapisan:

1. *Kernel* Linux: Merupakan *kernel* (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 yang 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 di kompilasi kedalam file Dalvik yang dapat dieksekusi). Dalvik merupakan mesin virtual spesial yang di rancang 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.

2.3.2 Struktur Aplikasi Android

Arsitektur Android mengadopsi konsep *component reuse*, yang memungkinkan untuk mempublikasikan dan berbagi *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 *nonintrusive* 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* ekternal.

2.4 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (*object-oriented programming* disingkat OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek,

dimana semua data dan fungsi dibungkus dalam *class-class* atau *object-object*. Setiap *object* dapat menerima pesan, memproses data, mengirim, menyimpan dan memanipulasi data. Beberapa *object* berinteraksi dengan saling memberikan informasi satu terhadap lainya. Masing-masing *object* harus berisikan informasi mengenai dirinya sendiri dan dapat dihubungkan dengan *object* yang lain. Konsep dasar dari Pemrograman berorientasi objek antara lain[5]:

1. Kelas (*Class*)

   Merupakan model yang berisi kumpulan *attribute* dan *method* dalam suatu unit untuk suatu 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 pemrograma berorientasi objek.

2. Atribut (*Attribute*)

   Attribute dari sebuah *class* adalah variabel global yang dimiliki sebuah *class*, *attribute* dapat memiliki hak akses *private*, *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 *classi* juga memiliki izin akses seperti halnya *attribute*.

4. Objek (*Object*)

   Merupakan perwujudan dari *class*, setiap *object* akan mempunyai *atrribute* 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 ribuan komponen elektronik, sistem mekanik dengan empat bauh ban, jok, stir kemudi dan sebagainya. Dengan abstraksi, suatu sistem yang kompleks dapat dipandang sebagai kumpulan subsistem-subsistem yang lebih sederhana, seperti halnya mobil merupakan suatu sistem yang terdiri atas berbagai subsistem, seperti subsistem kemudi, subsistem pengereman dan sebagainya.

6. Enkapsulasi (*Encapsulation*)

   Merupakan suatu mekanisme untuk menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar sistem dan sekaligus menyederhanakan penggunaan sistem tersebut. Memastikan pengguna sebuah *object* tidak dapat mengganti keadaan dalam dari sebuah *object* dengan cara yang 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 kepada 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 menggerakan 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.

2.5 UML

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML merupakan himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah 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 para 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. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan sistem.

2.5.1 Diagram UML

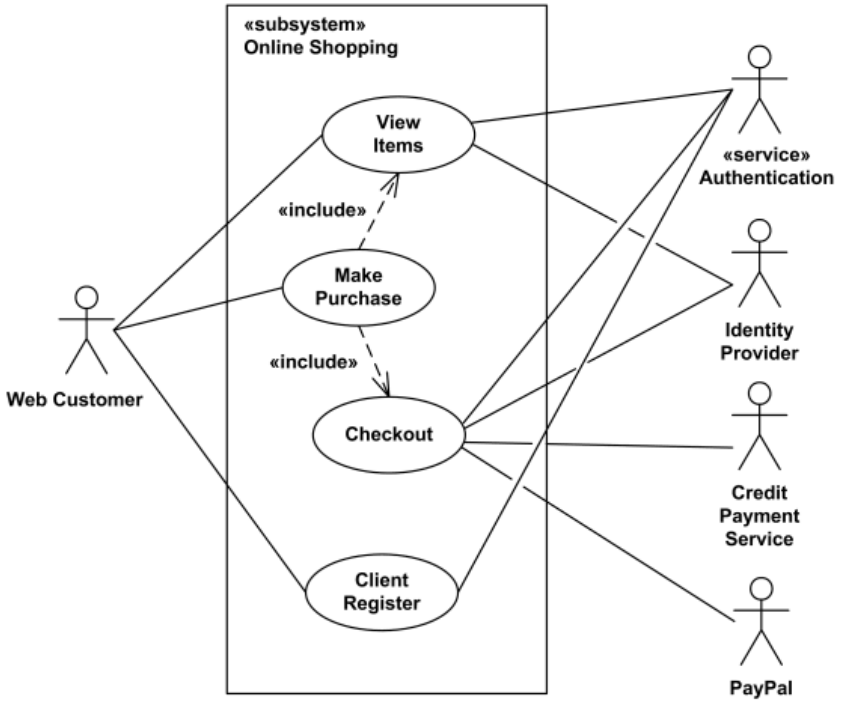
UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu[5]:

1. *Use Case Diagram,* untuk memodelkan proses bisnis.
2. *Conceptual Diagram,* untuk memodelkan konsep-konsep yang ada di dalam aplikasi.
3. *Sequence Diagram,* untuk memodelkan pengiriman pesan (*message*) antar objek.

4. *Collaboration Diagram,* untuk memodelkan interaksi antar objek.
5. *State Diagram,* untuk memodelkan perilaku objek di dalam sistem.
6. *Activity Diagram,* untuk memodelkan perilaku *Use Case* dan objek di dalam sistem.
7. *Class Diagram*, untuk memodelkan struktur kelas.
8. *Object Diagram*, untuk memodelkan struktur objek.
9. *Component Diagram,* untuk memodelkan komponen objek.
10. *Deployment Diagram,* untuk memodelkan distribusi aplikasi.

2.5.2 *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 anatara 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. 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. Penggambaran *use case diagram* dapat dilihat pada gambar 2.6 berikut.



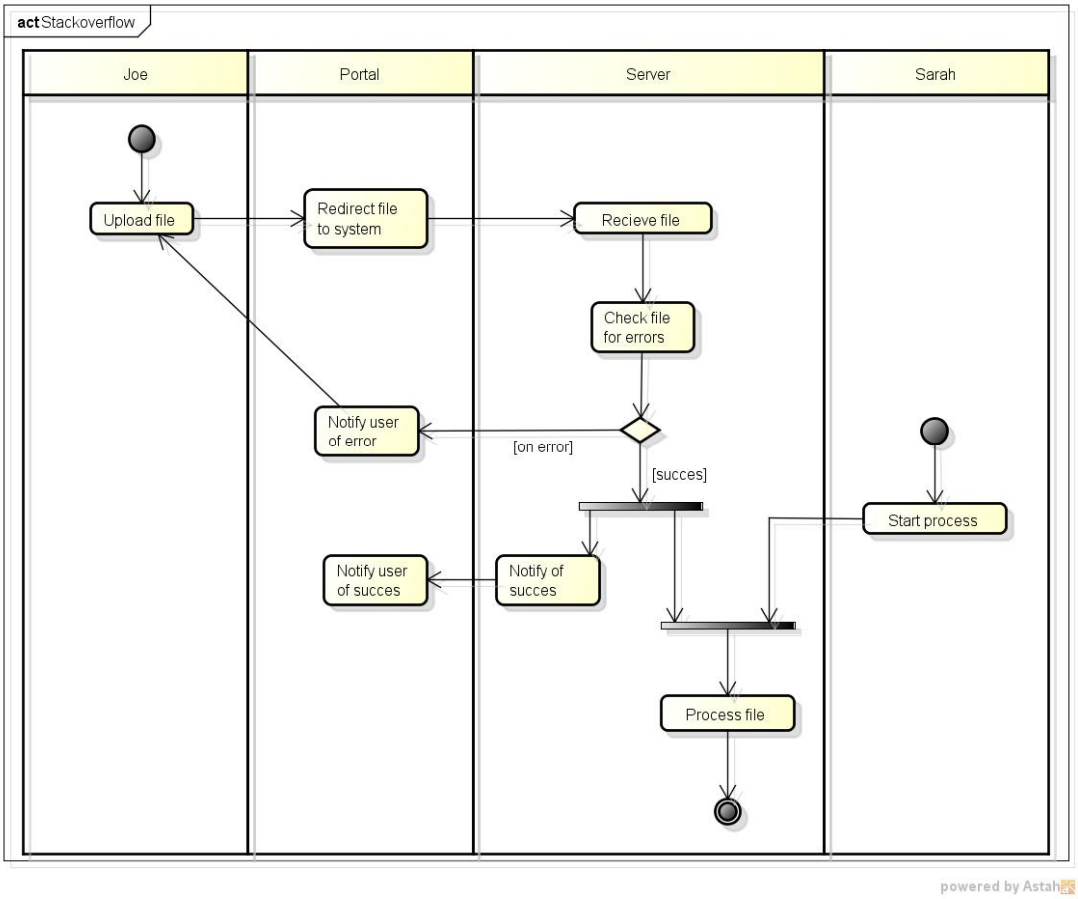
Gambar 2.11. Contoh Diagram *Use Case*

2.5.3 *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 diagrami* merupakan *state diagram* khusus, dimana 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. *Decision*

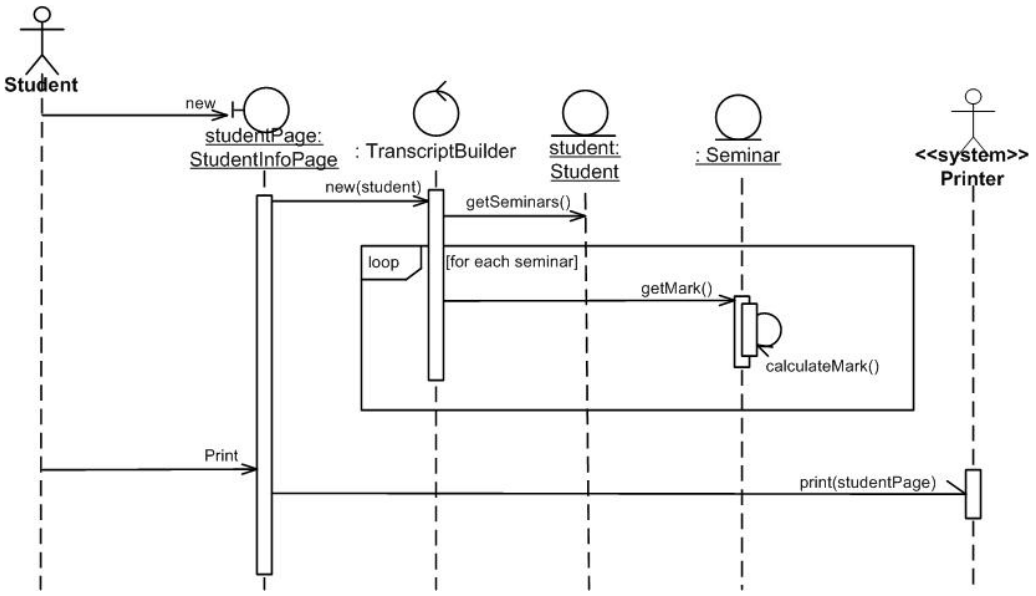
digunakan untuk menggambarkan *behaviour* pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa *object swimline* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu. Contoh *Activity* diagram dapat dilihat pada gambar 2.7 berikut.



Gambar 2.12. Contoh Diagram *Activity*

2.5.4 *Sequence* Diagram

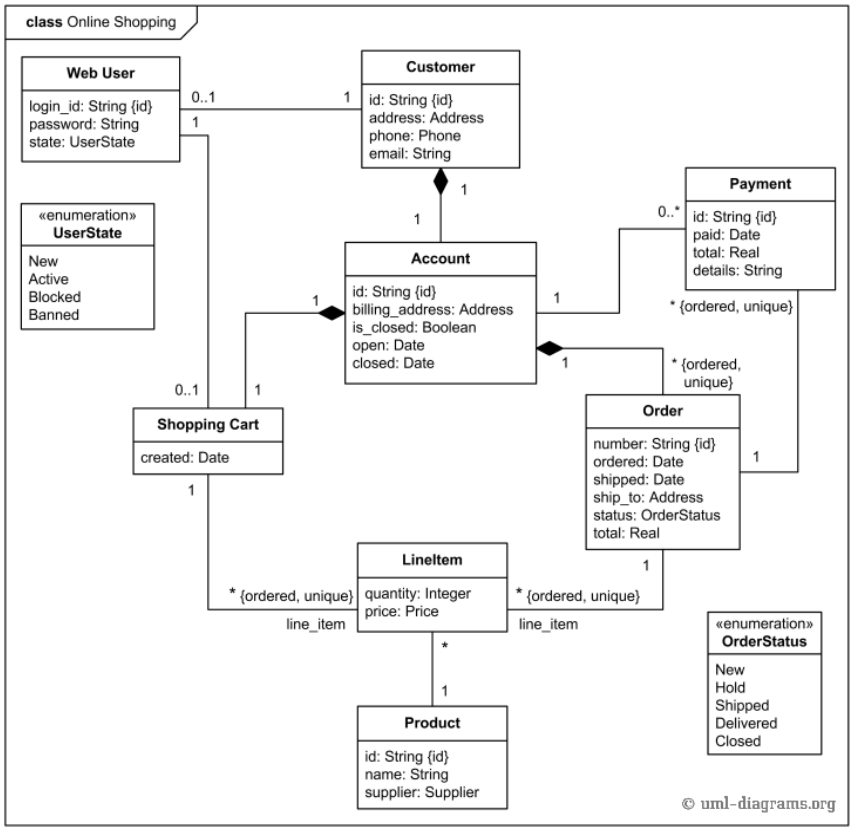
Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi dan informasi yang diperlukan oleh masing-masing operasi. Contoh *sequence* diagram dapat dilihat pada gambar 2.8 berikut.



Gambar 2.13. Contoh Diagram *Sequence*

2.5.5 *Class* Diagram

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. *Class* diagram menunjukan hubungan antar *class* dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan. Penggambaran *class diagram* dapat dilihat dalam gambar 2.9 berikut.



Gambar 2.14 Contoh Diagram *Class*

2.6 SaaS (Software as a service)

Saas (Software as a service atau perangkat lunak berbentuk layanan) adalah suatu model penyampaian aplikasi perangkat lunak oleh suatu vendor perangkat lunak yag mengembangkan aplikasi web yang dikepalai dan dioperasikan (baik secara mandiri maupun melalui pihak ketiga) untuk digunakan oleh pelanggannya melalui internet. Pelanggan tidak mengeluarkan uang untuk memiliki perangkat lunak tersebut melainkan hanya untuk menggunakan. Pelanggan menggunakan perangkat lunak tersebut melalui antarmuka pemrograman aplikasi (API) yang dapat diakses melalui web dan seringkali ditulis dengan layanan web atau REST.